Developing Efficient Background Processes for Windows

January 20, 2009

Abstract

This paper provides information to software developers about how to design efficient and secure services and tasks (collectively known as background processes) for the Windows® family of operating systems.

This paper summarizes background processes and the Service Control Manager (SCM) and Task Scheduler infrastructure in Windows for managing services and tasks. This paper also provides information about the improvements that have been made to this infrastructure in Windows 7, including information about how the APIs and tools have been updated to support these improvements.

This information applies for the following operating systems:  
 Windows 7  
 Windows Server® 2008  
 Windows Vista®

References and resources discussed here are listed at the end of this paper.

For the latest information, see:   
 [http://www.microsoft.com/whdc/system/pnppwr/powermgmt/BackgroundProcs.mspx](http://www.microsoft.com/whdc/)

Disclaimer: This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Change |  |  |  |
| January 16, 2009 | Added Service Trigger Events to Resources | | | |
| November 5, 2008 | First publication | | | |

Contents

[Background Processes Overview 4](#_Toc212885697)

[Optimizing Background Processes 5](#_Toc212885698)

[Performance and Responsiveness 5](#_Toc212885699)

[Energy Efficiency and Battery Life 6](#_Toc212885700)

[Security Attack Surface 6](#_Toc212885701)

[Services and the Service Control Manager 6](#_Toc212885702)

[Service States 7](#_Toc212885703)

[Service Management Utilities 8](#_Toc212885704)

[Sc.exe 8](#_Toc212885705)

[Services.MSC 8](#_Toc212885706)

[Optimizing Services 9](#_Toc212885707)

[Service Startup Types 9](#_Toc212885708)

[Demand Start by Using SCM Client API or COM Service Activation 10](#_Toc212885709)

[Stopping a Service When It Is Idle 10](#_Toc212885710)

[SCM Improvements in Windows 7 11](#_Toc212885711)

[Trigger Start Services 12](#_Toc212885712)

[SCM Service Triggers 13](#_Toc212885713)

[SCM Trigger Actions 14](#_Toc212885714)

[SCM Client API Support for Service Triggers 14](#_Toc212885715)

[Sc.exe Windows Command-Line Tool Support for Service Triggers 16](#_Toc212885716)

[Best Practices for Services Development 17](#_Toc212885717)

[Diagnosing Service Issues 18](#_Toc212885718)

[Diagnosing with Sc.exe 18](#_Toc212885719)

[Identifying a Service in a Shared Service Host 19](#_Toc212885720)

[Diagnosing with EventVwr.MSC 19](#_Toc212885721)

[Tasks and the Task Scheduler 20](#_Toc212885722)

[Task States 21](#_Toc212885723)

[Task Scheduler Management Utilities 21](#_Toc212885724)

[SchTasks.exe 21](#_Toc212885725)

[TaskSchd.MSC 21](#_Toc212885726)

[Optimizing Tasks 22](#_Toc212885727)

[Best Practices for Task Development 23](#_Toc212885728)

[Diagnosing Task Issues 23](#_Toc212885729)

[Diagnosing with TaskSchd.msc 23](#_Toc212885730)

[Diagnosing with EventVwr.msc 24](#_Toc212885731)

[Selecting the Appropriate Windows Background Process Model 25](#_Toc212885732)

[Next Steps 27](#_Toc212885733)

[Glossary 28](#_Toc212885734)

[Resources 29](#_Toc212885735)

[Service Resources 29](#_Toc212885736)

[Task Resources 30](#_Toc212885737)

[Other Resources 30](#_Toc212885738)

# Background Processes Overview

Background processes are processes that generally run without requiring user interaction, and their activity can span across multiple user sessions. In Windows® operating systems, a background process is typically implemented either as a Windows Service or as a Windows Task. In UNIX-based operating systems, background processes are generally known as daemon processes.

Background processes are essential providers of important system and user functionality. The purpose of a background process can vary from device, user session, or security management to system maintenance. On server systems, background processes may be services that run a file server, a Web server, or a database server. On client systems, background processes can vary from device management services to tasks that run on a schedule or on system startup. For example, a defragmentation task might be scheduled to run once a week to defragment the system hard disks. In other cases, an information technology (IT) administrator might deploy system security-scanning or log-collection processes that run continuously or intermittently as required.

Background processes also provide centralized resource management and sharing services to ensure that applications that depend on a common resource, such as a device or a database, do not cause conflicts in their requests. In this situation, a background process typically operates as a server, handling requests from multiple client applications, and appropriately allocates and shares the resources that it manages.

Most background processes are active on the system for an extended period of time. An example of a long-running process is a virus-scanning service that runs in the background, constantly monitoring the system for suspicious virus activity. Other background processes run only when a certain event of interest occurs. For example, an image archiving process runs only when a camera or other imaging device is attached to the system. Because background processes are generally active for long periods of time, they should be designed for efficiency so that they do not adversely affect the overall system performance or power consumption.

The Windows operating system features built-in infrastructure that helps manage background processes. Windows takes advantage of this built-in infrastructure to provide native system management, device management, and system maintenance features. The two background process infrastructure components that are discussed in this paper are the Windows Service Control Manager (SCM) and the Windows Task Scheduler.

Significant improvements have been made to this infrastructure for Windows 7. Third-party software developers can use these improvements to help them develop Windows services that are efficient and that contribute to the overall system performance and security.

Table 1 lists several examples of the many background processes in the Windows operating system.

Table 1: Example Windows Background Processes

|  |  |  |
| --- | --- | --- |
| Background process | Process type | Description |
| Windows Security Center | Windows Service | Monitors the system security settings and configuration. |
| Windows Event Log | Windows Service | Manages events and event logs. Enables event queries, subscriptions, and event log archiving. |
| Windows Error Reporting QueueReporting | Windows Task | Processes queued Windows Error Reporting reports. |
| Windows ScheduledDefrag | Windows Task | Defragments the system's hard disk drives. |

# Optimizing Background Processes

Inefficient background processes can adversely affect system performance, responsiveness, energy efficiency, security, and ultimately the user experience. Developers should optimize their background processes to improve system performance, decrease power consumption, extend portable computer battery life, and help lessen potential security issues.

Developers should be aware of the following optimization opportunities when they design their background processes.

## Performance and Responsiveness

Many services are configured to start automatically at system startup and run until the system is shut down. These background processes can affect the performance of both system startup and system shutdown. Therefore, they must be able to initialize and shut down quickly. Additionally, background processes that run continuously can affect the amount of available system memory, disk space, and other system resources. Therefore, background processes should be started and enabled only when they are required to provide the associated functionality.

An issue that is typically ignored during the development and optimization of background processes is server consolidation. Server consolidation includes scenarios where one physical server system has virtualization enabled and hosts many virtual server or client systems. In this situation, the performance impact of a single background process is multiplied and software developers should optimize their background processes for such scenarios.

When developing a background process, software developers are encouraged to design their process so that it runs only when the relevant scenario occurs. A continuously executing background process should be used only if the relevant scenario frequently occurs throughout the time that the system is in operation.

As an example, the Bluetooth service runs only when at least one Bluetooth device is plugged into the system. Conversely, the Plug and Play service runs constantly to provide hardware device management services.

## Energy Efficiency and Battery Life

System manufacturers, developers, enterprises, and consumers are all driven to reduce the power consumption of desktops PCs and to help increase battery life on portable computers. Optimally designed background processes that do not run constantly contribute directly to reduce power consumption on both portable computers and desktop PCs, and therefore contribute to increased portable computer battery life.

As an example, the Windows Disk Defragmenter service runs only when a user requests the hard disks to be defragmented, and it runs automatically only when the system is using AC power.

## Security Attack Surface

It is common for background processes to require system security privilege (for example, LocalSystem or LocalService) to run across multiple user sessions and to manage system resources, including attached devices. Frequently, a background process either explicitly listens on the network or calls an API that causes the process to listen on the network. This may be a valid requirement for the background process to provide its intended functionality. However, it increases the risk of exploitation of a vulnerability that might not be known during the development of the background process.

If an attacker exploits a vulnerability in a background process that is executing with system security privilege, the attacker can potentially elevate to system privilege and gain full control of the system. This affects both security and privacy. If the background process runs all the time that the system runs, it significantly increases the chances of the background process being compromised. However, if the background process is designed to run only for a limited time when the relevant scenario occurs, the possibility of an attack that results in the system being compromised is significantly reduced.

Therefore, software developers should limit the privileges that are associated with their background processes to only those privileges that are required to provide the intended functionality. A background process should not run in LocalSystem, LocalService, or any other system security privilege context unless it is absolutely required. Even when such requirements exist, we strongly encourage software developers to design their background processes so that they are split into two components: one small process that allows only local on-machine access and that runs in a high-privilege context and another larger process that allows network access and that runs in a low-privilege context.

# Services and the Service Control Manager

Windows Services (“*services*”) are designed to run as noninteractive system processes in the background. Services respond to application and user requests and manage devices or other system resources. Specifically, a Windows service is an application that is implemented by using the Windows Services programming model. Unlike typical user applications, services typically handle low-level tasks that require little or no user interaction.

Although services are typically invisible to end users, many scenarios would not be possible without services. Several essential operating system features are handled by services, including:

* Networking. Several system services support networking. For example, the Dynamic Host Configuration Protocol (DHCP) client service manages network configuration by registering and updating IP addresses for the system.
* Hardware. The Plug and Play service enables a computer to recognize and respond to changes in its hardware configuration, such as when a user adds or removes hardware while the system is running.
* Remote access. Terminal Services lets users log on to a computer from a remote location.

In addition to the services that are included with the Windows operating system, most computers also have several third-party applications that run partly or completely as services. Common examples of third-party services include network firewalls and antivirus applications.

The primary difference between services and typical applications is that services are managed by the SCM. Services are implemented by using the services API, which handles the interaction between the SCM and the service itself. The SCM maintains a database of installed services and provides a unified way to control the services, including:

* Starting and stopping services.
* Managing services that are running.
* Maintaining service-related state information.

## Service States

Services exist in one of three primary states—*RUNNING*, *STOPPED*, or *PAUSED*—as described in Table 2.

Table 2: Primary Service Running States

|  |  |
| --- | --- |
| Service state | Description |
| RUNNING | This is the typical running state for a service. The service is executing and responding to system events and client requests.  **Note:** The Sc.exe command-line tool reports this state as *RUNNING*, whereas the services Microsoft Management Console (MMC) snap-in—services.msc—reports this state as *Started*. |
| STOPPED | A stopped service has been completely shut down and must follow a normal startup procedure to enter the RUNNING state. The service does not respond to system events or client requests. The executable image for the service is not loaded or initialized. |
| PAUSED | A paused service suspends normal processing, but remains in memory and continues to respond to control requests. Paused services can return to the started state without going through the startup procedure. |

After a service is started, the SCM uses *control requests* to manage the service's state or to send notifications to the service of system-wide events. For example, the SCM sends control requests to notify a service that it is pausing, is resuming operation, or should be preparing to shut down. It also can notify a service when the machine is entering a sleep state.

The SCM's database also contains the security settings for each installed service. These settings control how much access each service has to system resources and enables system administrators to control the access to each service. For more information about these security settings, see “Service Security and Access Rights.”

## Service Management Utilities

Windows includes a command-line utility—sc.exe—that can be used to control a service. Its commands correspond to the functions that the SCM provides. Windows also includes an MMC snap-in—services.msc—that provides basic SCM functionality through a graphical user interface.

Developers and administrators should never edit the registry to configure service settings. Instead, they should use sc.exe, the services MMC snap-in, or the SCM client API.

### Sc.exe

Sc.exe is the SCM command-line tool in Windows that lets you create, delete, start, or stop a service, as well as query or modify the configuration of a service.

The following is an example of using sc.exe to query the state of a service:

C:\>sc query bits

SERVICE\_NAME: bits

TYPE : 20 WIN32\_SHARE\_PROCESS

STATE : 4 RUNNING

(STOPPABLE, NOT\_PAUSABLE, ACCEPTS\_SHUTDOWN)

WIN32\_EXIT\_CODE : 0 (0x0)

SERVICE\_EXIT\_CODE : 0 (0x0)

CHECKPOINT : 0x0

WAIT\_HINT : 0x0

### Services.MSC

The services MMC snap-in—services.msc—provides a graphical user interface that lets users or administrators perform basic control operations on services. These operations include displaying the names and descriptions of all the services that are installed in the system, starting or stopping a service, setting the startup type for a service, and specifying the account on the system in which a service runs.

An example of the services MMC snap-in is shown in Figure 1.

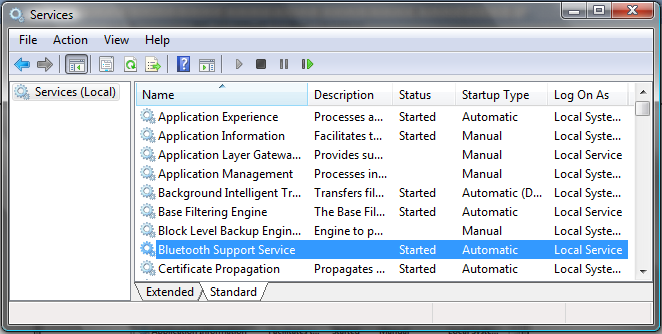


Figure 1: Services MMC snap-in

## Optimizing Services

The following sections discuss some options for services that can be used to optimize service performance. The key options that are discussed here are service startup types, starting services on demand by using the SCM client API or COM service activation, and stopping services when they are idle.

### Service Startup Types

The SCM defines different service startup types that correlate to different points in the system boot process. Developers of a service should select the appropriate service startup type based on the functionality that the service provides.

* **Automatic** specifies that a service should be started immediately after the system has completed starting the operating system.
* A service that uses the Automatic startup type directly affects system boot performance because it must be started as part of the system boot sequence.
* The Automatic startup type should be used only if the service provides critical functionality that cannot wait until after the system has completed booting or if the service cannot be demand started.
* Examples of services that use the Automatic startup type are the services that provide the core networking infrastructure and functionality.
* **Automatic (Delayed Start**) specifies that a service should be started approximately 2 minutes after the system has completed starting the operating system.
* The Automatic (Delayed Start) startup type is preferred over the Automatic startup type because it helps reduce the effect on the system's overall boot performance.
* An example of a service that use the Automatic (Delayed Start) startup type is the Windows Update service that scans for applicable security updates for the system.
* **Manual** specifies that a service can be started at any time during regular system operation or whenever a dependent service is started. A client application may start a service by calling the **StartService** SCM client API function. In Windows 7, a service can also be started on specific triggers, such as the addition of a particular hardware device.
* The Manual startup type should be used for services that are not required to run continuously. Services that provide functionality that is required rarely or less frequently should also use the Manual startup type and should be started on demand by using one of the methods discussed earlier.
* An example of a service that uses the Manual start type is the Windows Disk Defragmenter service that defragments hard disks.

Most services in the system perform activity only when one or more events of interest occur. This usually requires that the service use the Automatic startup type and run continuously, waiting and listening for certain events to occur. Setting a service to use the Automatic startup type adversely affects system performance because the service startup and shutdown times add to the system-wide startup and shutdown times. A service that runs constantly can also increase the potential security attack surface area of the system. The performance impact of multiple services in a system that run continuously adds up quickly to affect the key system performance metrics, including increased CPU, memory, and other resource usage.

### Demand Start by Using SCM Client API or COM Service Activation

A service can be started by using a service control program such as sc.exe. If an application must directly start a service when one or more scenarios of interest require the service, the application can start the service by calling the **StartService** SCM client API function. For an example of how to start a service from an application, see “Starting a Service.”

Services that are not required to run continuously can be started on demand by using the SCM client API or by using COM activation methods. Starting a service on demand helps optimize the overall system performance by starting the service only when it is required.

### Stopping a Service When It Is Idle

We highly recommend that a service be stopped after detecting that the service has not had any work to do for a certain period of time. In addition to starting a service only when it is required, stopping the service when it is no longer in use helps to further reduce the service footprint. The design of a service should consider an optimal idle time period for when it should be stopped based on the frequency of the service’s usage. Typically, 2 to 5 minutes is a reasonable idle time setting after which a service should be stopped.

The following are several things to consider when a service is stopping when it is idle:

* **Stopping the service**. It is simple for the service to stop itself. All it must do is run its shutdown sequence and then call the **SetServiceStatus** SCM client API function specifying a status of SERVICE\_STOPPED. This call can be made at any time while the service is running and, after it has been called, the SCM treats the service as having been stopped and thus allows future start requests.
* **Client calls**. Client applications must not fail due to the service being stopped when it is idle. The client API of the service should mask any failures that result from the service being stopped. Instead, if the service is stopped the client API should run the service's startup sequence by calling the **StartService** SCM client API function to enable servicing of all pending client requests. If low-privileged clients are required to start the service, then the service's Security Descriptor should be modified to allow service start by low-privileged clients. Note that COM services might not be required to do this because the COM runtime starts the service when **CoCreateInstance** is called.
* **Reference counting**. The service should reference count all client calls. For example, if the service's clients use remote procedure call (RPC), then incrementing the reference count on ”Open” and decrementing the reference count on ”Close” is a sufficient solution. The service should then use a timer to stop the service after the specified idle period expires after the reference count drops to zero. If the reference count becomes greater than zero, then the timer should be canceled. The thread pool provides APIs to create, set, and cancel timers. Note that the service must handle the race condition between the time that the shutdown sequence starts and a new client call is received. This can be handled gracefully by failing the client call with a specific error code and having the client-side API wait for the service to enter the stopped state. After the service is stopped, the client API should restart the service and retry the call to the service.

# SCM Improvements in Windows 7

The Windows 7 operating system includes significant improvements to the SCM. These improvements are designed to allow services to be started only when they are required and to help reduce the number of services that use Automatic Start.

To realize the vision of having services run only when they are absolutely required, the Windows SCM infrastructure was improved in Windows 7. Service developers should be aware of these improvements so that they can design their services to start on demand. The following section describes the improvements to the Windows SCM infrastructure for starting services only when they are required and provides guidance for developers on how to take advantage of these improvements to optimize the efficiency of their services.

The improvements in Windows 7 are designed to enable:

* **Demand Start of a service when a specific event occurs.** A service can register to be started or stopped on a supported event. These events are known as *triggers*. Starting a service that is based on a trigger is called a *Trigger Start*.
* **Demand Start of a service by using SCM client API or COM service activation.** A service can also be started on demand by calling an SCM client API function or by COM service activation when a client application requires functionality that is provided by the service. The infrastructure that supports this functionality was improved in Windows 7.
* **Stopping a service when idle:** A service can stop itself if it determines that it is not servicing any clients or performing any other activity for a specified amount of time.

Developers may design their service to use a combination of Demand Start and stopping when idle to further optimize service performance while still enabling service scenarios.

It should be noted that moving to Demand Start or Trigger Start may not be appropriate for all services. For example, a service that requires activity during most of the system’s uptime or that has a high frequency of intermittent activity should not move to a Demand Start or Trigger Start model. In these situations, it is appropriate to use Automatic Start.

The following section details Trigger Start support and the different triggers that are supported in Windows 7.

## Trigger Start Services

Windows 7 supports the ability to start or stop a service when specific event triggers occur. These events include the first IP address arrival, the last IP address departure, the joining and disjoining of the system to a domain, the connection of a hardware device, and Group Policy changes. A service can be designed to start or stop whenever a particular trigger event occurs.

An example is an imaging device service, which might be configured to start only when an imaging device such as a scanner or digital camera is connected to the system. In this situation, the service would use a hardware device arrival trigger that causes the SCM to immediately start the service. Similarly, if a service must be started only in domain environments, the service can be configured to start based on the domain join trigger. In this situation, millions of home computers that are not joined to a domain do not start the service.

Service triggers help enable services to start based on infrequent events and scenarios. The benefits of service triggers include the following:

* **Improved system boot performance**. The service does not use Automatic Start, which results in faster system startup.
* **Improved system shutdown performance**. The service is not started unless the service trigger occurs, which results in faster system shutdown.
* **Reduction in system attack surface**. Many services are network facing and run in system accounts, which contributes to the potential attack surface. Service trigger support lets services run only when they are required, thus reducing the ability of an attacker to probe the service at the time of their choosing and thus helping reduce the system attack surface against potential vulnerabilities.
* **Reduction in service footprint** **for scenarios where the service features are not in use**. Service triggers such as Group Policy and domain join enable services only in enterprises with domain environments and not in millions of home systems where they do not apply.

Figure 2 provides a simplified view of the SCM architecture that supports service triggers.

**Service 1**

**Subscribed to start on Device Connect trigger**

**Service 2**

**Subscribed to start on Domain join trigger, Stop on Domain disjoin trigger, and Start on GP trigger**

**Service Control Manager (SCM)**

**Device connect trigger**

**IP address trigger**

**Domain trigger**

**Group Policy (GP) trigger**

**Custom trigger**

Figure 2: SCM architecture supporting Trigger Start

### SCM Service Triggers

Table 3 describes the service triggers that are supported in the Windows 7 SCM.

Table 3: SCM Service Triggers

|  |  |
| --- | --- |
| Trigger | Description |
| IP address | The IP address trigger enables a service to be started on the first IP address arrival and stopped on the last IP address departure. The first IP address arrival is defined as the first preferred, non-loopback IPv4/IPv6 address that is indicated by the network stack. Similarly, the last IP address departure is defined as the last preferred, non-loopback IPv4/IPv6 address that is removed by the network stack.  Services that must function only when the system has at least one IP address, such as network-dependent services, should use this trigger. The IP address trigger enables the service to shut down on a system that does not have an IP address, such as when a portable computer is used on an airplane. |
| Hardware device | The hardware device trigger enables a service to be started when a specific hardware device is connected to the system. Services that must function only when the system has either a specific hardware device that is connected or a hardware device that is connected that is part of a specific hardware device class should use this trigger.  For example, a Bluetooth management service might be trigger started only when at least one Bluetooth device is connected to the system.  The hardware device trigger does not support a device removal trigger event because the hardware class interface of the device remains present on the system after the device is removed. |
| Domain | The domain trigger enables a service to be started if a system is joined to a domain and stopped if the system is not joined to a domain. A service that is not required to be active unless the system is joined to a domain environment should use this trigger. This results in the service not being started on systems that are in nondomain environments, which includes most home environments.  For example, in Windows 7 the time service is enabled only on domain-joined systems. The time service synchronizes the local system's time with the domain controller’s time. |
| Group Policy | The Group Policy trigger enables a service to be started if a Group Policy change is detected. Services that are not required to be started unless there is a Group Policy change in the system can subscribe to this trigger and be started on notification of policy change. On startup, such a service must validate if the specific policy change relates to the functionality that the service provides and determine whether to continue or shut down based on the type of policy change. After the service performs the required action, it can automatically shut down until the next Group Policy change event occurs.  A service can subscribe to machine policy change or user policy change triggers.  A machine policy change trigger is generated:   * Whenever the system is started if at least one machine policy setting exists. * On background policy refresh (approximately every 90 minutes) if changes to the machine policy settings result in at least one machine policy setting.   A user policy change trigger is generated:   * On every user logon if at least one user policy setting for the user exists. * On background policy refresh (approximately every 90 minutes) if changes to the user policy settings result in at least one user policy setting. |
| Custom | Service developers can design their own custom trigger events that can be raised from either user mode or kernel mode. A service can then be started or stopped based on the custom trigger events. |

### SCM Trigger Actions

The Windows 7 SCM supports the operations of service start and service stop on trigger events. Other operations are not currently supported by service triggers .

### SCM Client API Support for Service Triggers

In Windows 7, the SCM’s **ChangeServiceConfig2** and **QueryServiceConfig2** client API functions have been updated to let service installers add, update, and query service trigger information.

Service trigger registration is performed by calling **ChangeServiceConfig2**, passing SERVICE\_CONFIG\_TRIGGER\_INFO for the *dwInfoLevel* parameter, and providing the trigger registration information in a SERVICE\_TRIGGER\_INFO structure through the *lpInfo* parameter. In addition, one or more trigger-specific data items can be specified. When the trigger provider generates a trigger for which the service has requested a start action, the SCM starts the service only if at least oneofthe specified trigger data items exists in the generated trigger.

The trigger registration information for a service can be obtained from the SCM by calling **QueryServiceConfig2**, passing SERVICE\_CONFIG\_TRIGGER\_INFO for the *dwInfoLevel* parameter. The trigger registration information is returned in a SERVICE\_TRIGGER\_INFO structure through the *lpInfo* parameter.

The following is an example of a service installer function that creates a hardware device trigger for a service that is named *MyService*:

#define SERVICE\_NAME L"MyService"

//set the device guid

DEFINE\_GUID (

            MY\_DEVICE\_GUID,

            0x54820a4a,

            0x8452,

            0x7554,

            0x65, 0x30, 0x23, 0x09, 0xa3, 0xb5, 0xee, 0xa4

            );

BOOL \_SetServiceToStartOnDeviceTrigger()

{

    BOOL fResult = FALSE;

    SC\_HANDLE hScm = OpenSCManager(

        NULL, //local machine

        NULL, //active database

        SC\_MANAGER\_CONNECT);

    if(hScm != NULL)

    {

        SC\_HANDLE hService = OpenService(

            hScm,

            SERVICE\_NAME,

            SERVICE\_CHANGE\_CONFIG);

        if(hService != NULL)

        {

            //set service start type to SERVICE\_DEMAND\_START

            fResult = ChangeServiceConfig(

                hService,

                SERVICE\_NO\_CHANGE,

                SERVICE\_DEMAND\_START,

                SERVICE\_NO\_CHANGE,

                NULL,

                NULL,

                NULL,

                NULL,

                NULL,

                NULL,

                NULL);

            // NULL means no change for all the above

            if(fResult)

            {

                static WCHAR\* szDeviceString[] = {

                    L"HID\_DEVICE\_UP:000D\_U:0001",

                    L"HID\_DEVICE\_UP:000D\_U:0002"

                };

                const unsigned int cDeviceStrings = sizeof(szDeviceString)/sizeof(szDeviceString[0]);

                SERVICE\_TRIGGER\_SPECIFIC\_DATA\_ITEM stdi[cDeviceStrings];

                for(unsigned int i = 0; i < cDeviceStrings; ++i)

                {

                    stdi[i].dwDataType = SERVICE\_TRIGGER\_DATA\_TYPE\_STRING;

                    stdi[i].cbData = (wcslen(szDeviceString[i]) + 1) \* sizeof(TCHAR);

                    stdi[i].pData = (PBYTE)szDeviceString[i];

                }

                SERVICE\_TRIGGER st;

                st.dwTriggerType = SERVICE\_TRIGGER\_TYPE\_DEVICE\_INTERFACE\_ARRIVAL;

                st.dwAction = SERVICE\_TRIGGER\_ACTION\_SERVICE\_START;

                st.pTriggerSubtype = (GUID \*) &MY\_DEVICE\_GUID;

                st.cDataItems = cDeviceStrings;

                st.pDataItems = stdi;

                SERVICE\_TRIGGER\_INFO sti;

                sti.cTriggers = 1;

                sti.pTriggers = &st;

                sti.pReserved = 0;

                fResult = ChangeServiceConfig2(

                    hService,

                    SERVICE\_CONFIG\_TRIGGER\_INFO,

                    &sti);

            }

            CloseServiceHandle (hService);

        }

        CloseServiceHandle (hScm);

    }

    if(!fResult)

    {

        printf("Service trigger registration failed (%d)\n", GetLastError());

    }

    return fResult;

}

### Sc.exe Windows Command-Line Tool Support for Service Triggers

The sc.exe command-line tool was updated for Windows 7 to support configuring and querying a service for supported triggers. The following shows the help information for the sc.exe triggerinfo command:

C:\>sc triggerinfo

DESCRIPTION:

Changes the trigger parameters of a service.

USAGE:

sc <server> triggerinfo [service name] <option1> <option2>...

OPTIONS:

start/device/UUID/HwId1/... <Start the service on arrival of the

specified device interface class UUID

string with one or more hardware ID

strings and/or compatible ID strings>

start/custom/UUID/data0/.. <Start the service on arrival of an

event from the specified custom ETW

provider UUID string with one or more

binary data items in hexadecimal

string format such as ABCDABCD to

set 4 byte data>

stop/custom/UUID/data0/... <Stop the service on arrival of an

event from the specified custom ETW

provider UUID string with one or more

binary data items in hexadecimal

string format such as ABCDABCD to

set 4 byte data>

start/strcustom/UUID/data0/.. <Start the service on arrival of an

event from the specified custom ETW

provider UUID string with one or more

optional string data items>

stop/strcustom/UUID/data0/.. <Stop the service on arrival of an

event from the specified custom ETW

provider UUID string with one or more

optional string data items>

start/networkon <Start the service on first IP address>

stop/networkoff <Stop the service on zero IP addresses>

start/domainjoin <Start the service on domain join>

stop/domainleave <Stop the service on domain leave>

delete <Delete the existing trigger parameters>

For example, the following shows how to configure the *bthserv* service to *Trigger Start* on a hardware device connect trigger event:

C:\>sc triggerinfo bthserv start/device/0850302a-b344-4fda-9be9-90576b8d46f0

[SC] ChangeServiceConfig2 SUCCESS

And the following shows how to query the *bthserv* service’s trigger information:

C:\>sc qtriggerinfo bthserv

[SC] QueryServiceConfig2 SUCCESS

SERVICE\_NAME: bthserv

START SERVICE

DEVICE INTERFACE ARRIVAL : 0850302a-b344-4fda-9be9-90576b8d46f0 [I

NTERFACE CLASS GUID]

**Note:** The trigger information for a service can be managed by using the sc.exe command-line tool and the SCM client API. The trigger configuration information cannot be managed by using the services.msc MMC snap-in.

## Best Practices for Services Development

The following is a checklist to help developers make appropriate choices for service attributes when they consider the development of a service.

* **Determine if a service is appropriate.**

Consider if the scenarios are better supported by a service or by a task. See “Determining the Appropriate Windows Background Process Model” later in this paper.

* **Consider the startup requirements**.

Unless the service must be started during system startup, do not use Automatic Start. Instead, use Demand Start or Trigger Start*.*

* **Shut down on idle.**

If the service is not required for long durations of time after it has completed activity, then stop the service when it becomes idle.

* **Use the security context of least privilege.**

Select the most appropriate user account in which the service must run. The account choice should be based on the level of access that the service requires. **LocalService** is the least-privileged account, and **LocalSystem** is the most-privileged account. For more information, see “Service User Accounts.”

* **Remove unnecessary access rights.**

Tighten the access rights to the service so that only selective users or groups have the required access to service configuration data and service management functions. Specifically, low-privileged users should not have SERVICE\_CHANGE\_CONFIG access because that access can be used to change the service account, service image path, and so on, thereby elevating the privilege of the service. For more information, see “Service Security and Access Rights.”

The Windows ecosystem supports many application and service types and designs. The best practices for developing services that were mentioned earlier do not address all application and service scenarios. Windows service developers should use this paper and other development resources that are listed in “Resources” at the end of this paper to ensure that their services correctly handle service management scenarios.

## Diagnosing Service Issues

Services may encounter service startup errors, service hangs, and various other failures. The following are some common service error codes:

* ERROR\_SERVICE\_LOGON\_FAILED
* ERROR\_SERVICE\_DEPENDENCY\_FAILED
* ERROR\_SERVICE\_START\_HANG

This section provides information about tools and methods to help you troubleshoot and resolve service issues. For more information about service-related error messages, see “Windows Server 2003 Managing System Services.”

### Diagnosing with Sc.exe

Sc.exe is a very powerful command-line tool that is included with Windows and lets you query the configuration and status of a service.

#### Querying Service Configuration

Developers can use the “sc.exe qc [ServiceShortName]” command to query and validate that the service configuration is correct.

The following example shows how to query the configuration information for the BITS service:

C:\>sc qc BITS

[SC] QueryServiceConfig SUCCESS

SERVICE\_NAME: BITS

TYPE : 20 WIN32\_SHARE\_PROCESS

START\_TYPE : 2 AUTO\_START (DELAYED)

ERROR\_CONTROL : 1 NORMAL

BINARY\_PATH\_NAME : C:\Windows\System32\svchost.exe -k netsvcs

LOAD\_ORDER\_GROUP :

TAG : 0

DISPLAY\_NAME : Background Intelligent Transfer Service

DEPENDENCIES : RpcSs

: EventSystem

SERVICE\_START\_NAME : LocalSystem

#### Querying Service Status

Developers can use the “sc.exe query [ServiceShortName]” command to query and validate that the status of the service is correct.

The following example shows status information for the BITS service:

C:\>sc query BITS

SERVICE\_NAME: BITS

TYPE : 20 WIN32\_SHARE\_PROCESS

STATE : 4 RUNNING

(STOPPABLE, NOT\_PAUSABLE, ACCEPTS\_SHUTDOWN)

WIN32\_EXIT\_CODE : 0 (0x0)

SERVICE\_EXIT\_CODE : 0 (0x0)

CHECKPOINT : 0x0

WAIT\_HINT : 0x0

Developers can also use the “sc.exe queryex [ServiceShortName]” command that additionally displays the process ID (PID) and FLAGS values.

### Identifying a Service in a Shared Service Host

Identifying a service within a shared service host process is typically the preliminary step in identifying a particular service that might be suspected of high CPU usage, a nonpaged pool memory leak, or excessive thread consumption. Developers should use the tasklist.exe tool to identify the PID of the process that hosts the service.

The following example shows the output from running the tasklist.exe tool:

C:\>tasklist /svc

Image Name PID Services

========================= ======== ============================================

System Idle Process 0 N/A

System 4 N/A

smss.exe 504 N/A

csrss.exe 576 N/A

wininit.exe 648 N/A

csrss.exe 660 N/A

services.exe 692 N/A

lsass.exe 704 KeyIso, Netlogon, ProtectedStorage, SamSs

lsm.exe 712 N/A

winlogon.exe 820 N/A

svchost.exe 892 DcomLaunch, PlugPlay

svchost.exe 968 RpcSs

svchost.exe 1012 WinDefend

svchost.exe 1116 Audiosrv, Dhcp, Eventlog, lmhosts, wscsvc

svchost.exe 1180 AudioEndpointBuilder, EMDMgmt, hidserv,

Netman, PcaSvc, SysMain, TrkWks,

UmRdpService, UxSms, WdiSystemHost,

Wlansvc, WPDBusEnum, wudfsvc

LVPrcSrv.exe 1228 LVPrcSrv

...

In this example, the svchost.exe process with PID 892 is hosting the DcomLaunch and the PlugPlay services.

### Diagnosing with EventVwr.MSC

The SCM logs events in the Windows System Event Log for service operations, such as starting and stopping a service, and for service issues. The event viewer MMC snap‑in—eventvwr.msc—allows you to view the events in the System Event Log that were logged by the SCM. The SCM event IDs range from 7000 to 7038. The service event ID dialog box provides a hyperlink that you can use to locate more details about the event and information that might help resolve the issue. For more information about common service events, see “Service Events Logging.”

Figure 3 shows an example of a service error event.

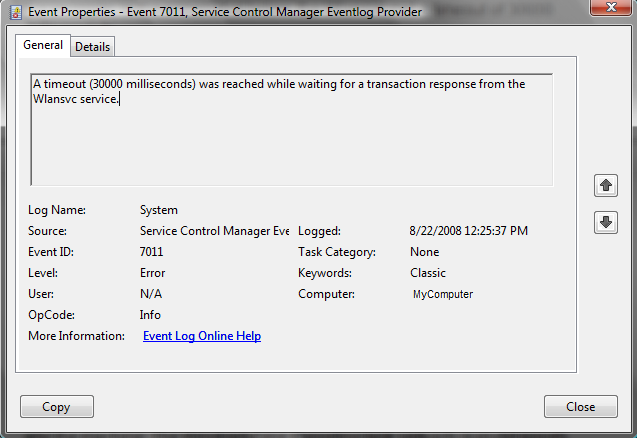


Figure 3: Service error event

# Tasks and the Task Scheduler

Like services, Windows Tasks (tasks) are designed to run as noninteractive system or user background processes. The Task Scheduler infrastructure in Windows operating systems provides for running a command or an application when an event of interest—a trigger—occurs. For example, the Windows ScheduledDefrag task is run on a weekly schedule trigger on every Wednesday at 1:00 a.m., but only if the system is idle and is running on AC power.

The Task Scheduler supports running tasks that are based on many triggers, including Event, Time, Daily, Weekly, Monthly, and other triggers. For more information about all the supported types of triggers, see “Trigger Types.”

Windows operating systems contain many native tasks that provide various functionalities. For example, the Windows Certificate Services Client automatically manages digital identities such as Certificates, Keys, and Credentials for the users and the machine, and the Windows Error Reporting task uploads queued error reports to Microsoft.

The primary difference between tasks and ordinary applications is that tasks are managed by the Task Scheduler. Tasks can be implemented by using the tasks API as a COM handler or can be Windows executables that are hosted by the Task Scheduler framework. The Task Scheduler maintains a database of installed tasks and provides a unified way to control them. The control operations for tasks include:

* Starting and stopping tasks.
* Managing running tasks.
* Maintaining task-related state information.

## Task States

Tasks exist in one of four states—*Ready*, *Queued*, *Running*, or *Disabled*—as described in Table 4.

Table 4: Task States

|  |  |
| --- | --- |
| Task state | Description |
| Ready | The task is ready to be run, but no instances of the task are queued or running. |
| Queued | One or more instances of the task are queued to be run. |
| Running | One or more instances of the task are running. |
| Disabled | The task is registered with the operating system but is disabled. No instances of the task are queued or running. The task cannot be run until it is enabled. |

After a task starts to run, the Task Scheduler manages the task’s state. For example, the Task Scheduler tries to stop the task if it exceeds the task execution time limit value. The Task Scheduler’s database contains the task settings for each task. These settings control the access that a task has to system resources and enables system administrators to control the access to each task. For more information about access control for tasks, see “Security Contexts for Tasks.”

## Task Scheduler Management Utilities

### SchTasks.exe

SchTasks.exe is the Task Scheduler command-line tool in Windows that lets you create, delete, start, or stop a task, query or modify the configuration of a task, and perform various other operations.

The following is an example of using SchTasks.exe to query the configuration of the ScheduledDefrag task:

C:\>schtasks /query /tn Microsoft\Windows\Defrag\ScheduledDefrag

Folder: \

TaskName Next Run Time Status

======================================== ====================== ===============

ScheduledDefrag 8/6/2008 1:00:00 AM Ready

### TaskSchd.MSC

The Task Scheduler MMC snap-in—taskschd.msc—provides a graphical user interface (GUI) that lets users or administrators perform basic task control operations on tasks. These operations include displaying the name and description of a task, starting and stopping a task, and specifying the account on the system in which a task runs.

An example of the Task Scheduler MMC snap-in is shown in Figure 4.

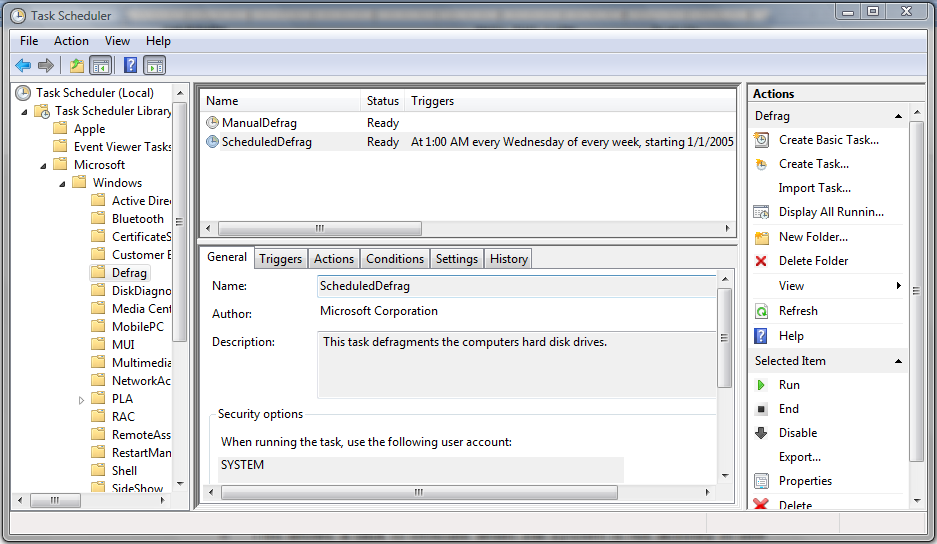


Figure 4: Task Scheduler MMC snap-in

## Optimizing Tasks

Tasks are run based on triggers that relate to the functionality of each task. The Task Scheduler includes settings that can help optimize the running of each task.

Developers should be aware of the following settings that can help improve system efficiency, power consumption, and user experience.

* **Idle condition**
* The idle condition setting specifies that the task should be run only when the system is idle. A system is considered to be idle if no keyboard or mouse activity has occurred during the previous 15 minutes and if there has been 0‑percent CPU usage and 0‑percent disk input or output for 90‑percent of that same period of time. The idle condition setting also allows stopping a running task whenever any user activity is detected.
* The idle condition setting enables a task to run whenever the system is not actively in use by a user. This helps reduce the potential of degraded performance while the user is actively interacting with the system. Additionally, the *End Task on Idle End* condition setting allows restoring the system to a more efficient user experience when the user returns to interacting with the system.
* **Power condition**
* The power condition setting specifies that the task should be run only when the system is running on AC power. The power setting also allows stopping a running task whenever the system changes from AC power to battery (DC) power.
* The power setting enables the system to conserve battery life by running the task only when the system is running on AC power.
* **Network condition**
* The network condition setting specifies that the task should be run only when the system is connected to a specific network that the system was connected to before, or more generally, when the system is connected to any network.
* The network condition setting enables the system not to run the task if the system is not connected to a network, such as a when a portable computer is used on an airplane.

## Best Practices for Task Development

The following is a checklist to help developers make appropriate choices for task attributes when they consider the development of a task:

* **Determine if a task is appropriate.**

Consider if the scenarios are better supported by a service or by a task. For more information, see “Determining the Appropriate Windows Background Process Model” later in this paper.

* **Help improve boot performance.**

Unless the task must run during system startup or when a user logs on, do not configure the task to use the boot or user logon triggers.

* **Select the appropriate constraints.**

Select the idle, power, and network condition settings to run the task only at the appropriate times. For more information, see “Optimizing Tasks” earlier in this paper.

* **Use the account with least privilege.**

Select the most appropriate user account in which the task must run. The account choice should be based on the level of access that the task requires. **LocalService** is the least-privileged account, and **LocalSystem** is the most-privileged account. For more information, see “**Principal.LogonType** **Property**.”

* **Remove unnecessary access rights.**

Tighten the access rights to the task so that only selective users or groups have the required access to the task configuration data and task management functions. This helps ensure that only authorized users can change the task configuration or delete the task. For more information, see “**IRegistrationInfo::SecurityDescriptor** Property.”

## Diagnosing Task Issues

This section provides information about tools and methods to help you troubleshoot and resolve task issues.

### Diagnosing with TaskSchd.msc

The Task Scheduler MMC snap-in—taskschd.msc—can be used to diagnose task issues. The **History** tab in the **Task Details** view lists all the events that are related to the currently selected task.

Figure 5 shows an example of the history of events for a task.

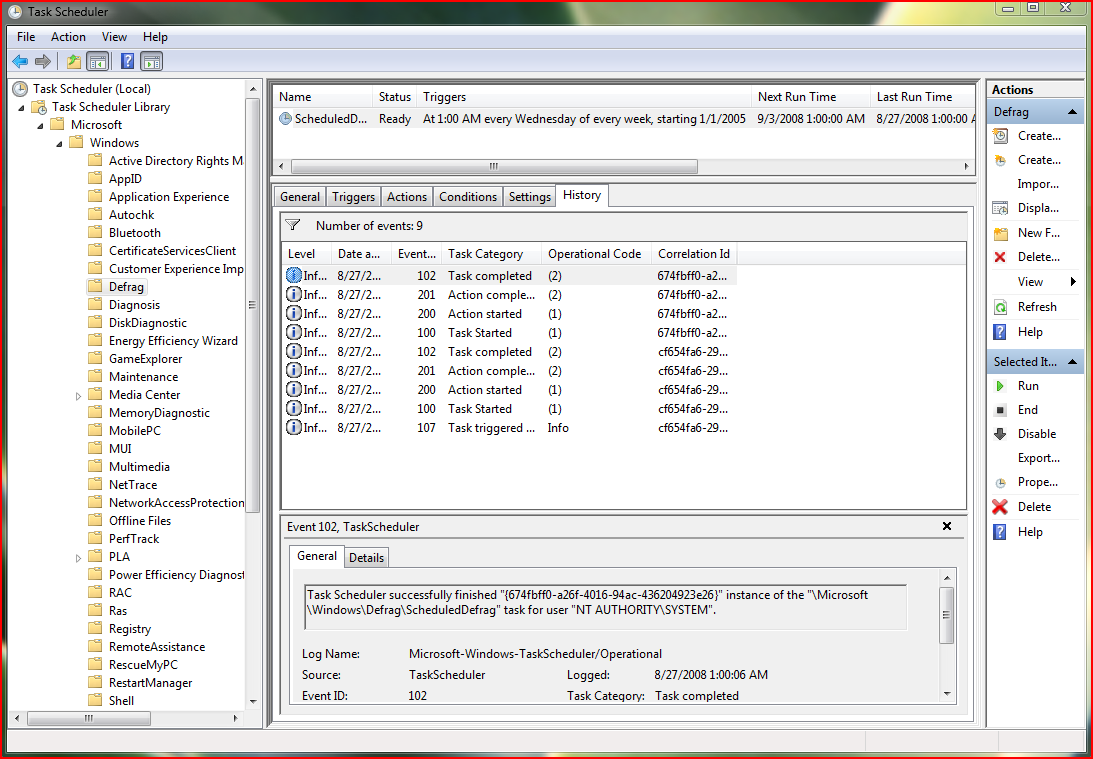


Figure 5: History of events for a task

The history of events for a task is very useful for identifying “missed run” issues, such as when a user expects a task to run but a task setting such as “do not run while on batteries” prevents the task from running.

### Diagnosing with EventVwr.msc

A more advanced method to investigate task issues is to use the Event Viewer MMC snap-in—eventvwr.msc—to examine the events that are generated by the Task Scheduler service that were logged in the \Applications and Service Logs\Microsoft \Windows\TaskScheduler\Operational channel. This method is effective in detecting issues that affect the correct operation of the Task Scheduler service and related issues such as running out of resources such as internal buffers, queues, or memory.

An even more advanced method to investigate task issues is to use the Event Viewer MMC snap-in to examine the events that were generated by the Task Scheduler service that were logged in the \Applications and Service Logs\Microsoft\Windows  
\TaskScheduler\Diagnostic channel. Note that this channel is not enabled by default. To enable this channel, an administrator must start Event Viewer and on the **View** menu enable **Show Analytic and Debug Logs**, right-click **Diagnostic channel**, select **Properties**, and select the **Enable logging** check box in the **Properties** dialog box. As with any logging facility, only the events that are generated after the logging is enabled are available to be examined. Therefore, after the diagnostic channel is enabled, you might be required to reproduce the task issue to be able to collect and examine the corresponding events.

# Selecting the Appropriate Windows Background Process Model

Developers must select the appropriate model for their background process. Selecting the appropriate model involves evaluating the functionality, performance, and security requirements for the background process.

Developers should use the following information to help them select between the service and task models.

Services are generally appropriate for background processes that have one or more of the following characteristics:

* Runs for a long period of time or must stay running as long as the system is running.
* Requires no user interaction.
* Must be aware of system-wide events such as power events or session change events.
* Provides functionality such as device management, database, or Web server services.

For example, a network firewall process that runs continuously in the background is best designed as a service.

Tasks are generally appropriate for background processes that are intermittent, such as running based on a schedule or running in the context of each user on the system. For example, a weekly disk defragmentation activity is best designed as a task.

Table 5 contains a side-by-side comparison of background process features that are supported by services and tasks.

Table 5: Background Process Features

|  |  |  |
| --- | --- | --- |
| Background process feature | Service | Task |
| Account | Any account that has the logon as service right for service logons privilege, such as **LocalService**, **NetworkService**, or **LocalSystem**. | **LocalService**, **NetworkService**, or **LocalSystem** for service logons.  Interactive user for interactive logons (a task can run as a user or as a member of a group).  Any account that has logon as batch right for batch logons. |
| Hosting | Shared process or own process. Can select a specific shared group based on security characteristics. | Own process. |
| Programming model | Implement **ServiceMain** function, Service Control Handler function, and report service status to the SCM. For more information, see “Service Programs.” | Uses COM interfaces. Implement Start and Stop procedures and report task completed to the Task Scheduler. For more information, see “ITaskHandler Interface.”  A task can also be a stand-alone application that does not use a specific programming model. |
| Start models | Automatic, Demand Start, and Automatic (Delayed Start).  Can be started and stopped on triggers, such as device arrival, IP address availability, and custom Event Tracing for Windows (ETW) events (available in kernel mode). | Can be started on triggers or Demand Start.  Supported triggers: Session change (such as user logon), system boot, schedule (hourly, daily, weekly, and so on), system idle, task registration/ update, Windows Event Log service event (available in kernel mode).  Can be stopped if the system is no longer idle or if the system is switching to battery power. |
| Configurable settings for checks before starting | None. | Multiple instances policy, machine has been idle for the specified duration, Demand Start allowed, start on battery allowed, and network available. |
| Session | Session 0. | Session 0 for service and batch logons.  Interactive session (session 1, 2, and so on) for interactive logons only. |
| Least-privileged user account (LUA) | Built-in accounts (**LocalService**, **NetworkService**, **LocalSystem**) run at system integrity level.  Accounts that are not built in run at high-integrity level. | Built-in service logon accounts (**LocalService**, **NetworkService**, **LocalSystem**) run at system-integrity level.  Service logon accounts that are not built in run at high-integrity level.  Administrator’s group batch logons run at high-integrity level.  Nonadministrator’s group batch logons run at medium-integrity level.  Administrator’s group interactive logons can run at medium- or high-integrity level, depending on the task settings.  Nonadministrator’s group interactive logons run at medium-integrity level. |
| Notifications or commands | The SCM gives the following notifications or commands in the control handler:  device event, power event, session event, machine preshutdown, stop, machine shutdown. | For own process (Windows executable files only), WM\_CLOSE on stop. For shared COM tasks, the Stop procedure is called on stop. |
| Dependency model | One service can depend on other services to ensure startup and shutdown order. | None. |
| Instances | One per system. | Zero or more instances, as indicated by the Multiple Instances Policy that is specified in the task definition. Available options are:   * Ignore new instances * Parallel * Queued * Stop existing |
| Recovery policies | Recovery policies on crash or on error are specified by calling **SetServiceStatus**. Possible actions are to restart the service, reboot the machine, or run a script. | Recovery policies on crash or on error are specified in the task completed callback. The only possible action is to restart the task. |
| State | Start pending, running, stop pending, stopped. Can lengthen the start or stop time dynamically by calling **SetServiceStatus**. | Running, queued, or stopped. |
| Process hardening | Can opt in to having a unique service security identifier (SID) in the process token groups list. Can run the service in a write-restricted token. Can control network traffic that is based on the service SID. | By default, tasks that run in **LocalService** and **NetworkService** accounts in Windows 7 are hardened. The process token groups list has the task SIDs that correspond to all hosted tasks in the process. Tasks can opt out of this behavior. By default, **SeImpersonatePrivilege** is removed from the process token. Tasks can opt out of the removal of the **SeImpersonatePrivilege** and can specify the required privileges, in which case all unnecessary privileges are permanently removed from the process token. The default token DACL of hardened task processes lets only the task SIDs in that process and **LocalSystem** have full access. |
| Remoting | Remotable API. | Remotable API. A user must have Administrator privileges to create a task on a remote system. |

# Next Steps

Developers should optimize their background processes to help improve system performance and responsiveness, to improve energy efficiency, to help lessen security threats, and to provide a good user experience. The background process infrastructure and optimization information in this paper should be reviewed before designing background processes for Windows.

In summary, the following are the specific actions that developers should take to make the best choices when they design a background process.

* Review the information in this paper to help optimize background processes for performance, energy efficiency, security, and user experience.
* Determine the appropriate Windows background process model for the required functionality. Generally, use the service programming model for long-running background processes that run activity across multiple user sessions and that manage system resources such as device, database, or network resources. Use the task programming model for intermittent or less frequently run background processes that run in user sessions.
* When developers design a service, consider the following:
* Design the service to start only when it is required to be running.
* Stop the service when it is idle.
* Specify the service to run in a secure account.
* Specify the service configuration to limit service management to specific accounts.
* When developers design a task, consider the following:
* Design the task not to use frequent triggers, such as user logon, unless it is absolutely required to provide the desired functionality.
* Specify task condition settings that let the task run only when the system is idle, the system is running on AC power, and the system is connected to network, as is appropriate for the task.
* Specify the task to run in a secure account.
* Specify the task configuration to limit task management to specific accounts.

For additional information about services and tasks, see “Resources” at the end of this paper.

# Glossary

The following terms are related to Windows background processes:

Automatic Start

The startup type for a service that is started automatically every time that the operating system is started.

Automatic (Delayed Start)

The startup type for a service that is started automatically every time that the operating system is started. However, it is not started until approximately 2 minutes after the operating system has completed system startup.

Demand Start/Manual Start

The startup type for a service that is started by a dependent service, by an explicit call to the SCM client API, or by COM service activation.

Least-Privileged User Account (LUA)

(Also known as Limited User Account, Least User Access, and other variations.)  
A computer user account that cannot make changes that affect other users of the system or the operating system itself. In Windows, these accounts are typically a member of the built-in Users group. Members of this group are explicitly not members of powerful groups (such as Administrators, Power Users, or Backup Operators), and they do not hold elevated privileges (such as Load and unload device drivers, Take ownership of files or other objects, or Act as part of the operating system).

service

A background process that is implemented by using the Windows Services programming model and API.

Service Control Manager (SCM)

A Windows background process infrastructure component that maintains a database of installed services and starts services based on each service's startup type. It maintains the status of running services and sends control requests to services.

task

A background process that is run whenever an event of interest occurs.

Task Scheduler

A Windows background process infrastructure component that schedules tasks to be run whenever the events of interest for each task occur.

trigger

An event that can be used to start or stop a service or to run a task.

Trigger Start

The startup type for a service where the service can be started or stopped based on trigger events.

# Resources

## Service Resources

Services

<http://msdn.microsoft.com/en-us/library/ms685141(VS.85).aspx>

Services in Windows Vista

<http://www.microsoft.com/whdc/system/vista/Vista_Services.mspx>

Service Programs

<http://msdn.microsoft.com/en-us/library/ms685967(VS.85).aspx>

Starting a Service

<http://msdn.microsoft.com/en-us/library/ms686315(VS.85).aspx>

Service Trigger Events

<http://msdn.microsoft.com/en-us/library/dd405513(VS.85).aspx>

Service User Accounts

<http://msdn.microsoft.com/en-us/library/ms686005(VS.85).aspx>

Service Security and Access Rights

<http://msdn.microsoft.com/en-us/library/ms685981(VS.85).aspx>

Impact of Session 0 Isolation on Services and Drivers in Windows Vista

<http://www.microsoft.com/whdc/system/vista/services.mspx>

Windows Server 2003 Managing System Services

<http://www.microsoft.com/downloads/details.aspx?FamilyID=a70b06cb-b0f2-4800-997b-2a27ce8fcdc2&displaylang=en>

Service Events Logging

<http://technet.microsoft.com/en-us/library/cc756348.aspx>

Service Control Manager API Reference

<http://msdn.microsoft.com/en-us/library/ms685974(VS.85).aspx>

## Task Resources

Task Scheduler

<http://msdn.microsoft.com/en-us/library/aa383614(VS.85).aspx>

Windows Vista Task Scheduler

<http://technet.microsoft.com/en-us/appcompat/aa906020.aspx>

Tasks

<http://msdn.microsoft.com/en-us/library/aa382589(VS.85).aspx>

Trigger Types

<http://msdn.microsoft.com/en-us/library/aa383987(VS.85).aspx>

Security Contexts for Tasks

<http://msdn.microsoft.com/en-us/library/aa382140(VS.85).aspx>

Principal.LogonType Property

<http://msdn.microsoft.com/en-us/library/aa382075(VS.85).aspx>

IRegistrationInfo::SecurityDescriptor Property

<http://msdn.microsoft.com/en-us/library/aa381077(VS.85).aspx>

ITaskHandler Interface

<http://msdn.microsoft.com/en-us/library/aa381370(VS.85).aspx>

Task Scheduler Reference

<http://msdn.microsoft.com/en-us/library/aa383608(VS.85).aspx>

## Other Resources

User Account Control

<http://msdn.microsoft.com/en-us/library/bb648649.aspx>

Windows Vista Integrity Mechanism Technical Reference

<http://msdn.microsoft.com/en-us/library/bb625964.aspx>