



Microsoft Dynamics™ GP
Workflow Integration Guide
Release 10.0

Copyright

Copyright © 2008 Microsoft Corporation. All rights reserved.

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. Notwithstanding the foregoing, the licensee of the software with which this document was provided may make a reasonable number of copies of this document solely for internal use.

Trademarks

Microsoft, Microsoft Dynamics, Dexterity, Outlook , InfoPath, SharePoint server, Visual Studio, SQL server, Windows, and Windows Server are either registered trademarks or trademarks of Microsoft Corporation or its affiliates in the United States and/or other countries.

The names of actual companies and products mentioned herein may be trademarks or registered marks - in the United States and/or other countries - of their respective owners.

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

Intellectual property

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.

Warranty disclaimer

Microsoft Corporation disclaims any warranty regarding the sample code contained in this documentation, including the warranties of merchantability and fitness for a particular purpose.

Limitation of liability

The content of this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Microsoft Corporation. Microsoft Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this manual. Neither Microsoft Corporation nor anyone else who has been involved in the creation, production or delivery of this documentation shall be liable for any indirect, incidental, special, exemplary or consequential damages, including but not limited to any loss of anticipated profit or benefits, resulting from the use of this documentation or sample code.

License agreement

Use of this product is covered by a license agreement provided with the software product. If you have any questions, please call the Great Plains Customer Assistance Department at 800-456-0025 (in the U.S. or Canada) or +1-701-281-6500.

Publication date

September 2008 -- Last updated September 4, 2008

Contents

Introduction	2
What's in this manual.....	2
Prerequisites.....	2
Symbols and conventions	3
Product support	3
What's new in Workflow for Microsoft Dynamics GP 10.0.....	4
What to do next.....	4
Part 1: Workflow Basics	6
Chapter 1: Architecture	7
Architecture diagram	7
Microsoft Office SharePoint Server	8
Workflow server.....	8
Workflow type assembly	10
Workflow clients	11
Chapter 2: Integration Types	15
Creating a new workflow	15
Extending an existing workflow.....	16
Chapter 3: Sample Workflow Type	19
Workflow type sample overview.....	19
Workflow type sample files.....	20
Installing the sample application.....	21
Installing the document type assembly.....	21
Installing eConnect Transaction Requester components	22
Installing the Sales Lead web service.....	22
Updating the Dynamics Security Service.....	23
Installing the server workflow assembly	25
Installing the document viewer	27
Installing the client workflow assembly.....	28
Viewing the workflow type application.....	29
Chapter 4: Sample Workflow Extension	31
Workflow extension sample overview.....	31
Workflow extension sample files.....	32
Installing the purchase order integration sample	32
Installing the web service extension sample.....	33
Installing the workflow extension sample	34
Viewing the workflow extension sample	36
Part 2: Creating a New Workflow	38
Chapter 5: Designing a New Workflow	39

Is a workflow appropriate?	39
Microsoft Dynamics GP client.....	39
Business logic.....	40
Data access	40
Workflow server.....	41
Chapter 6: Dictionary Changes.....	43
Tables	43
Table security	44
Command form.....	44
Window integration.....	46
Notifications	48
Home Page integration	51
List integration	53
Application assembly	67
Chapter 7: Client Workflow Assembly.....	69
Overview.....	69
Creating a Visual Studio project	69
Creating a BusinessObjectKey	70
Creating a form controller	72
Creating a list controller.....	84
Creating a form factory	91
Building the client workflow assembly	95
Chapter 8: Web Service.....	97
Creating a document type	97
Creating a web service	114
Securing the web service.....	125
Testing the web service	132
Chapter 9: Server Workflow Assembly.....	135
Creating a Visual Studio project	135
Creating a workflow type.....	137
Adding business logic	149
Signing your server workflow assembly.....	154
Creating a document viewer	155
Creating a workflow event subscription helper application.....	160
Building the assembly and application	165
Chapter 10: Deploying the New Workflow	167
Installing the application	167
Installing the workflow document type.....	167
Installing the web service	167
Installing the server workflow assembly	169
Installing the client workflow assembly.....	171
Viewing the workflow.....	173

Part 3: Extending an Existing Workflow	176
Chapter 11: Designing a Workflow Extension	177
Making data available to workflow	177
Workflow extension assembly	178
Data access	178
Chapter 12: Web Service Extension	179
Creating a Visual Studio project	179
Defining the extension data.....	180
Adding the web service event handler	181
Building the web service extension.....	182
Registering the web service extension	183
Testing the web service extension	184
Chapter 13: Workflow Extension Assembly	187
Creating a Visual Studio project	187
Adding a resources file	188
Adding the workflow extension code.....	189
Building the workflow extension	192
Chapter 14: Deploying the Workflow Extension	193
Web service extension	193
Adding assemblies to the global assembly cache	193
Installation helper	194
Registering the workflow extension	195
Part 4: Dynamics Workflow Web Service	198
Chapter 15: Connecting to the Web Service	199
Web service URL	199
Workflow web service proxy.....	200
Web service namespace.....	200
Creating a web service instance.....	200
Chapter 16: Using the Web Service	203
Dynamics Workflow Web Service Reference	203
Dynamics Workflow web service example	203
Business object keys.....	204
BusinessObjectKey reference.....	205
Appendix	208
Appendix A: Troubleshooting	209
Resetting the workflow server	209
Verifying authorization	209
Appendix B: Debugging	211
Debugging the client workflow assembly.....	211
Debugging the server workflow assembly	212

C O N T E N T S

Appendix C: Changing Passwords	215
Glossary	217
Index	219

Introduction

Welcome to Workflow for Microsoft Dynamics™ GP SDK. The workflow SDK provides files, tools, and services that allow applications to customize the Microsoft Dynamics GP workflow approval process. This documentation explains how to use workflow to complete the following tasks:

- Create a new workflow type.
- Extend an existing workflow
- Interact with the Dynamics Workflow web service.

Before you begin installing and using workflow, take a few moments to review the information presented here.

What's in this manual

The Microsoft Dynamics GP Workflow Integration Guide is designed to give you an in-depth understanding of how to create integrations for workflow. The manual guides you through the steps to add a new workflow or extend an existing workflow.

Information is divided into the following parts:

- [Part 1, Workflow Basics](#) describes the workflow architecture, the types of workflow integrations you can create, and the sample workflow integration.
- [Part 2, Creating a New Workflow](#) describes how to build the components you need to create a new approval workflow for a back-office document.
- [Part 3, Extending an Existing Workflow](#) describes how to build the components you need to add new data fields to an existing workflow.
- [Part 4, Dynamics Workflow Web Service](#) describes how to connect to and use the Dynamics Workflow web service from an external application.

To learn about installing Microsoft Dynamics GP Workflow, refer to the Workflow Installation Guide. For additional information about configuring and managing workflow for Microsoft Dynamics GP, see the **Microsoft Dynamics GP Workflow Administrator's Guide**.

Prerequisites

Before you begin developing with the Microsoft Dynamics GP Workflow SDK, you should be familiar with the workflow administration. It is also assumed that you have previous knowledge of the following:



- Knowledge about the Microsoft Dynamics GP approval workflow.
- Experience using Visual Studio.
- Experience using the C# programming language.
- Experience with Microsoft Dynamics GP Dexterity development.
- How to use Visual Studio Tools for Dynamics GP.

- How to create and use XSLT files.
- Experience using Web Services for Microsoft Dynamics GP and eConnect for Microsoft Dynamics GP.
- How to use the Dynamics Security Service to update web service security roles.
- Experience using the Microsoft SQL Server™ Management Studio.
- Experience with Microsoft Internet Information Services (IIS) administration.
- You must have Visual Studio installed on the workflow server where you plan to use the SDK.

If you need to improve your skills in any of these areas, consult any of the various reference materials that discuss them.

Symbols and conventions

To help you use this documentation more effectively, we've included the following symbols and conventions within the text to make specific types of information stand out.

Symbol	Description
	The light bulb symbol indicates helpful tips, shortcuts, and suggestions.
	Warnings indicate situations you should be aware of when completing tasks.
<i>Margin notes summarize important information.</i>	Margin notes call attention to critical information and direct you to other areas of the documentation where a topic is explained.
Convention	Description
Part 2, Creating a New Workflow	Bold type indicates a part name.
Chapter 1, "Architecture" <i>Overview</i>	Quotation marks indicate a chapter name. Italicized type indicates a section name.
<code>using System.Xml;</code>	This font is used to indicate script examples.
Microsoft Office SharePoint Server (MOSS)	Acronyms are spelled out the first time they're used.
TAB or ALT+M	Small capital letters indicate a key or a key sequence.

Product support

Microsoft Dynamics GP technical support can be accessed online or by telephone. Go to www.microsoft.com/Dynamics and click the CustomerSource or PartnerSource link, or call 888-477-7877 (in the US and Canada) or 701-281-0555.

What's new in Workflow for Microsoft Dynamics GP 10.0

Microsoft Dynamics GP 10.0 Feature Pack 1 changes how Microsoft Dynamics GP archives workflow history information. When a workflow completes, the workflow history and tracking information for that document are stored in the Microsoft Dynamics GP database. The components that the Feature Pack uses to store workflow history are now available for use with all workflow types.

A key component of the Microsoft Dynamics GP Workflow archive is the `IDynamicsWorkflowHistory` interface. The interface includes methods you use to define how your workflow saves and retrieves workflow history. Refer to [Chapter 9, "Server Workflow Assembly,"](#) for more information about implementing the `IDynamicsWorkflowHistory` interface.

What to do next

To begin learning about the Microsoft Dynamics GP Workflow, we recommend the following steps:

1. Read Part 1, Workflow Basics

This will provide the appropriate background for developing integrations with Microsoft Dynamics GP Workflow.

2. Install the sample files from the Microsoft Dynamics GP Workflow SDK.

If you have not already done so, install all the files from the Microsoft Dynamics GP Workflow SDK. The Microsoft Dynamics GP Workflow SDK contains references and sample files that will help you to understand the materials presented in this guide.

3. Install the sample application and sample workflow integration.

A sample application is included with the Microsoft Dynamics GP Workflow SDK. This sample application demonstrates the techniques you will use as you develop your own workflow integrations.

Refer to [Chapter 3, "Sample Workflow Type,"](#) for more information about installing and using the sample application.

4. Continue reading the Workflow Integration Guide.

The remaining chapters in the Workflow Integration Guide will provide you with information about creating integrations with Microsoft Dynamics GP Workflow.

Part 1: Workflow Basics

This portion of the documentation contains basic information you should know before adding or extending a workflow. The following information is discussed:

- [Chapter 1, “Architecture.”](#) provides an overview of workflow and what Workflow for Microsoft Dynamics GP provides.
- [Chapter 2, “Integration Types.”](#) describes the two types of workflow integrations you can create.
- [Chapter 3, “Sample Workflow Type.”](#) describes the workflow type sample included with the Microsoft Dynamics GP Workflow SDK. Details about installing the sample and the components used to create it are provided.
- [Chapter 4, “Sample Workflow Extension.”](#) describes the workflow extension sample included with the Microsoft Dynamics GP Workflow SDK. Details about installing the extension sample and the components used to create it are provided.

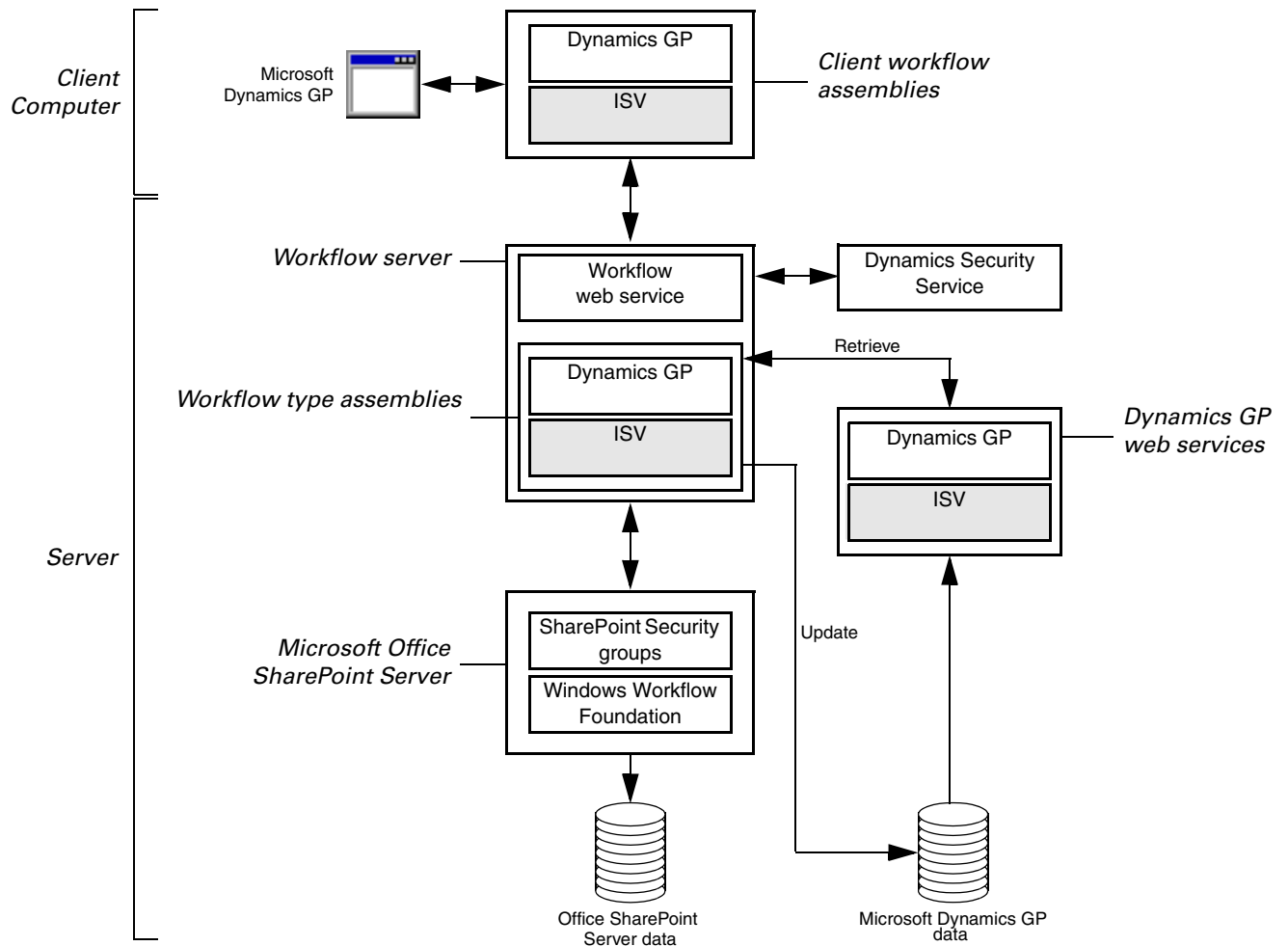
Chapter 1: Architecture

To create custom workflow solutions, you add and register components with the Microsoft Dynamics GP Workflow framework. To understand how your components combine with Microsoft Dynamics GP Workflow, it will be helpful to understand the architecture of the workflow framework. Information about workflow architecture is divided into the following sections:

- [Architecture diagram](#)
- [Microsoft Office SharePoint Server](#)
- [Workflow server](#)
- [Workflow type assembly](#)
- [Workflow clients](#)

Architecture diagram

Workflow is a process that automates routing a document to users responsible for performing actions on that document. Microsoft Dynamics GP implements Workflow using a framework that attaches workflow functionality to specific Microsoft Dynamics GP documents. The following diagram shows the main components of Microsoft Dynamics GP Workflow:



You create a new workflow or extend an existing workflow by supplying components that plug into the Microsoft Dynamics GP Workflow framework. The shaded elements represent the components you use to customize Workflow.

Microsoft Office SharePoint Server

The Microsoft Dynamics GP Workflow server builds upon the workflow capabilities provided by Microsoft Office SharePoint Server (MOSS). Currently, Microsoft Dynamics GP uses MOSS to define and deploy an “approval” workflow type. All the available workflows are based on this “approval” workflow type.

The Dynamics GP approval workflow is built from a generic Windows Workflow Foundation schedule. The schedule controls what types of tasks a workflow can perform and how those tasks are scheduled. Workflow uses MOSS to host the workflow schedule. As a host, MOSS provides the runtime infrastructure for each workflow instance and stores the data and history for each workflow instance.

You customize this generic Windows Workflow Foundation schedule by combining it with workflow metadata. You create this metadata when you use the Microsoft Dynamics Workflow web site to configure workflow preferences and approval hierarchies. MOSS stores your selections as workflow metadata.

When Workflow runs, your configuration metadata is combined with the generic workflow schedule to create the approval routing hierarchy for a specific Workflow instance. Separating the workflow schedule from the configuration metadata allows you to extend Workflow to other documents.

Workflow server

Workflow includes several components that you install on the same server as Microsoft Office SharePoint Server (MOSS). The Workflow server component allows client applications to initiate, monitor, and retrieve workflow information.

Workflow web service

The Workflow server component uses a web service to enable a client to initiate workflow actions and retrieve workflow information. The web service’s web methods allow you to interact with Workflow without having to learn Microsoft Office SharePoint Server or the Windows Workflow Foundation.

Workflow components

The Workflow components behind the web service implement workflow business logic, perform actions using MOSS, and retrieve workflow information from MOSS and Microsoft Dynamics GP. To support a client’s workflow request, the Workflow server components utilize the following resources:

- Workflow uses MOSS to store the workflow definitions. A workflow definition contains the activities, schedule, and configuration information that control the operation of workflow instances.
- Workflow uses MOSS to host the workflow process. For example, when you call the `SubmitForApproval` web method in the Dynamics Workflow web service, MOSS places the data for the workflow instance into a SharePoint list item and starts a workflow instance that performs the `SubmitForApproval` actions specified by the workflow metadata in the list item.

- Workflow uses MOSS to store information about each workflow instance.
- Workflow uses Microsoft .NET assemblies to define business objects. A business object represents a Microsoft Dynamics GP documents. The business object must be able to retrieve and update Microsoft Dynamics GP data.
- Workflow uses events to trigger actions by the business objects. When you register your assembly with Workflow, you specify the events that your business object handles.

Security

The Workflow server components provide a security framework for all workflows. Workflow security uses the following components:

- Workflow secures access to the Workflow web service via membership in an Office SharePoint Server group. You create and maintain these groups through the Workflow SharePoint site. For additional information on creating and maintaining Office SharePoint Server groups, refer to the **Microsoft Dynamics GP Workflow Administrator's Guide**.
- During installation, Microsoft Dynamics GP creates a SQL role named DYNWORKFLOWGRP. This role provides access to the objects and tables on the Microsoft Dynamics GP SQL Server that workflow uses to perform tasks.



To allow a new workflow to access the Microsoft Dynamics GP SQL database, you grant DYNWORKFLOWGRP the appropriate permissions to access the SQL Server objects or tables used by that workflow.

- When Workflow accesses resources, it uses the identity of the SharePoint web site application pool or the Windows SharePoint Services Timer service. To allow Workflow access to Microsoft Dynamics GP data and Microsoft Dynamics GP Web Services, the account assigned to the application pool and the Windows SharePoint Services Timer service must be a member of the DYNWORKFLOWGRP role on the Microsoft Dynamics GP SQL Server.
- To access Microsoft Dynamics GP web services, Workflow uses the Dynamics Security Service (DSS) to ensure the user is authorized to perform the web service task. During installation, the login account used by the SharePoint application pool and Windows SharePoint Services Timer service is assigned the Workflow Administrator role. This role is assigned the web service tasks required by the existing Dynamics GP workflows that are available after installation.



To use additional Dynamics GP web services, you may need to add tasks to the Workflow Administrator role using the Dynamics Security Console. To learn how to configure the Dynamics Security Service, review Security in the Web Services for Microsoft Dynamics GP Installation and Administration Guide.

Workflow type assembly

To use Workflow, the generic workflow schedule defined in MOSS must be associated with specific Microsoft Dynamics GP business objects. For example, the Purchase Order Approval workflow applies the generic approval workflow schedule to Microsoft Dynamics GP Purchase Order documents. The association between Workflow and a business object is called a Workflow type.

Microsoft Dynamics GP uses Microsoft .NET assemblies to define each Workflow type. A Workflow type assembly provides the following capabilities:

- The Workflow type assembly provides metadata that uniquely identifies each Workflow and its associated business object.
- The Workflow type assembly implements the IDynamicsWorkflow interface. The interface ensures all Workflow types provide the required set of methods.
- The Workflow type assembly retrieves and updates Microsoft Dynamics GP data for the business object associated with the workflow type.
- The Workflow type assembly implements the event handlers that cause a workflow to react to actions performed by the user.

The ability to raise and handle events is a key Workflow capability. The Workflow server uses events to notify subscribers when specific workflow events occur. The subscriber implements an event handler for that event. The event handler retrieves the Microsoft Dynamics GP document, executes business logic related to the workflow event, and saves the Microsoft Dynamics GP document to the database.

To subscribe to an event, you must register your workflow type with the Workflow server. New workflow types use the same registration process as the workflows included with Microsoft Dynamics GP. You will learn more about this in [Chapter 10, "Deploying the New Workflow."](#)

Data management

When the workflow type assembly responds to an event, it must retrieve the Microsoft Dynamics GP document for which a workflow action is being performed. The workflow type assembly must also save any document changes to the Microsoft Dynamics GP database.

Workflow does not mandate how to retrieve or update Microsoft Dynamics GP data that is affected by workflow operations. The existing Dynamics GP workflows use both Dynamics GP web services and direct database commands to perform data access tasks.

When a workflow type assembly accesses web services or the Microsoft Dynamics GP database, it uses the identity of the SharePoint web site application pool.

Document viewer

Each workflow type must supply a web-based document viewer. The document viewer displays key information about a specified document. The architecture diagram does not include the document viewer.

Displays information from the sales quote document.

Sales Quote Number:	QTEST1024
Date:	4/12/2017
Quote expiration date:	
Workflow priority:	Normal
Workflow status:	Approved
Customer:	AARONFIT0001
Ship to:	Aaron Fitz Electrical 11403 45 St. South Chicago IL 60603-0776 (312) 555-0102 X0000
Customer PO number:	
Salesperson Id:	PAUL W.
Shipping method:	LOCAL DELIVERY
Payment terms:	Net 30
Requested ship date:	

Item Number	Description	Quantity	U of M	Requested ship date:	Unit Price	Extended Price	Shipping method:	Discount
24X IDE	24x CD-ROM	1	Each		\$40.00	\$40.00	LOCAL DELIVERY	0.00 %
Subtotal:					\$40.00			
Misc:					\$0.00			
Tax:					\$0.00			
Freight:					\$0.00			
Trade Discount:					\$0.00			
Total:					\$40.00			

Comment:

You access the document viewer when you click the Record ID link in the list of documents found in the Workflow Documents tab of the Microsoft Dynamics Workflow web site. You also use the document viewer when you click the “View Document link in the email messages sent with some workflow notifications.

View Document: [PO000000000000038](#)
 Workflow Step Name: Step 1
 View Workflow History and Status: [PO000000000000038](#) *The link opens the specified record in the document viewer.*

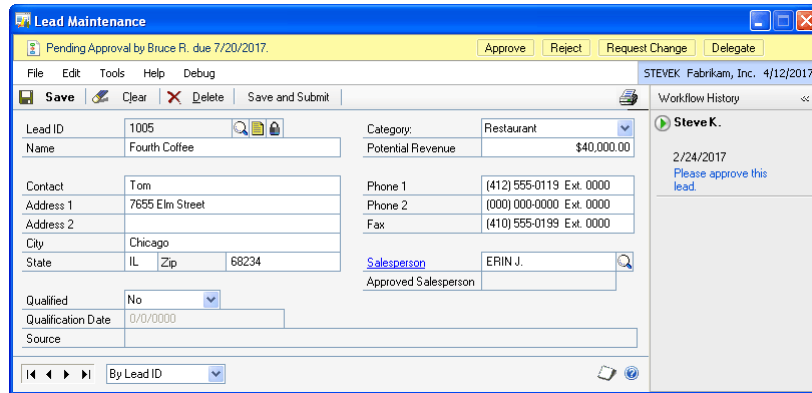
Workflow clients

The [Architecture diagram](#) shows a Microsoft Dynamics GP client being used to access workflow. Workflow also supports a web client and a Microsoft Office Outlook 2007 email client. These clients use a Microsoft Office InfoPath form that allows a workflow approver to approve or reject a document.

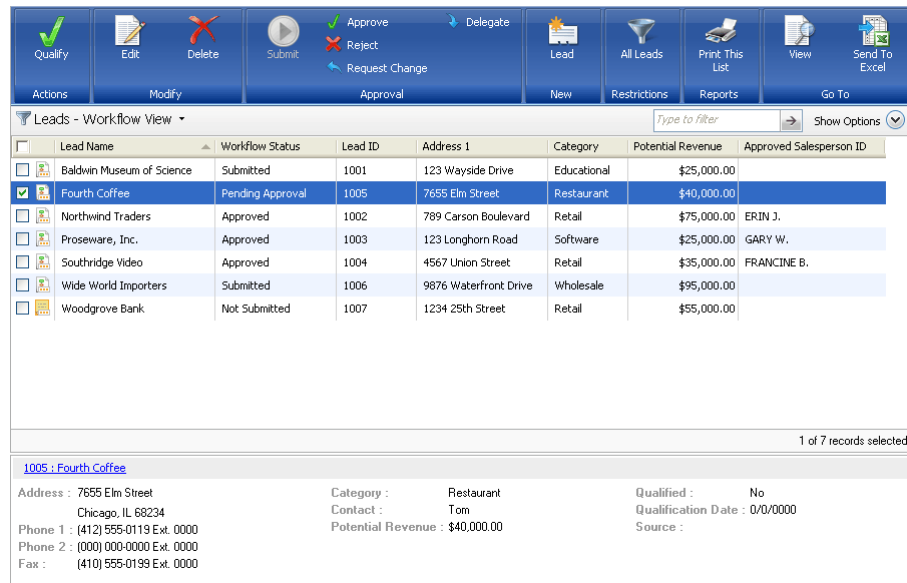
Microsoft Dynamics GP client

When creating a new workflow for a Microsoft Dynamics GP document, you must add workflow support to the Microsoft Dynamics GP client. To support your new workflow, several changes and additions are required in the Dexterity-based application dictionary. Some of these changes implement core portions of the new workflow. Other changes allow forms or lists you add to the Microsoft Dynamics GP client to use the workflow client interface. For additional information about adding to the application dictionary, see [Chapter 6, “Dictionary Changes.”](#)

To allow a document to use a new workflow, you create a client workflow assembly that implements the workflow client interface. The dictionary changes and client workflow assembly allow a Microsoft Dynamics GP client form to display workflow information and to initiate workflow actions.



The client workflow assembly also allow a Microsoft Dynamics GP list to display workflow information and initiate workflow actions.

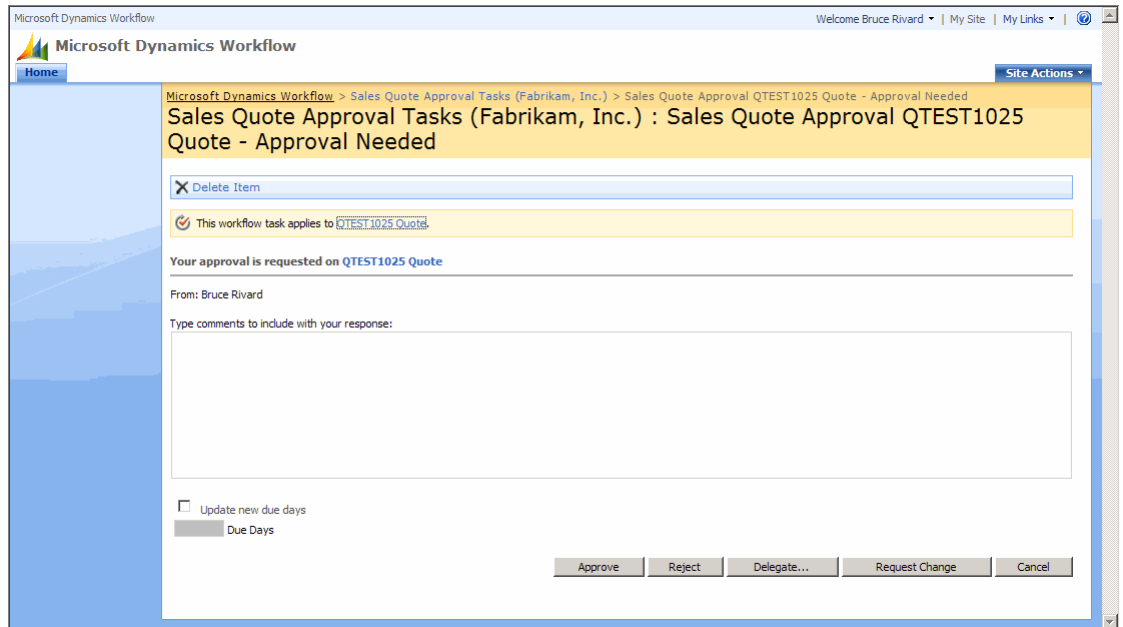


You will learn more about the client workflow assembly in [Chapter 7, “Client Workflow Assembly.”](#)

SharePoint client

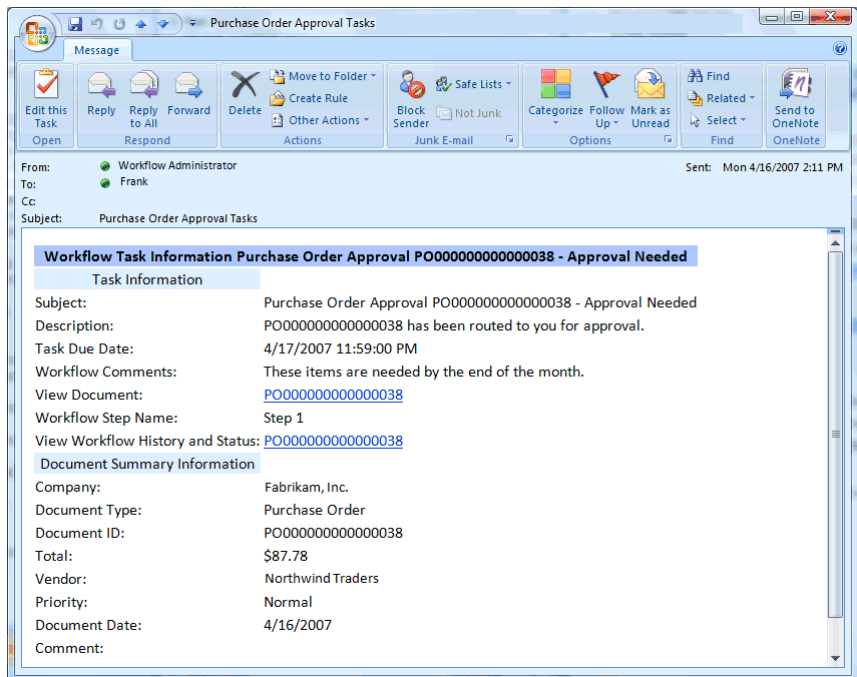
When Workflow for Dynamics GP is installed, the Microsoft Dynamics Workflow web site is created. This web site is built on MOSS and can be accessed with Microsoft Internet Explorer. The web site allows administrators to configure and manage workflows. The Microsoft Dynamics Workflow site also allows users to view and approve documents and batches that are assigned to them.

The following shows the SharePoint approval task for the Sales Quote approval workflow.



Outlook client

When a document or batch is assigned to a user for approval, an e-mail message can be sent to that approver. The e-mail message displays information about the document or batch. The **Edit this Task** button allows the user to approve or reject the document or batch.



Chapter 2: Integration Types

Microsoft Dynamics GP Workflow allows you to create a new workflow for back-office documents or to extend an existing workflow. Information about both types of workflow integration is contained in the following sections:

- [Creating a new workflow](#)
- [Extending an existing workflow](#)

Creating a new workflow

You can add Workflow support to back-office documents for Microsoft Dynamics GP. If a document requires one or more individuals to review and approve a new or updated document, it is a candidate to have workflow support added for it. For detailed information about creating and deploying a new workflow, see [Part 2, Creating a New Workflow](#).

To integrate a back-office document with Workflow, you complete the following tasks:

1. Make application dictionary changes.

Create or update the Microsoft Dynamics GP form that displays your document. You modify the form to support the workflow process and to enable the workflow client interface. For additional information about the dictionary changes, see [Chapter 6, "Dictionary Changes."](#)

2. Add workflow columns to the database.

Add columns required by workflow to the data table where you store your document data. To learn about the data table change, see [Tables](#) on page 43 of [Chapter 6, "Dictionary Changes."](#)

3. Add a web service.

Create a web service that provides access to the new back-office document. For additional information about creating a web service, see [Chapter 8, "Web Service."](#)

4. Create a client workflow assembly.

Create a .NET assembly for the Microsoft Dynamics GP client. The assembly works with your Microsoft Dynamics GP form to display the workflow client interface and to populate that interface with workflow information for the current document. To learn about the client workflow assembly, see [Chapter 7, "Client Workflow Assembly."](#)

5. Create a server workflow assembly.

Create a .NET assembly for the Microsoft Dynamics GP server that defines your workflow type. The server assembly retrieves and updates the document when specified workflow events occur. For additional information about the server workflow assembly, see [Chapter 9, "Server Workflow Assembly."](#)

6. Create a document viewer.

Create a document viewer that displays key information from your back-office document in the workflow web client. To learn about workflow document viewers, see [Creating a document viewer](#) on page 155 of [Chapter 9, "Server Workflow Assembly."](#)

7. Update the Dynamics Security Service.

Update security roles in the Dynamics Security Service that enable workflow access to Microsoft Dynamics GP web services. For information about updating the Dynamics Security Service, see [Creating a workflow event subscription helper application](#) on page 160 of [Chapter 9, "Server Workflow Assembly."](#)

8. Update Microsoft SQL server security

Give the DYNWORKFLOWGRP database role permission to access tables in the Microsoft Dynamics GP database. Add tables to this role where you added workflow information. For information about updating SQL server security, see [Table security](#) on page 44 of [Chapter 6, "Dictionary Changes."](#)

9. Deploy the server workflow assembly.

To add your new workflow, you must install and register your server assembly on the server where you have installed MOSS and Workflow. For additional information about deploying server workflow assembly, see [Chapter 10, "Deploying the New Workflow."](#)

10. Deploy the web service.

Install the new web service on the Web Services for Microsoft Dynamics GP server. For information about installing the web service, see [Chapter 10, "Deploying the New Workflow."](#)

11. Deploy the client workflow assembly.

To enable the Microsoft Dynamics GP client to use your new workflow, install your client dictionary changes and the client assembly on each of Microsoft Dynamics GP client machine. You must also edit the Dynamics.exe.config file of each client. For additional information about deploying the client workflow assembly, see [Chapter 10, "Deploying the New Workflow."](#)

Extending an existing workflow

Microsoft Dynamics GP installs workflow support for several existing document types. The existing workflow types are as follows:

Workflow type	Microsoft Dynamics GP form
Purchase order approval	Purchase Order Entry
Sales quote approval	Sales Transaction Entry
Customer credit limit override approval	Sales Transaction Entry
General ledger batch approval	Batch Entry
Payables Management batch approval	Payables Batch Entry
Receivables Management batch approval	Receivables Batch Entry

If you have a Microsoft Dynamics GP integration that adds one or more data fields to one of these Dynamics GP documents, use a workflow extension to add your data to the existing workflow. For detailed information on extending an existing workflow, see [Part 3, Extending an Existing Workflow.](#)

To extend an existing workflow, complete the following tasks:

1. Create a web service extension.

Create a web service extension that allows the Microsoft Dynamics GP web service to retrieve data from your new data store. For information about the web service extension, see [Chapter 12, "Web Service Extension."](#)

2. Create a workflow extension assembly.

Create a .NET assembly for the Microsoft Dynamics GP server that includes your new data fields with the existing Workflow document. For additional information, see [Chapter 13, "Workflow Extension Assembly."](#)

3. Create an "installation helper" application.

Create a helper application that registers the workflow extension assembly. For additional information, see [Chapter 14, "Deploying the Workflow Extension."](#)

4. Deploy the workflow extension assembly.

To deploy a workflow extension assembly, install the workflow extension assembly on the Workflow server. To register the assembly, install and run the "installation helper." For additional information about deploying a workflow extension assembly, see [Chapter 14, "Deploying the Workflow Extension."](#)

Chapter 3: Sample Workflow Type

Included with the Microsoft Dynamics GP Workflow SDK is a sample application that demonstrates common techniques you will use in your own workflow integrations. The sample application integrates a new document type with Microsoft Dynamics GP and uses workflow to approve changes to the document. Information about the Microsoft Dynamics GP Workflow SDK sample application is divided into the following sections:

- [*Workflow type sample overview*](#)
- [*Workflow type sample files*](#)
- [*Installing the sample application*](#)
- [*Installing the document type assembly*](#)
- [*Installing the Sales Lead web service*](#)
- [*Updating the Dynamics Security Service*](#)
- [*Installing eConnect Transaction Requester components*](#)
- [*Installing the server workflow assembly*](#)
- [*Installing the document viewer*](#)
- [*Installing the client workflow assembly*](#)
- [*Viewing the workflow type application*](#)

Workflow type sample overview

The Microsoft Dynamics GP Workflow SDK includes a sample application (Develop.cnk) that integrates with Microsoft Dynamics GP. The Microsoft Dynamics GP Workflow SDK also includes a collection of Visual Studio projects that allow the sample application to integrate with Microsoft Dynamics GP Workflow.

The sample application includes a Lead Maintenance form used to track potential customer leads. The sample uses workflow to approve the assignment of a salesperson to a lead.

- Workflow ensures a specified approver reviews and approves the salesperson initially assigned to a new lead.
- Workflow ensures an approver reviews changes to the salesperson assigned to a lead. If the approver rejects the change, the lead is automatically reassigned to the last approved salesperson.

Since leads are a new document type for Microsoft Dynamics GP, Web Services for Microsoft Dynamics GP does not include web methods that retrieve them. To access lead data, the Microsoft Dynamics GP Workflow SDK sample includes a Visual Studio project that builds a Sales Lead web service. The Sales Lead web service extends the Dynamics GP web services and provides the following capabilities:

- The Sales Lead web service allows workflow to retrieve individual lead documents. The Workflow uses the web service GetByKey method to retrieve a lead document whenever it needs lead data from the Microsoft Dynamics GP database.
- The Sales Lead web service uses existing Dynamics Security Service roles to secure access to the web service. This allows you to manage access to new web service methods using the Dynamics Security Service.

- The Sales Lead web service uses Microsoft Dynamics GP web service context objects to specify how a web service call should be performed. For example, use OrganizationKey property of the context object to specify the Microsoft Dynamics GP company database that stores the lead document.

The SDK includes a Visual Studio project that builds a client workflow assembly. The assembly adds workflow support to the Lead Maintenance and Lead list created by the sample application.

The SDK includes a Visual Studio project that builds a server workflow assembly. The assembly adds workflow approval support for lead documents. The project includes a helper application that registers the new workflow type.

The server workflow assembly project also includes a document viewer that displays a lead document in the Workflow web client.

Workflow type sample files

To demonstrate a workflow integration, the Microsoft Dynamics GP Workflow SDK includes the files and Visual Studio projects needed to build and install the lead sample application and workflow. These files are in an archive file named Samples.

The SDK's workflow type sample includes the following files and Visual Studio projects.

Category	File or project name	Description
GP client components	Application.SampleIntegratingApp.dll	A Visual Studio tools for Microsoft Dynamics GP assembly that allows the client workflow assembly to work with the Lead Maintenance and Lead List forms.
GP client components	Develop.cnk	A file that installs the Lead Maintenance sample application for Microsoft Dynamics GP.
Web services	Microsoft.Dynamics.GPSamples, SalesLeads.LeadGetByKey.xslt	An XSLT file that allows web services to retrieve Lead data using the eConnect Transaction Requester.
Web services	WSLead.sql	A SQL script that inserts a record into the eConnect_Out_Setup table of the TWO database.
Web services	InstallSalesLeadSecurityMetadata	A Visual Studio C# project. The project produces an application that adds operations and tasks to the Dynamics Security Service for securing the Leads web service.
Web services	Microsoft.Dynamics.GPSamples.SalesLeads	A Visual Studio C# project. The project produces an assembly that defines a Lead document type for use with web services and workflow.
Web services	SampleSalesLeadWebService	A Visual Studio C# web service project. The project produces a .asmx file, and assembly that enables you to access lead data through a web service.
Web services	WebServiceTest	A Visual Studio C# console application project. The project produces a test application you can use to verify that the Sales Lead web service is working.
Workflow	WorkflowClientSample	A Visual Studio C# project. The project produces an assembly that adds workflow functionality to the Lead Maintenance and Lead List forms created by the sample application.
Workflow	WorkflowServerSample	A Visual Studio C# project. The project that produces an assembly that add the Microsoft Dynamics GP approval workflow to leads.

To install the SDK workflow type sample, extract the files and projects from the archive. Extract the project folders to a workflow server with Visual Studio installed. Use Visual Studio with the project solution files to build the sample's assembly and application files.

Installing the sample application

To use the workflow sample application, you must first install the back-office sample application. The sample application contains two forms to which workflow support has been added: the Lead Maintenance form and the Lead list form.

To install the back-office sample application, complete the following procedure:

1. Install the sample application files.

Copy the Application.SampleIntegratingApp.dll, and Develop.cnk files to the Microsoft Dynamics GP client install folder. The client install is typically in the location:

C:\Program Files\Microsoft Dynamics\GP

2. Start Microsoft Dynamics GP

A message will be displayed asking whether you want to include new code. Click Yes. When the Login window appears, log in as the System Administrator (sa), or DYNSA.



You must log in as the System Administrator (sa) or DYNSA the first time after installing the sample, so the application's SQL tables will be properly created.

3. Log into the sample company.

Choose to log into the sample company, Fabrikam, Inc.

4. Verify the installation.

If the sample application was installed properly, the IG Sample toolbar will be displayed in Microsoft Dynamics GP. You can open the Lead Maintenance form by clicking its associated icon on the IG Sample toolbar.

For additional information about installing the back-office sample application, see the Microsoft Dynamics GP Integration Guide that is included with Dexterity.

Installing the document type assembly

To use the web service to access the data associated with the back-office sample application, you must install the .NET assembly that defines a lead.

To install the sample assembly that defines leads, complete the following procedure:

1. Open the solution file with Visual Studio.

Open the solution file Microsoft.Dynamics.GP.Samples.SalesLeads.sln with Visual Studio.

2. Verify References for the project.

Use Solution Explorer to verify the References. Reload any references that are not able to find the specified .dll file. To reload a reference, browse to the Dynamics GP web services "bin" folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

3. Build the assembly.

From the Build menu, choose Build Solution. Visual Studio builds the assembly in the project's "\bin\debug" folder.

4. Copy the assembly to the web services server.

Copy the Microsoft.Dynamics.GP.Samples.SalesLeads.dll file to the server where you installed Web Services for Microsoft Dynamics GP. Place the .dll in the Dynamics GP web service "bin" folder, typically found in this location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

Installing eConnect Transaction Requester components

The Sales Lead web service uses the eConnect Transaction Requester to retrieve data from the Microsoft Dynamics GP database. To add your new document type to the eConnect Transaction Requester, complete the following procedure:

1. Update the eConnect_Out_Setup table in the TWO database.

Copy the WSLead.sql file to your Dynamics GP data server. Open the WSLead.sql file with Microsoft SQL Server Management Studio. From the Query menu, choose Execute. The query adds a new entry to the eConnect_Out_Setup table in the TWO database.

2. Install the XSLT file to the web server.

Copy the Microsoft.Dynamics.GP.Samples.SalesLeads.LeadGetByKey.xslt file to the web server where you installed the Web Services for Microsoft Dynamics GP. Place the file in the Dynamics GP web service "XSLT" folder, typically found in this location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin\XSLT

Installing the Sales Lead web service

To enable access to the back-office sample application's data, install a web service that returns a specified lead document. To install the web service, complete the following procedure:

1. Open the solution file with Visual Studio.

Open the SampleSalesLeadWebService.sln with Visual Studio.

2. Verify References for the project.

Use Solution Explorer to verify the References. Reload any references that are not able to find the specified .dll file. To reload a reference, browse to the Dynamics GP web services "bin" folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

3. Build the application.

From the Build menu, choose Build SampleSalesLeadWebService. Visual Studio builds the web service SampleSalesLeadWebService.dll assembly in the project's "bin" folder.

4. Install the web service file.

Copy the SampleSalesLeadWebService.asmx file to the server where you installed Web Services for Microsoft Dynamics GP. Place the SampleSalesLeadWebService.asmx file in the Dynamics GP web service “WebServices” folder, typically found in this location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices

To help secure the web service, use the security properties of the SampleSalesLeadWebService.asmx file to specify user access permissions. To enable users of Microsoft Dynamics GP web services to also access this web service, right-click the SampleSalesLeadWebService.asmx file, and choose Properties. When the Properties window opens, click the Security tab, and then click Add. In the Select Users, Computers, or Groups window, type Authenticated Users into the Enter the object names to select box, and then click OK. Mark the Allow check box for Read, and Read & Execute permissions. Click OK to close the Properties window.

5. Install the web service assembly.

Copy the SampleSalesLeadWebService.dll assembly to the server where you installed Web Services for Microsoft Dynamics GP. Place the SampleSalesLeadWebService.dll assembly in the Dynamics GP web services “bin” folder, typically found in this location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\bin

6. Test the web service (optional).

If you want to verify that the web service is working properly, you can use the WebServiceTest sample. This is a console application that calls the Sales Lead web service, retrieves a specified lead document, and then displays the name of the lead.

Before you run the WebServiceTest sample, you might need to update the web reference to use the URL of your Sales Lead web service. You might also need to change the value supplied for the LeadKey Id.

Updating the Dynamics Security Service

To prevent unauthorized access of the Sales Lead web service, run the following “security helper” application:

Microsoft.Dynamics.GP.Samples.InstallSalesLeadSecurityMetadata.exe

The sales lead “security helper” application adds three security operations and a security task to the Dynamics Security Service. The application also assigns the new security operations to the Dynamics Security Service Superuser role

To build and run this application, complete the following procedure:

1. Open the solution file with Visual Studio.

Open the following solution file:

Microsoft.Dynamics.GP.Samples.InstallSalesLeadSecurityMetadata.sln

2. Verify References for the project.

Use Solution Explorer to verify the References. Reload any references that are not able to find the specified .dll file. To reload a reference, browse to the Dynamics GP web services “bin” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

3. Build the application.

From the Build menu, choose Build Solution. Visual Studio builds the application file and the application configuration file in the project’s “\bin\debug” folder.

4. Install the application and application configuration files.

Find the following application and application configuration files:

```
Microsoft.Dynamics.GP.Samples.InstallSalesLeadSecurityMetadata.exe
Microsoft.Dynamics.GP.Samples.InstallSalesLeadSecurityMetadata.exe.config
```

Copy both files to the Dynamics GP web service “GPWebServices” folder, typically found in this location:

C:\Program Files\Microsoft Dynamics\GPWebServices

5. Run the application.

Open a command prompt and change the working location to the folder where you installed the “security helper” application.



The Dynamics Security Service uses Active Directory Application Mode (ADAM) to store information about web service security roles, operations, and tasks. To run the “security helper” application, you must be logged in as user who is an ADAM administrator. If you run the “security helper” application using a login that is not an ADAM administrator, ADAM will return a “System.UnauthorizedAccessException” error. By default, the user who installed Dynamics GP web services is an ADAM administrator. To add another user as an ADAM administrator, see the Web Services Installation and Administration Guide.

Type the following command and press Enter:

```
Microsoft.Dynamics.GP.Samples.InstallSalesLeadSecurityMetadata.exe /load
```

To secure access to the Sales Lead web service, the helper application creates three security operations, adds each operation to the “View Sales Leads” task, and then adds the security operations to the Dynamics Security Service Superuser role.

To verify that the security helper application succeeded, open the Dynamics Security Console, expand the DynamicGPWebServices node, click Roles, click the Superuser role, and then click Properties. The list of tasks and operations should include operations named Get Leads List, Get Sales Lead, and Query Sales Leads.

To permit other Dynamics Security Service roles to access the Sales Lead web service, use the Dynamics Security Console to assign the “View Sales Leads” task to each role. Another option is to create additional helper applications that update Dynamics Security Service roles. For example, the next section includes an event subscription helper application that adds the “View Sales Leads” security task to the Dynamics Security Service Workflow Administrator role.

Installing the server workflow assembly

To allow Microsoft Dynamics GP Workflow to manage the approval of salesperson assignments on leads, you must install the server workflow assembly on your workflow server.

You will also build and run the event subscription helper application that subscribes to the workflow StatusChange event. The event subscription helper application also demonstrates how a helper application can add the “View leads” security task to the Dynamics Security Service Workflow Administrator role.

To install the server workflow assembly on the workflow server, complete the following procedure:

1. Open the server workflow solution file.

Open WorkflowServerSample.sln file using Visual Studio.

2. Verify references for the project.

Use Solution Explorer to verify the References. Reload any references that are not able to find the specified assembly. To reload a reference, look for the assembly in one of the following locations:

```
C:\Program Files\Common Files\Microsoft Shared\Web Server
Extensions\12\ISAPI
C:\Program Files\Microsoft Dynamics\Workflow
C:\inetpub\wwwroot\wss\VirtualDirectories\

```

For the port number, specify the port number of the web site where you installed Workflow for Microsoft Dynamics GP.

3. Update the web reference.

Use Solution Explorer to expand the Web References folder. Right-click the SalesLeadWebService and choose Properties. Update the URL property to specify the location of the Sales Lead web service.

To update the web reference, right-click SalesLeadWebService, and choose Update Web Reference.

4. Update the app.config file

Use Solution Explorer to open the Microsoft.Dynamics.Workflow.Sample.RegisterSalesLeads folder. Open the App.config file. Verify that the value of the BusinessObjectsConfiguration key in the appSetting section points to the location of the DynamicsWorkflowEvents.config file. The DynamicsWorkflowEvents.config file is typically found in the following location:

```
C:\inetpub\wwwroot\wss\VirtualDirectories\port#\bin\
```

The port number specifies the port associated with the web site where you installed Workflow for Microsoft Dynamics GP.

5. Build the server workflow assembly and the event subscription helper application.

From the Build menu, choose Build Solution. This creates the following files:

```
Microsoft.Dynamics.GP.Workflow.Samples.Server.dll
Microsoft.Dynamics.Workflow.Sample.RegisterSalesLeads.exe
Microsoft.Dynamics.Workflow.Sample.RegisterSalesLeads.exe.config
```

6. Install the assembly to the workflow server.

To install the workflow type assembly, copy the files from the Visual Studio project's "\bin\debug" folders to the following folders:

- Place the following files in the workflow installation folder:

```
Microsoft.Dynamics.GP.Workflow.Samples.Server.dll
Microsoft.Dynamics.Workflow.Sample.RegisterSalesLeads.exe
Microsoft.Dynamics.Workflow.Sample.RegisterSalesLeads.exe.config
```

The workflow installation folder is typically found in this location:

```
C:\Program Files\Microsoft Dynamics\Workflow.
```

- Place the following file in the "bin" folder of the virtual directory for the Microsoft Dynamics GP Workflow web site:

```
Microsoft.Dynamics.GP.Workflow.Samples.Server.dll
```

The "bin" folder is typically found in this location:

```
C:\inetpub\wwwroot\wss\VirtualDirectories\port#\bin
```

You will need to know the port number you are using for the virtual directory that the Workflow web service was installed into.

7. Add the server workflow assembly to the global assembly cache.

On your workflow server, use Microsoft Windows Explorer to open the Assembly folder. Typically, this will be the following:

```
C:\Windows\Assembly
```

Drag and drop your server workflow assembly file into the Assembly folder.

8. Register the workflow schedule.

On the workflow server, open a command prompt and set the working directory to the “Workflow” folder, typically found in this location:

```
C:\Program Files\Microsoft Dynamics\Workflow
```

Type the following command and press Enter.

```
Microsoft.Dynamics.Workflow.Install.RegisterSchedule.exe /ACTION=LOAD
/DATABASE=DYNAMICS /ASSEMBLY=Microsoft.Dynamics.GP.Workflow.Samples.Server
```

Leave the command prompt open for use with the following step.

9. Register the server workflow assembly.

From the command prompt that was opened in the previous step, type the following command and press Enter.

```
Microsoft.Dynamics.Workflow.Install.ApplicationServer.exe /ACTION=LOAD
/ASSEMBLY=Microsoft.Dynamics.GP.Workflow.Samples.Server /CLASSES=
Microsoft.Dynamics.GP.Workflow.Samples.Server.LeadApprovalWorkflowSample
/COREPERFORMANCECOUNTERS=true /WORKFLOWPERFORMANCECOUNTERS=true
/DYNAMICSEVENTLOGSOURCES=true
```

10. Run the event subscription helper application.

Open a command prompt and set the working directory to the folder where you installed the event subscription helper application.



To avoid creating duplicate event subscriptions, take care to run the event subscription helper application one time. The application does not verify whether the specified event handler exists and will create duplicate event handlers in the configuration file.

To add your event to the DynamicsWorkflowEvents.config file, use the following command to run your event subscription helper application:

```
Microsoft.Dynamics.Workflow.Sample.RegisterSalesLeads.exe
```

For additional information about installing the subscription application, see [Installing the server workflow assembly](#) in [Chapter 10, “Deploying the New Workflow.”](#)

Installing the document viewer

Copy the Dynamics.Workflow.GP.Samples.SalesLeadViewer.aspx file to the Microsoft Office SharePoint Server “layouts” folder. Typically, the folder is found in the following location:

```
C:\Program Files\Common Files\Microsoft Shared\web server
extensions\12\templates\layouts
```

Installing the client workflow assembly

To use workflow with the Lead Maintenance and Lead list forms, install the client workflow assembly on a machine where you installed the sample application for the Microsoft Dynamics GP client.

To install the assembly on a client machine, complete the following procedure:

1. Open the client workflow solution file.

Open `Microsoft.Dynamics.GP.Workflow.Samples.Client.sln` using Visual Studio.

2. Verify references for the project.

Use Solution Explorer to verify the References. Reload any references that are not able to find the specified .dll file.

3. Build the client workflow assembly

From the Build menu, choose Build Solution. This creates the `Microsoft.Dynamics.GP.Workflow.Samples.Client.dll` in the project's "`\bin\debug`" folder.

4. Copy the assembly to the client machine.

Copy the `Microsoft.Dynamics.GP.Workflow.Samples.Client.dll` to the Dynamics GP client directory, typically found at the following location:

`C:\Program Files\Microsoft Dynamics\GP`

5. Update the Dynamics.exe.config file.

On the client machine, use Microsoft Windows Explorer to open the Dynamics GP client folder, typically found at the following location:

`C:\Program Files\Microsoft Dynamics\GP`

Make a backup copy of the `Dynamics.exe.config` file and store it in a safe location.

Open the `Dynamics.exe.config` file with a text editor and add the following two entries to the `<formFactories>` node:

```
<add name="Workflow-SalesLeads" productId="3333" formId="22000"
factoryType="Microsoft.Dynamics.GP.Workflow.Samples.Client.LeadWorkflowFo
rmFactory,Microsoft.Dynamics.GP.Workflow.Samples.Client" />
<add name="Workflow-SampleWorkflowStatus" productId="3333" formId="22002"
windowId="22001" factoryType="Microsoft.Dynamics.GP.Workflow.Samples.Clien
t.WorkflowStatusFormFactory,Microsoft.Dynamics.GP.Workflow.Samples.Client
" />
```

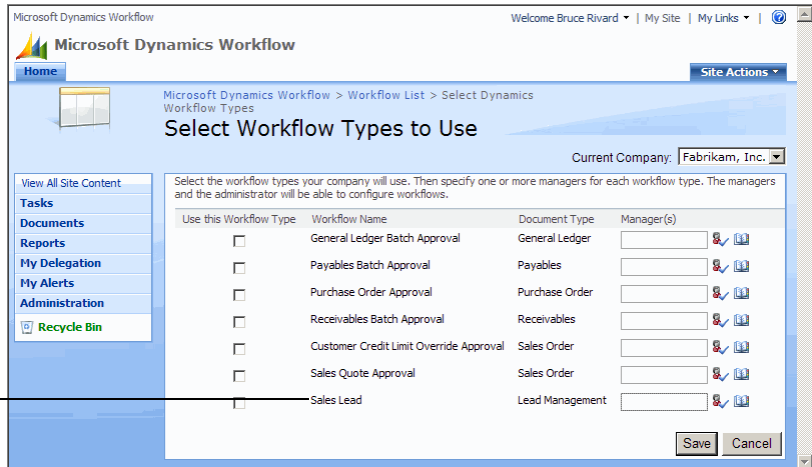
These entries create form factories that are used to add workflow actions to the Lead Maintenance and Lead list forms. The form factories also verify the status of the Sales Leads workflow when a user logs into Microsoft Dynamics GP.

Viewing the workflow type application

After you complete all the steps to install the application, the web service, and the workflow assemblies, open a command prompt and run `iisreset.exe` on your workflow server and your Web Services for Microsoft Dynamics GP server.

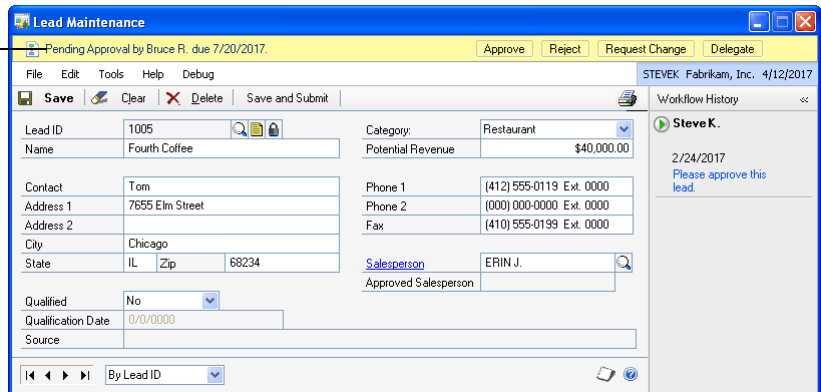
Use Microsoft Internet Explorer to open the Microsoft Dynamics Workflow site. Click Administration to view the Workflow List. Click Select Workflow Type to Use. The list of Workflow Names includes the sample application's Sales Lead workflow.

This is the workflow for the sample application.



Enable the Sales Lead workflow and click Save. Use the Microsoft Dynamics Workflow Administration site to configure a simple workflow with a single approver step. Open the Lead Maintenance form, create a new lead, assign a salesperson, and click Save and Submit. Workflow makes the lead available to the specified approver.

The workflow status of the lead.



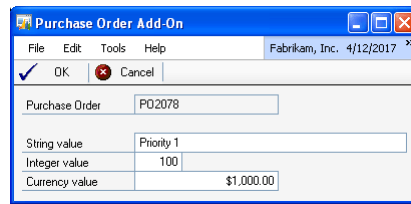
Chapter 4: Sample Workflow Extension

Included with the Microsoft Dynamics GP Workflow SDK is a sample application that demonstrates how to extend an existing workflow. Information about the sample workflow extension is divided into the following sections:

- [Workflow extension sample overview](#)
- [Workflow extension sample files](#)
- [Installing the purchase order integration sample](#)
- [Installing the web service extension sample](#)
- [Installing the workflow extension sample](#)
- [Viewing the workflow extension sample](#)

Workflow extension sample overview

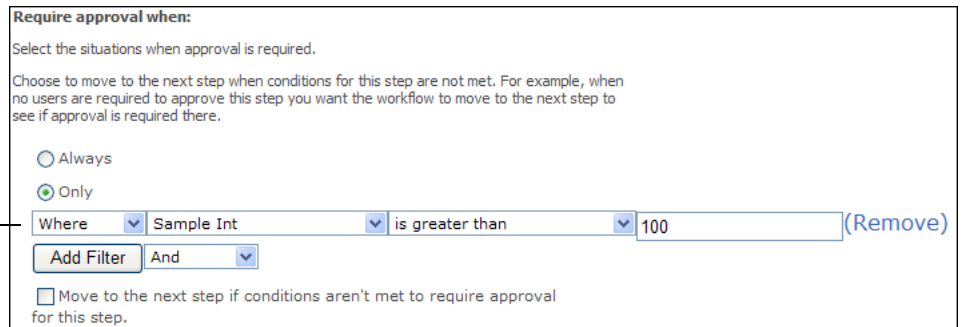
The sample workflow extension consists of a Dexterity-based integration that makes additional properties available for purchase order documents. These additional properties are accessed through an additional window opened from the Purchase Order Entry window. This additional window is shown in the following illustration.



Purchase order documents can be accessed through the Dynamics GP web service. A web service extension assembly makes the additional data from the sample purchase order integration available for each purchase order document.

An approval workflow is available for purchase order documents. The sample workflow extension makes the additional data values from this sample integration available to the purchase order workflow. The data can be used by the workflow administrator as they set up the approval steps for the workflow.

Data made available by the workflow extension can be used when defining a workflow step.



The additional data can also be made available in the business document summary that is included with each workflow notification.

Workflow extension sample files

The Microsoft Dynamics GP Workflow SDK includes the files and Visual Studio projects needed to build and install the sample purchase order workflow extension for the sample purchase order integration. These files are in an archive file named Samples.

The Samples archive contains the following files and Visual Studio projects for the sample workflow extension:

File	Description
POPAddOn.cnk	A file that installs the sample purchase order integration for Microsoft Dynamics GP.
POExtension	A Visual Studio C# project. This project produces a web service extension assembly that provides access to the additional data made available by the purchase order sample integration.
ExtensionTest	A Visual Studio C# project. This project is a test application that verifies the web service extension is working properly for purchase order documents.
POWorkflowExtension	A Visual Studio C# project. This project produces the workflow extension assembly that makes the additional data from the purchase order integration available to the purchase order workflow.
WFExtensionInstaller	A Visual Studio C# project. This project produces an installation helper application that registers the sample workflow extension.

Extract these files and projects from the archive.

Installing the purchase order integration sample

To use the purchase order integration sample, you must first install it for Microsoft Dynamics GP. This sample application implements the data storage for the additional purchase order data. It also provides the form used to access this data from the Purchase Order Entry window.

To install the purchase order sample integration, complete the following procedure:

1. Install the sample application file.

Copy the POPAddOn.cnk file to the Microsoft Dynamics GP client install folder. The client install is typically in the location:

C:\Program Files\Microsoft Dynamic\GP

2. Start Microsoft Dynamics GP.

A message will be displayed asking whether you want to include new code. Click Yes. When the Login window appears, log in as the System Administrator (sa) or DYNSA.



You must log in as the System Administrator (sa) or DYNSA the first time after installing the sample, so the application's SQL table will be properly created.

3. Log into the sample company.

Choose to log into the sample company, Fabrikam, Inc.

4. Verify the installation.

If the sample application was installed properly, you should see an “Additional” menu when you open the Purchase Order Entry window in Microsoft Dynamics GP. This menu will contain a menu item you can use to open the Purchase Order Add-On window.

Installing the web service extension sample

The web service extension allows the purchase order document to retrieve the additional data from the sample purchase order integration. You must install the .NET assembly that defines the web service extension. To do this, complete the following procedure:

1. Open the solution file with Visual Studio.

Open the solution file POExtension.sln with Visual Studio.

2. Verify References for the project.

Use the Solution Explorer to verify the References. Reload any references that are not able to find the specified .dll file. The needed assemblies are typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

3. Build the assembly.

From the Build menu, choose Build Solution. Visual Studio builds the assembly in the project’s “\bin\debug” folder.

4. Copy the assembly to the web services server.

Copy the POExtension.dll file to the server where you installed Web Services for Microsoft Dynamics GP. Place the .dll in the Dynamics GP web service “bin” folder, typically found in this location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

5. Make a backup of the BusinessObjectFile.config file.

You must add registration information to the BusinessObjectFile.config file for the web service extension to be found. This file is in XML format. It is located in the Dynamics GP web service “bin” folder, typically found in this location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

Make a copy of the BusinessObjectFile.config file. Store the backup copy in a safe location.

6. Register the web service extension.

Using a text editor, open the BusinessObjectFile.config file. Locate the dictionary entry for the Purchase Order document:

```
<DictionaryEntry>
  <Key xsi:type="xsd:string">Microsoft.Dynamics.GP.PurchaseOrder</Key>
  <Value xsi:type="BusinessObjectConfiguration">
```

Add the following event registration to the end of this section:

```
<Event>
  <EventName>Retrieved</EventName>
  <EventHandlerType>
    <Type>Microsoft.Dynamics.Common.BusinessObjectEventHandler</Type>
    <Assembly>Microsoft.Dynamics.Common</Assembly>
  </EventHandlerType>
  <EventHandler>
    <SoftwareVendor>MicrosoftDocumentation</SoftwareVendor>
    <Type>POExtension.PurchaseOrderExtensionEventHandler</Type>
    <StaticMethod>Retrieved</StaticMethod>
    <Assembly>POExtension</Assembly>
    <Execute>>true</Execute>
  </EventHandler>
</Event>
```

Save the changes to the file.

7. Reset IIS for the server running the Dynamics GP web service.

Perform an “iisreset” command to be sure the web service extension has been loaded into memory.

8. Test the web service extension (optional).

If you want to verify that the web service extension is working properly, you can use the ExtensionTest sample. This is a console application that calls the Dynamics GP web service, retrieves the specified purchase order document, and then displays the additional data added by the purchase order web service extension.

Installing the workflow extension sample

The workflow extension makes the additional data from the purchase order extension available to the purchase order workflow. You must install and register the .NET assembly that defines the workflow service extension. To do this, complete the following procedure:

1. Open the solution file with Visual Studio.

Open the solution file POWorkflowExtension.sln with Visual Studio.

2. Verify References for the project.

Use the Solution Explorer to verify the References. Reload any references that are not able to find the specified .dll file. The following assemblies are referenced from Dynamics GP Workflow.

- Microsoft.Dynamics.Common
- Microsoft.Dynamics.Common.Types
- Microsoft.Dynamics.Workflow
- Microsoft.Dynamics.Workflow.Controls

These assemblies are typically found in this location:

C:\Program Files\Microsoft Dynamics\Workflow

The following additional assemblies for Dynamics GP Workflow must also be referenced:

- Microsoft.Dynamics.Workflow.Common
- Microsoft.Dynamics.GP.WebServices.Proxy

These assemblies are typically found in the “bin” folder of the virtual directory used by the Workflow web service:

```
C:\Inetpub\wwwroot\wss\VirtualDirectories\port#\bin
```

You will need to know the port number you are using for the virtual directory that the Workflow web service is installed into.

3. Build the assembly.

From the Build menu, choose Build Solution. Visual Studio builds the assembly in the project’s “\bin\debug” folder.

4. Add the workflow extension assembly to the global assembly cache.

On the Workflow server, use Microsoft Windows Explorer to open the Assembly folder. Typically, this will be in the following location:

```
C:\Windows\Assembly
```

Drag and drop the workflow extension assembly into the Assembly folder.

5. Add the web service extension assembly to the global assembly cache.

The web service extension assembly (POExtension.dll) must also be added to the global assembly cache so that it can be found by Workflow. Locate the POExtension.dll assembly and drag it to the Assembly folder to add it to the global assembly cache.

6. Locate the workflow extension installer application.

A “helper” application is needed to register the workflow extension assembly. A compiled version of this application is found in the “\bin\debug” folder for the WFExtensionInstaller project. Open the folder that contains the installer application.

7. Edit the App.config file for the installer application.

The App.config file for the workflow extension installer application contains the path needed to access a Workflow configuration file. Edit the setting in this file to specify the path to the Workflow configuration file.

```
<add key ="BusinessObjectsConfigurationPath" value=
"C:\Inetpub\wwwroot\wss\VirtualDirectories\10072\bin\
DynamicsWorkflowEvents.config " />
```

The standard path to the Workflow web service is shown above. You will need to update the path to use the port number (likely something other than 10072) that is being used for the Workflow web service.

Save the changes to the App.config file.

8. Run the workflow extension installer.

The application will locate the DynamicsWorkflowEvents.config file, and add a large entry to it that defines the necessary events for your workflow extension.

Viewing the workflow extension sample

You can see the additional properties made available by the workflow extension when setting up a step for a purchase order workflow. The sample workflow extension makes three additional properties from the purchase order integration available.

The additional properties from the workflow extension will appear in this list.

Require approval when:

Select the situations when approval is required.

Choose to move to the next step when conditions for this step are not met. For example, when no users are required to approve this step you want the workflow to move to the next step to see if approval is required there.

Always
 Only

Where is greater than (Remove)

And

Move to the next step if conditions aren't met to require approval for this step.

You will also be able to see two of the additional properties that were added to the summary document that is included in purchase order workflow notifications.

Part 2: Creating a New Workflow

This portion of the documentation describes how to build a new approval workflow type for Microsoft Dynamics GP Workflow. The following information is discussed:

- [Chapter 5, “Designing a New Workflow.”](#) describes key design elements to consider as you plan your new workflow type.
- [Chapter 6, “Dictionary Changes.”](#) describes changes required for a Dexterity-based integration for Microsoft Dynamics GP so it can integrate with Microsoft Dynamics GP Workflow.
- [Chapter 7, “Client Workflow Assembly.”](#) describes how to create a new .NET assembly allows an integrated application to use Microsoft Dynamics GP Workflow.
- [Chapter 8, “Web Service.”](#) describes how to create a new web service to retrieve data used by the new workflow type.
- [Chapter 9, “Server Workflow Assembly.”](#) describes how to create a .NET assembly that integrates with Microsoft Dynamics GP Workflow. The assembly handles the workflow tasks and provides any business logic associated with workflow events.
- [Chapter 10, “Deploying the New Workflow.”](#) describes how to install your new workflow integration.

Chapter 5: Designing a New Workflow

Several things must be considered when you are creating a new workflow type. This portion of the documentation describes what you need to think about as you create a new workflow type. It also describes how this design process was followed for the sample workflow type included with the Workflow for Microsoft Dynamics GP SDK. The following sections describe the process.

- [*Is a workflow appropriate?*](#)
- [*Microsoft Dynamics GP client*](#)
- [*Business logic*](#)
- [*Data access*](#)
- [*Workflow server*](#)

Is a workflow appropriate?

Workflow for Microsoft Dynamics GP implements a single workflow schedule. The Dynamics GP approval workflow schedule automates the routing of documents between specified individuals for their approval.

If you have a back-office document type that needs to be reviewed by one or more individuals pending their approval, that document type is a candidate for having workflow implemented.

For instance, the sample integrating application included with the Workflow for Microsoft Dynamics GP SDK contains the Lead Maintenance form used to track potential sales leads. Each lead is assigned to a salesperson. Since this assignment is significant for each salesperson, the initial assignments or changes to assignments should be approved by someone with proper authority. Implementing workflow for the Lead documents is an ideal way to manage this approval process.

Microsoft Dynamics GP client

When implementing workflow for a document type in Microsoft Dynamics GP, consider which windows and lists will have workflow support. Workflow support can be implemented for the following areas:

- The maintenance or transaction window for the document
- Any inquiry windows for the document
- Any lists that display the document

The sample workflow type implements workflow support for the Lead Maintenance window, as well as the Leads list. You will learn more about the details of implementing workflow support for windows and lists in [Chapter 6, "Dictionary Changes,"](#) and [Chapter 7, "Client Workflow Assembly."](#)

Business logic

When creating a new workflow, carefully consider how the business logic for the document will be impacted by the workflow. Identify the data changes that require review and approval. Your workflow implementation will need to handle cases like the following:

- What action should be taken when a document or transaction is approved in the workflow?
- What action should be taken when a document or transaction is rejected in the workflow?
- Should the workflow require approval of new documents or only when specific data fields are updated for the document?
- Should the document be validated before it is submitted to workflow?
- Should the window look or operate differently if workflow isn't enabled?

If many cases, the business logic for the window in Microsoft Dynamics GP that manages the document will need to be modified to support the workflow. This is described in [Window integration](#) on page 46. The server component of the new workflow will also implement business logic to support the workflow actions. This is explained in [Adding business logic](#) on page 149.

For the Sales Lead sample workflow, the business logic in the Lead Maintenance window was also modified to consider the workflow state. For instance, if the Sales Lead workflow was enabled, a new lead document had to be submitted to workflow using the "Save and Submit" button.

The server assembly for the sample workflow was coded to include the appropriate business logic to support the Sales Lead workflow. For example, when the assigned salesperson for a lead is approved in the workflow, the Approved Salesperson ID that is stored with each lead is updated to the salesperson that was assigned.

Data access

To be used with Workflow for Microsoft Dynamics GP, a document must be accessible through the Dynamics GP web service or through an additional web service that runs in conjunction with the Dynamics GP web service. If a document type for which you are implementing workflow isn't accessible through the web service, you must implement web service access for it. This process is described in detail in [Chapter 8, "Web Service."](#)



It's important that you properly secure any web service you create to access additional data. You want to limit data access only to authorized users.

The lead information for the sample integrating application is not available through the Dynamics GP web service. This required that the Sales Lead web service had to be created and installed alongside the Dynamics GP web service. The Dynamics Security Service (included with Web Services for Microsoft Dynamics GP) is used to secure the Sales Lead web service.

Workflow server

After the originator submits a document to workflow, the Workflow server guides the document through the approval process specified by the workflow's configuration. You will create several components that are used by the Workflow server as it manages the documents through the processes defined by your new workflow. [Chapter 9, "Server Workflow Assembly,"](#) describes in detail how you will create these components for the Workflow server. The following are some of the components you will build:

Server workflow assembly

The server workflow assembly defines the fundamental characteristics of your new workflow, such as:

- The workflow's name
- The properties available for filtering
- The data included in the summary document for the workflow's notifications

Give careful consideration to these values as you define your new workflow type.

The server workflow assembly implements various handlers that perform the processing required by the new workflow. For instance, one event handler manages the processing that occurs when the workflow state for a document is changed. The event handlers are where you will implement much of the business logic for your new workflow type.

For example, in the sample Sales Lead workflow we want to approve changes to the salesperson assigned to each lead document. If the approver rejects the change, we want to be able to restore the prior salesperson assignment. To perform this process, the server workflow assembly implements the following:

- When a new document is submitted, the assigned salesperson is copied to another field that stores the last approved salesperson.
- When an originator submits a document for which the salesperson has been changed, the data change is saved.
- If the approver approves the salesperson change, the last approved salesperson field is updated with the new value and saved.
- If the approver rejects the change, the salesperson field is restored to the previously approved value by using the value stored in the last approved salesperson field.

Workflow history

The server workflow assembly retrieves and saves workflow history information. To implement a server workflow assembly, you need to determine where to store your workflow history and how to access that store. To archive workflow history, choose one of the following:

- Microsoft Dynamics GP Workflow provides an archive that stores workflow history information. You can save and retrieve your workflow history information to that archive.
- Use the server workflow assembly to save and retrieve workflow history from a data store outside Microsoft Dynamics GP. To use an external data store, your server workflow assembly needs to access, update, and retrieve workflow history data from that store.

Document viewer

The document viewer is another server component you will create for your workflow type. It is an .aspx page used by SharePoint to display details about the documents being managed by your workflow. When approvers use the web or Outlook client to review the document, they will use the document viewer.



When defining the content that is shown for each document, be sure to include the data that approvers will need to make the approval decision

Chapter 6: Dictionary Changes

When creating a new workflow, several changes and additions are required in the Dexterity-based application dictionary. Some of these changes implement core portions of the new workflow. Other changes implement the client interface for the workflow in Microsoft Dynamics GP.

When implementing the dictionary changes for the new workflow, try to limit the impact on the application. If workflow isn't installed or activated, your application should operate essentially as it did before you made your dictionary changes.

Information about the needed dictionary changes is divided into the following sections:

- [Tables](#)
- [Table security](#)
- [Command form](#)
- [Window integration](#)
- [Notifications](#)
- [Home Page integration](#)
- [List integration](#)
- [Application assembly](#)

Tables

Update table definitions to contain workflow information.

Fields that store the workflow approval status and workflow priority information must be added to the main table (such as a master table) for the document to which workflow is being implemented. Microsoft Dynamics GP has defined the following global fields for this purpose:

Global field	Description
Workflow Approval Status	A drop-down list field that defines the various workflow approval states for the item. The possible values are: Not Submitted Submitted Not Needed Pending Approval Pending Changes Approved Rejected Ended Not Activated Deactivated
Workflow Priority	A drop-down list field that defines the workflow priority values for the item. The possible values are: Low Normal High

If you have created card lists or transaction lists for the item for which you are implementing workflow, you will need to add these global fields to the table definitions to the temporary tables used for the lists.

Additional table fields may also be needed to support the business logic associated with the workflow. This will depend on whether the workflow you're implementing needs to track additional information.

For the sample workflow integration, the workflow global fields were added to the IG_Leads_MSTR table and the IG_Leads_List_TEMP table. The Approved Salesperson ID field was also added to both of these tables. This is a new table field that provides the storage for the Salesperson ID that was approved when the lead was processed through the new workflow.

Table security

Add tables to the workflow role.

The workflow engine in Office SharePoint Server must be able to access the workflow information in the tables for your integration. The database role named DYNWORKFLOWGRP is defined for databases in Microsoft Dynamics GP to allow the workflow engine to access tables. You must add any tables for your integration that contain workflow information to this database role. Typically, you will do this using the same technique you use to add tables to DYNGRP role for your integration.

For example, the sample workflow integration adds workflow data to the IG_Leads_MSTR table, so this table must be added to the DYNWORKFLOWGRP role. The IG_Setup_SQL_Tables procedure in the sample integration was modified to add the table and auto-generated stored procedures to this role. The following is the code that performs these steps:

```
local boolean result;

{Assign the IG_Leads_MSTR table to the DYNWORKFLOWGRP}
result = GrantAccess(physicalname(table IG_Leads_MSTR), false,
➤ "DYNWORKFLOWGRP", 'Intercompany ID' of globals) of form 'SQL Maintenance';
result = GrantAccess(physicalname(table IG_Leads_MSTR), true,
➤ "DYNWORKFLOWGRP", 'Intercompany ID' of globals) of form 'SQL Maintenance';
```

Command form

Update the command form to retrieve and store the new workflow status.

The command form for an integration is opened immediately after the user logs into Microsoft Dynamics GP. It remains open the entire time the application is running. These characteristics make it the ideal place to retrieve and store the status of the new workflow you are creating. Each Microsoft Dynamics GP client must call out to the Workflow web service to determine whether the specific workflow is enabled. The result of this call must be stored locally so that forms and lists can use the workflow status information without having to request it repeatedly.

WorkflowStatus window

The initial workflow operations from the Microsoft Dynamics GP client are all initiated using the form factory functionality provided by the Dexterity runtime. This means that a window must be displayed to cause the form factory code to be run. Each integrating application should implement its own modal dialog window that will use the form factory to call the Workflow web service and determine the status of the specific workflow.

The command form is a good place to implement this modal dialog window. The form is always available, and the form can be used to store the results of the web service call. A window named WorkflowStatus was added to the Command_IG_Sample form for the sample integration. This is a modal window, with the AutoOpen property set to false.

Sample Integrating Application

Retrieving status of Lead Approval workflow

The following code was added to the form pre script for the Command_IG_Sample form to open the WorkflowStatus window and allow the form factory for this window to be run. The force redraw is required to display the static text in the window. After five seconds the window is closed and the login process proceeds as normal.

```
local long delay;

open window WorkflowStatus;
Window_ForceRedraw(window WorkflowStatus);

{Sleep for 5 seconds}
delay = Timer_Sleep(5000);

{Close the window}
close window WorkflowStatus;
```

Workflow Enabled checkbox

The enabled or disabled status of the Lead Approval workflow must be stored by the application. The Workflow Enabled global field is a checkbox that can be used for this purpose. For the sample workflow implementation it was added to the Dummy window for the Command_IG_Sample form. The workflow controller code that runs in response to the form factory for the WorkflowStatus modal dialog window will set the value of this checkbox field based on whether the Lead Approval workflow is enabled.

Retrieving the workflow status needs to be performed only once when the user logs in. Other areas of the integrations like maintenance windows and lists should use the value of the Workflow Enabled checkbox to determine whether to display workflow functionality. For example, the following script that is part of the IG_Lead_Maintenance window checks the value of this field to find out whether to configure the window for workflow.

```
if 'Workflow Enabled' of window Dummy of form Command_IG_Sample = true then
    unlock '(L) Submit';
    enable '(L) Submit';
else
    {Workflow is not implemented or disabled}
    lock '(L) Submit';
    disable '(L) Submit';
end if;
```

Window integration

Several changes are required to implement workflow functionality for a Dexterity-based window. These changes made with Dexterity work together with the client workflow controller to provide the workflow functionality. This section explains the changes that must be made with Dexterity and provides an example for the Lead Maintenance form in the sample integration.

Several changes are required to implement workflow for a maintenance window.

The screenshot shows a window titled "Lead Maintenance" with a status bar indicating "Pending Approval by Bruce R. due 7/20/2017". The window contains a form with the following fields:

- Lead ID: 1005
- Name: Fourth Coffee
- Category: Restaurant
- Potential Revenue: \$40,000.00
- Contact: Tom
- Address 1: 7655 Elm Street
- Address 2:
- City: Chicago
- State: IL
- Zip: 68234
- Phone 1: (412) 555-0119 Ext. 0000
- Phone 2: (000) 000-0000 Ext. 0000
- Fax: (410) 555-0199 Ext. 0000
- Salesperson: ERIN J.
- Approved Salesperson:
- Qualified: No
- Qualification Date: 0/0/0000
- Source:

At the bottom of the window, there is a navigation bar with "By Lead ID" selected. On the right side, there is a "Workflow History" panel showing a user "Steve K." with a date "2/24/2017" and a message "Please approve this lead."

Add hidden window fields needed for workflow.

Hidden fields

Several hidden fields are required to support workflow. The following is the list of fields to be added:

WFGGetInfo This field is global boolean field that signals the workflow controller to retrieve workflow information for the current record. The **run script** statement is used to cause the "validate" action to be performed by the workflow controller.

Workflow Approval Status This is a field from the main table for the item. It stores the current workflow approval state for the record.

Workflow Priority This is a field from the main table for the item. It stores the current workflow priority for the record.

ConfigureWindowForWorkflow This is a local integer field that has an attached script to configure the window for use with workflow. The following is the script for this field in the Lead Maintenance window of the sample workflow integration:

```
if 'Workflow Enabled' of window Dummy of form Command_IG_Sample = true then
    unlock '(L) Submit';
    enable '(L) Submit';
else
    {Workflow is not implemented or disabled}
    lock '(L) Submit';
    disable '(L) Submit';
end if;
```

RestartForm This is a local string field that has an attached script to perform a **restart form** statement for the form. The script is run by the workflow controller after the Save and Submit button is used to submit the current lead to the Lead Approval workflow.

Revise business logic in your application to support the workflow.

Business logic

It's likely that the business logic for the Dexterity-based window will need to be enhanced to support workflow functionality. The specific changes will depend on how you decide the window will interact with the workflow features. To get ideas about how the business logic for a window may change, examine the changes that were made to the business logic of the Lead Maintenance window to support the Lead Approval workflow. These changes include:

- Configuring the window based on whether workflow is enabled or disabled. The configuration code is centralized in the change script for the `ConfigureWindowForWorkflow` hidden field. This script is called from the window pre script and the window activate script.
- Adding the Save and Submit button. This button is enabled only when the Lead Approval workflow is active. When the user changes the Salesperson ID for a lead, this button allows the user to save the current record and submit the change to the workflow.
- Revised the code for the Save button. If the user is creating a new record or has changed the Salesperson ID assigned to an existing record, they will be prompted to use the Save and Submit button rather than the Save button. This will submit the lead to the approval workflow.
- Revised the code for the Delete button. When workflow information is attached to a record, the record should not be able to be deleted.

Workflow wrapper

The workflow wrapper which surrounds the maintenance window must be kept synchronized with the record being displayed in the window. This is done by running the change script on the `WFGetInfo` hidden field on the window. Care is required so the window and wrapper work together properly. Areas to keep in mind are:

- The workflow wrapper must be updated each time a different record is being displayed in the window. This requires changes in the code for any field that can cause a different record to be displayed, such as the control field or the browse buttons for the window. In the sample workflow integration, this block of `sanScript` code is used in the browse buttons and control field to keep the wrapper synchronized with the window.

```
{Display the workflow status information}
if 'Workflow Enabled' of window Dummy of form Command_IG_Sample = true
then
    run script WFGetInfo;
end if;
```

- The workflow wrapper must be reset when the window is cleared, such as after the user clicks Clear, Save, or Save and Submit.

Add code to keep the workflow wrapper synchronized with the window content.

Add form-level procedures needed to support workflow.

Form-level procedures

A form-level procedure must be added that can be called by the workflow controller to update the priority for the current record. In the sample workflow integration this procedure is named UpdateWorkflowPriority. The following is the code in this procedure.

```
in string Lead_ID;
in integer Workflow_Priority;

'Lead ID' of table IG_Leads_MSTR = Lead_ID;
change table IG_Leads_MSTR;
if err() = OKAY then
    {Set the workflow priority}
    'Workflow Priority' of table IG_Leads_MSTR = Workflow_Priority;

    {Save the updated record}
    save table IG_Leads_MSTR;
end if;
```

Another form-level procedure is needed to open the maintenance window from a workflow notification. This form-level procedure must have parameters that specify which document to display when the window is opened. For the sample workflow integration, this procedure is named OpenWindow, and is included in the IG_Lead_Maintenance form. The following is the code in this procedure:

```
in string Lead_ID;

if isopen(form IG_Lead_Maintenance) = false then
    open form IG_Lead_Maintenance;
    'Lead ID' of window 'Lead Maintenance' of form IG_Lead_Maintenance =
    ➤ Lead_ID;
    run script 'Lead ID' of window 'Lead Maintenance' of form
    ➤ IG_Lead_Maintenance;
else
    {Is an existing record being displayed?}
    if 'Display Existing Record' of window 'Lead Maintenance' = false then
        open window 'Lead Maintenance';
        'Lead ID' of window 'Lead Maintenance' of form IG_Lead_Maintenance =
        ➤ Lead_ID;
        run script 'Lead ID' of window 'Lead Maintenance' of form
        ➤ IG_Lead_Maintenance;
    end if;
end if;
```

Notifications

Enable client notifications for the new workflow.

Workflow can be configured to display notifications of workflow events within the Microsoft Dynamics GP client. Notifications are the “toast” items that appear in the lower-right corner of the application, notifying the user of a workflow event or task they must perform. You will want these notifications to work for your workflow integration as well.

A procedure trigger must be registered for the Get3rdPartyWorkflowAlertsInfo of the syWorkflowWebservice form to allow your integration to be informed of notification that should be displayed. The following is the trigger registration for the sample workflow integration.

```

l_result = Trigger_RegisterProcedure(script Get3rdPartyWorkflowAlertsInfo of
  ➤ form syWorkflowWebservice, TRIGGER_AFTER_ORIGINAL, script
  ➤ Workflow_ProcessDesktopNotification);
if l_result <> SY_NOERR then
  warning "Procedure trigger registration for workflow notifications
  ➤ failed.";
end if;

```

The trigger processing procedure must have the following parameters:

```

in reference WS_Node;
in string WS_WFBOType;
in string WS_WFNotificationType;
out boolean fHighPriority;
out boolean bShowAlert;
out string sDexLink;

```

Because several third-party workflows will register triggers for the procedure `Get3rdPartyWorkflowAlertsInfo`, your trigger processing procedure must find out whether the specific third-party workflow notification is for your workflow. Use the string passed in to the `WS_WFBOType` parameter to decide whether this notification is for your workflow. If it's not, the script should perform no action.

The following is the `Workflow_ProcessDesktopNotification` procedure for the sample workflow integration. It is the trigger processing procedure that handles the notifications for the sample workflow.

```

in reference WS_Node;
in string WS_WFBOType;
in string WS_WFNotificationType;
out boolean fHighPriority;
out boolean bShowAlert;
out string sDexLink;

local reference WS_ChildNode;
local reference WFBOKey, WFNode, KeyNode;
local string WS_WFSubject, WS_WFDescription, WS_WFComments, WS_GUIDKey;
local integer length;
local integer i;
local string sName;
local string Lead_ID;

{Is this our object?}
if WS_WFBOType = "Sales Lead" then

  {Retrieve the details from the XML node for the document being processed}

  {Extract the key information for the document}
  try
    WFBOKey = WS_Node.selectNodes(XPATH_BUSINESSOBJECTKEYQUERY of form
      ➤ syWorkflowWebservice);
  else
    {Consume all errors}
  end try;

```

```

{Look through each part of the key}
length = WFBOKey.length;
for i = 1 to length do
    WFNode = WFBOKey.nextNode();
    KeyNode = WFNode.selectSingleNode(NODE_NAME of form
    ➤ syWorkflowWebservice);
    sName = KeyNode.text;

    if sName = "LeadID" then
        KeyNode = WFNode.selectSingleNode(NODE_PARTVALUE of form
        ➤ syWorkflowWebservice);
        Lead_ID = KeyNode.text;
    end if;
end for;

WS_ChildNode = WS_Node.selectSingleNode(NODE_SUBJECT of form
➤ syWorkflowWebservice);
if not empty(WS_ChildNode) then
    WS_WFSubject = WS_ChildNode.text;
end if;
WS_ChildNode = WS_Node.selectSingleNode(NODE_DESCRIPTION of form
➤ syWorkflowWebservice);
if not empty(WS_ChildNode) then
    WS_WFDescription = WS_ChildNode.text;
end if;
WS_ChildNode = WS_Node.selectSingleNode(NODE_COMMENTS of form
➤ syWorkflowWebservice);
if not empty(WS_ChildNode) then
    WS_WFComments = WS_ChildNode.text;
end if;

WS_ChildNode = WS_Node.selectSingleNode(NODE_KEY_ID of form
➤ syWorkflowWebservice);
if not empty(WS_ChildNode) then
    WS_GUIDKey = WS_ChildNode.text;
end if;

{Based on the type of notification, build the DexLink string}
{Tasks can be TEXT_INFORMATIONAL or TEXT_TASK}
if WS_WFNotificationType = TEXT_TASK of form syWorkflowWebservice then
    sDexLink = HREF_DEXTERITY of form syHomePageXML + HREF_PRODUCT of form
    ➤ syHomePageXML + str(IG_PROD_ID) + HREF_SCRIPT of form syHomePageXML
    ➤ + "OpenWindow" + HREF_FORM of form syHomePageXML +
    ➤ technicalname(form IG_Lead_Maintenance) + HREF_ARGS of form
    ➤ syHomePageXML + "'" + Lead_ID + "'";
end if;

bShowAlert = true;
end if;

```

Notice that the procedure must extract information about the notification from the WS_Node (an XML document) passed into the procedure. The key value or values for the notification, as well as the subject, description, comments and node key GUID are all retrieved from this XML document.

Once the information about the notification has been retrieved, the procedure must determine which type of notification is being displayed. The `WS_WFNotificationType` parameter is set to either the `TEXT_INFORMATIONAL` or `TEXT_TASK` constant defined in the `syWorkflowWebservice` form. The action performed when the user clicks the “toast” that is displayed for the notification depends on the notification type. The `sDexLink` string specifies the action. If it’s an informational notification, the core Dynamics GP code will build the `sDexLink` string to display a standard workflow notification dialog.

If the notification is a task notification, you must build the `sDexLink` string to display the window that allows the user to perform the requested action. Typically, this will be the window for which you are implementing workflow. The syntax used for the `sDexLink` string is described in the Dexterity help file, in the topic for the `Shell_DisplayNotification()` function.

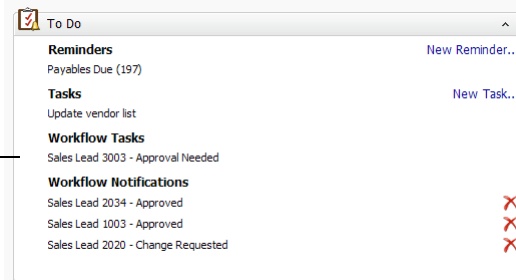
Finally, the procedure sets the `bShowAlert` parameter to true. This tells the core Microsoft Dynamics GP code to display the notification. If this isn’t your notification, *do not* set this parameter value to false. You may prevent another integrating application’s notifications from being displayed.

Home Page integration

Add Home Page support for the new workflow.

The Home Page for a user can contain the To Do section listing the actions the user needs to perform. The user can configure the To Do section to contain workflow tasks and workflow notifications. Your new workflow should add the appropriate tasks to this section of the Home Page.

Tasks and notifications for your new workflow should appear in the To Do section of the Home Page.



A procedure trigger must be registered for the `Get3rdPartyWorkflowTaskInfo` of the `syHomePageXML` form to allow your integration to be informed when tasks should be added to the To Do list on the home page. The following is the trigger registration for the sample workflow integration.

```
l_result = Trigger_RegisterProcedure(script Get3rdPartyWorkflowTaskInfo of
  form syHomePageXML, TRIGGER_AFTER_ORIGINAL, script
  Workflow_AddTasksToHomePage);
if l_result <> SY_NOERR then
  warning "Procedure trigger registration for adding To Do items failed.";
end if;
```

The trigger processing procedure must have the following parameters:

```
in reference WS_Node;
in string WS_BOType;
inout boolean fHighPriority;
inout boolean bAddTask;
inout string sDexLink;
```

Because several third-party workflows will register triggers for the procedure `Get3rdPartyWorkflowTaskInfo`, your trigger processing procedure must find out whether the specific third-party workflow task is for your workflow. Use the string passed in to the `WS_BOType` parameter to decide whether this task is for your workflow. If it's not, the script should perform no action.

The following is the `Workflow_AddTasksToHomePage` procedure for the sample workflow integration. It is the trigger processing procedure that handles adding tasks for the sample workflow.

```

in reference WS_Node;
in string WS_BOType;
inout boolean fHighPriority;
inout boolean bAddTask;
inout string sDexLink;

local reference WFBOKey, WFNode, KeyNode;
local integer length;
local integer i;
local string sName;
local string Lead_ID;

{Is this our object?}
if WS_BOType = "Sales Lead" then
    {Extract the key information for the document}
    try
        WFBOKey = WS_Node.selectNodes(XPATH_BUSINESSOBJECTKEYQUERY of form
            ➤ syWorkflowWebservice);
    else
        {Consume all errors}
    end try;

    {Look through each part of the key}
    length = WFBOKey.length;
    for i = 1 to length do
        WFNode = WFBOKey.nextNode();
        KeyNode = WFNode.selectSingleNode(NODE_NAME of form
            ➤ syWorkflowWebservice);
        sName = KeyNode.text;

        if sName = "LeadID" then
            KeyNode = WFNode.selectSingleNode(NODE_PARTVALUE of form
                ➤ syWorkflowWebservice);
            Lead_ID = KeyNode.text;
        end if;
    end for;

    {Build up the link for the task}
    sDexLink = HREF_DEXTERITY of form syHomePageXML + HREF_PRODUCT of form
        ➤ syHomePageXML + str(IG_PROD_ID) +
        HREF_SCRIPT of form syHomePageXML + "OpenWindow" + HREF_FORM of form
        ➤ syHomePageXML + technicalname(form IG_Lead_Maintenance) +
        ➤ HREF_ARGS of form syHomePageXML + "'" + Lead_ID + "'";

    {Indicate that the task should be added to the list}
    bAddTask = true;
end if;

```

Notice that the procedure must extract information about the workflow task from the WS_Node (an XML document) passed into the procedure. The key value or values for the document associated with the task are retrieved from this XML document.

You must build the sDexLink string to display the window that allows the user to perform the workflow task. Typically, this will be the window for which you are implementing workflow. The syntax used for the sDexLink string is described in the Dexterity help file, in the topic for the **Shell_DisplayNotification()** function.

Finally, the procedure sets the bAddTask parameter to true. This tells the core Microsoft Dynamics GP code to add the task. If this isn't your task, *do not* set this parameter value to false. You may prevent another integrating application's tasks from being added to the list.

List integration

Numerous changes are needed to support workflow functionality from list in Microsoft Dynamics GP. These changes made with Dexterity work together with the client workflow controller to provide the workflow functionality for the list. This section explains the changes that must be made with Dexterity and provides an example for the Leads list in the sample integration.

Several additions and changes are needed to support workflow for a list.

Lead Name	Workflow Status	Lead ID	Address 1	Category	Potential Revenue	Approved Salesperson ID
Baldwin Museum of Science	Submitted	1001	123 Wayside Drive	Educational	\$25,000.00	
Fourth Coffee	Pending Approval	1005	7655 Elm Street	Restaurant	\$40,000.00	
Northwind Traders	Approved	1002	789 Carson Boulevard	Retail	\$75,000.00	ERIN J.
Proseware, Inc.	Approved	1003	123 Longhorn Road	Software	\$25,000.00	GARY W.
Southridge Video	Approved	1004	4567 Union Street	Retail	\$35,000.00	FRANCINE B.
Wide World Importers	Submitted	1006	9876 Waterfront Drive	Wholesale	\$95,000.00	
Woodgrove Bank	Not Submitted	1007	1234 25th Street	Retail	\$55,000.00	

1005 : Fourth Coffee			
Address :	7655 Elm Street	Category :	Restaurant
	Chicago, IL 68234	Contact :	Tom
Phone 1 :	(412) 555-0119 Ext. 0000	Potential Revenue :	\$40,000.00
Phone 2 :	(000) 000-0000 Ext. 0000	Qualified :	No
Fax :	(410) 555-0199 Ext. 0000	Qualification Date :	0/0/0000
		Source :	

Define constants for the workflow actions and dialog callback in the list.

Constants

Several constants must be added to the list object form when implementing workflow. The first set of constants defines the workflow actions that can be performed for the list items:

Constant	Value	Description
ACTION_WF_APPROVE	100	The Approve action
ACTION_WF_SUBMIT	101	The Submit action
ACTION_WF_REQUESTCHANGE	102	The Request Change action
ACTION_WF_REJECT	103	The Reject action
ACTION_WF_DELEGATE	104	The Delegate action

Another constant must be added to the list object form that specifies the fully-qualified name of the form-level procedure to be called when the drop-dialog for the workflow action has been closed. This constant must have the name WFDIALOG_CALLBACK_PROC. The value of the constant must contain the fully-qualified procedure name. For the Leads list in the workflow integration sample the value following values are used:

Constant Name: **WFDIALOG_CALLBACK_PROC**
Value: **WorkFlowDialogsReturn of form ListObj_Leads**

Hidden fields

Add hidden fields needed by the list to support workflow.

Several hidden fields must be added to the State window for the list object. These hidden fields are used to trigger processing in the workflow controller. They also store information used by workflow as it processes items in the list. The following is the list of fields to be added:

WFReturn This is a local integer field. It has the following change script attached, which starts the processing of the workflow action the user has chosen.

```
local integer nAction;

{Retrieve the values from the drop-dialog window}
call GetReturnValues of form WorkflowDialogs,
    nAction,
    '(L) ChangeDaysToAct',
    '(L) DaysToAct',
    'Workflow Priority',
    '(L) DelegateUser',
    '(L) Comment';

{Important -- Notice that the constants for the action retrieved from the
workflow dialog are not the same as the constants for the commands for the
workflow actions.}
case nAction
    in[WF_ACTION_APPROVE]
        call ExecuteListAction of form syCardList, ACTION_WF_APPROVE of form
            ListObj_Leads, true, getmsg(11332); {Approve}
    in[WF_ACTION_DELEGATE]
        call ExecuteListAction of form syCardList, ACTION_WF_DELEGATE of form
            ListObj_Leads, true, getmsg(11378); {Delegate}
    in[WF_ACTION_REJECT]
        call ExecuteListAction of form syCardList, ACTION_WF_REJECT of form
            ListObj_Leads, true, getmsg(11379); {Reject}
```

```

in[WF_ACTION_REQUESTCHANGE]
    call ExecuteListAction of form syCardList, ACTION_WF_REQUESTCHANGE of
    ➤ form ListObj_Leads, true, getmsg(11381); {Request Change}
in[WF_ACTION_SUBMIT]
    call ExecuteListAction of form syCardList, ACTION_WF_SUBMIT of form
    ➤ ListObj_Leads, true, getmsg(10496); {Submit}
end case;

```

ChangeDaysToAct This is a local checkbox field. It is used to change the number of days allowed for a workflow task.

DaysToAct This is a local integer field. It specifies the new number of days to act on a workflow task.

Workflow Priority This global field tracks the priority for the current workflow item being processed.

DelegateUser This is a local string field based on the global datatype STR255. It specifies the user to whom a workflow task is being delegated.

Comment This is a local text field with a keyable length of 1500. It stores the comment supplied for the workflow action.

wfResetWorkflowHandler This is a global boolean field.

WFTransferInitiate This is a local integer field used to start the process of adding workflow items to be processed.

WFTransfer This is a local boolean field used to transfer a single workflow item to be processed.

WFTransferComplete This is a local integer field used to indicate that the process of adding workflow items to be processed is complete.

WFVerifySuccess This is a local boolean field.

ActionStatusID This is a global string field.

DateString This is a local string with keyable length 20. It stores a date value used in internal processing for workflow.

TimeString This is a local string with keyable length 20. It stores a time value used in internal processing for workflow.

Attached tables

The following tables must be attached to the list object form:

- syListActionStatusHdr
- syListActionStatusLine

These tables provide access to the message bar status information that must be updated when an error occurs as a workflow item is processed for the list.

Attach tables needed for the workflow.

Defined the commands for the workflow actions.

Action Pane

A new group containing workflow actions must be added to the Action Pane for your list. This group is typically named “Approval” and contains the various workflow actions. The following table lists the commands you will need to create for the workflow actions:

Command	Names	Type	Image	Button Type
CL_ApprovalGroup	Display Name: Approval Tooltip: Approval	Command List	Type: Icon Normal Image: Workflow	N/A
Approve	Display Name: Approve Tooltip: Approve	Script	Type: Icon Normal Image: Approve	Drop Dialog
Delegate	Display Name: Delegate Tooltip: Delegate	Script	Type: Icon Normal Image: Delegate	Drop Dialog
Reject	Display Name: Reject Tooltip: Reject	Script	Type: Icon Normal Image: Reject	Drop Dialog
RequestChange	Display Name: Request Change Tooltip: Request Change	Script	Type: Icon Normal Image: ChangeRequest	Drop Dialog
Submit	Display Name: Submit Tooltip: Submit	Script	Type: Icon Normal Image: Submit	Drop Dialog

The commands that are of the type Script will each have a script attached that performs the specified action. The following is an example of this script from the Approve command in the sample integration:

```
call OpenWindowFromList of form WorkflowDialogs,
    WF_ACTION_APPROVE,
    IG_PROD_ID,
    WFDIALOG_CALLBACK_PROC;
```

Notice that it passes a constant for the workflow action to approve, the product ID for the product that is implementing the workflow, and the constant for the drop-dialog callback procedure.

The constant for the workflow action to perform corresponds to one of the following constants:

Constant	Description
WF_ACTION_APPROVE	The Approve action
WF_ACTION_DELEGATE	The Delegate action
WF_ACTION_REJECT	The Reject action
WF_ACTION_REQUESTCHANGE	The Request Change action
WF_ACTION_SUBMIT	The Submit action



These constants are not the same as those that you created for your list actions. Later, you will re-map the workflow actions for your list to these constants.

Add the workflow actions to the Action Pane.

The commands for the workflow actions must be added to the Action Pane. This is done in the `CreateListRibbonData` procedure for the list. The following is a portion of this procedure from the Leads list in the sample workflow integration. It adds the Approval group and the workflow items for it.

```
{Approval group -- for workflow}
increment nGroupSeq;
nCmdID = resourceid(command CL_ApprovalGroup);
nStatus = AddGroup(nListDictID, nListID, LIST_PRIMARYVIEWID, nGroupSeq,
    nCmdDictID, nCmdFormID, nCmdID,
    "{caption}",
    true{visible}) of form syListViewCmdBarObj;

{--- add default commands to the Approval group ---}
nSeq = 0;

nParentDictID = nCmdDictID;
nParentFormID = nCmdFormID;
nParentCmdID = nCmdID;

nCmdDictID = IG_PROD_ID;
nCmdFormID = resourceid(form ListObj_Leads);

{ Submit }
increment nSeq;
nStatus = AddCommand(nListDictID, nListID, LIST_PRIMARYVIEWID,
    nParentDictID, nParentFormID, nParentCmdID,
    nSeq,
    nCmdDictID, nCmdFormID, resourceid(command Submit),
    LISTACTIONPRIORITY_PRIMARY,
    LISTACTIONBTNSIZE_LARGE,
    "{caption}",
    true{visible}) of form syListViewCmdBarObj;

{ Approve }
increment nSeq;
nStatus = AddCommand(nListDictID, nListID, LIST_PRIMARYVIEWID,
    nParentDictID, nParentFormID, nParentCmdID,
    nSeq,
    nCmdDictID, nCmdFormID, resourceid(command Approve),
    LISTACTIONPRIORITY_SECONDARY,
    LISTACTIONBTNSIZE_SMALL,
    "{caption}",
    true{visible}) of form syListViewCmdBarObj;

{ Reject }
increment nSeq;
nStatus = AddCommand(nListDictID, nListID, LIST_PRIMARYVIEWID,
    nParentDictID, nParentFormID, nParentCmdID,
    nSeq,
    nCmdDictID, nCmdFormID, resourceid(command Reject),
    LISTACTIONPRIORITY_SECONDARY,
    LISTACTIONBTNSIZE_SMALL,
    "{caption}",
    true{visible}) of form syListViewCmdBarObj;
```

```

{ Request Change }
increment nSeq;
nStatus = AddCommand(nListDictID, nListID, LIST_PRIMARYVIEWID,
    nParentDictID, nParentFormID, nParentCmdID,
    nSeq,
    nCmdDictID, nCmdFormID, resourceid(command RequestChange),
    LISTACTIONPRIORITY_SECONDARY,
    LISTACTIONBTNSIZE_SMALL,
    "{caption}",
    true{visible}) of form syListViewCmdBarObj;

{ Delegate }
increment nSeq;
nStatus = AddCommand(nListDictID, nListID, LIST_PRIMARYVIEWID,
    nParentDictID, nParentFormID, nParentCmdID,
    nSeq,
    nCmdDictID, nCmdFormID, resourceid(command Delegate),
    LISTACTIONPRIORITY_SECONDARY,
    LISTACTIONBTNSIZE_SMALL,
    "{caption}",
    true{visible}) of form syListViewCmdBarObj;

```

Register the workflow actions for the Action Pane.

The new commands you create must be registered for the Action Pane. This is done in the RegisterCommands procedure for the list. The following is a portion of the RegisterCommands procedure for the Leads list in the sample integration. It registers the command list and commands for the workflow integration.

```

{ Approvals }
List_RegisterGroup(list_object, command CL_ApprovalGroup) of form syListObj;
List_RegisterAction(list_object, command Submit, LISTCMDTYPE_MULTISELECT,
    ➤ ACTION_WF_SUBMIT) of form syListObj;
List_RegisterAction(list_object, command Approve, LISTCMDTYPE_MULTISELECT,
    ➤ ACTION_WF_APPROVE) of form syListObj;
List_RegisterAction(list_object, command Reject, LISTCMDTYPE_MULTISELECT,
    ➤ ACTION_WF_REJECT) of form syListObj;
List_RegisterAction(list_object, command RequestChange,
    ➤ LISTCMDTYPE_MULTISELECT, ACTION_WF_REQUESTCHANGE) of form syListObj;
List_RegisterAction(list_object, command Delegate, LISTCMDTYPE_MULTISELECT,
    ➤ ACTION_WF_DELEGATE) of form syListObj;

```

Define the access rules for the new workflow actions.

The access for the new commands must also be updated in the CheckActionAccessForRecord procedure for the list. The following is the updated version of this procedure for the Leads list in the sample workflow integration. The script contains entries for the workflow actions to control the conditions under which they are enabled or disabled.

```

in ListObjState list_object;
in integer nActionCmdTag;
out integer nAccessStatus;

nAccessStatus = LISTACTIONACCESS_COUNT_NEUTRAL;

if nActionCmdTag = Command_GetTag(command DeleteLead) then
    nAccessStatus = LISTACTIONACCESS_COUNT_FOR;
elseif nActionCmdTag = Command_GetTag(command EditLead) then
    nAccessStatus = LISTACTIONACCESS_COUNT_FOR;

```



```

elseif nActionCmdTag = Command_GetTag(command ViewLead) then
    nAccessStatus = LISTACTIONACCESS_COUNT_FOR;
elseif nActionCmdTag = Command_GetTag(command QualifyLead) then
    {Examine whether the lead is already qualified}
    if 'Qualified Lead' of table (list_object:'Table Reference') = 1 then
        {Lead has not been qualified}
        nAccessStatus = LISTACTIONACCESS_COUNT_FOR;
    else
        {Lead has already been qualified, so disable that action}
        nAccessStatus = LISTACTIONACCESS_COUNT_AGAINST;
    end if;
elseif nActionCmdTag = Command_GetTag(command Submit) then
    {Check for the 'not submitted' state, and that the Salesperson ID and
    Approved Salesperson ID don't match}
    if 'Workflow Approval Status' of table(list_object:'Table Reference') =
    ➤ WF_APPROVAL_STATUS_NOTSUBMITTED then
        {Lead has not been submitted.}
        nAccessStatus = LISTACTIONACCESS_COUNT_FOR;
    else
        {Lead is in some other workflow state, so it shouldn't be submitted.}
        nAccessStatus = LISTACTIONACCESS_COUNT_AGAINST;
    end if;

    if 'Salesperson ID' of table(list_object:'Table Reference') =
    ➤ 'Approved Salesperson ID' of table(list_object:'Table Reference') then
        {Salesperson ID and Approved Salesperson ID match, so no need to
        approve.}
        nAccessStatus = LISTACTIONACCESS_COUNT_AGAINST;
    end if;
elseif nActionCmdTag = Command_GetTag(command Approve) then
    {Check for the Pending Approval state}
    if 'Workflow Approval Status' of table(list_object:'Table Reference') =
    ➤ WF_APPROVAL_STATUS_PENDINGAPPROVAL then
        {Lead is waiting for approval.}
        nAccessStatus = LISTACTIONACCESS_COUNT_FOR;
    else
        {Some other state, so count against}
        nAccessStatus = LISTACTIONACCESS_COUNT_AGAINST;
    end if;
elseif nActionCmdTag = Command_GetTag(command Reject) then
    {Check for the Pending Approval state}
    if 'Workflow Approval Status' of table(list_object:'Table Reference') =
    ➤ WF_APPROVAL_STATUS_PENDINGAPPROVAL then
        {Lead is waiting for approval.}
        nAccessStatus = LISTACTIONACCESS_COUNT_FOR;
    else
        {Some other state, so count against}
        nAccessStatus = LISTACTIONACCESS_COUNT_AGAINST;
    end if;
elseif nActionCmdTag = Command_GetTag(command RequestChange) then
    {Check for the Pending Approval state}
    if 'Workflow Approval Status' of table(list_object:'Table Reference') =
    ➤ WF_APPROVAL_STATUS_PENDINGAPPROVAL then
        {Lead is waiting for approval.}
        nAccessStatus = LISTACTIONACCESS_COUNT_FOR;
    else

```

```

        {Some other state, so count against}
        nAccessStatus = LISTACTIONACCESS_COUNT_AGAINST;
    end if;
elseif nActionCmdTag = Command_GetTag(command Delegate) then
    {Check for the Pending Approval state}
    if 'Workflow Approval Status' of table(list_object:'Table Reference') =
        ➤ WF_APPROVAL_STATUS_PENDINGAPPROVAL then
        {Lead is waiting for approval.}
        nAccessStatus = LISTACTIONACCESS_COUNT_FOR;
    else
        {Some other state, so count against}
        nAccessStatus = LISTACTIONACCESS_COUNT_AGAINST;
    end if;
end if;
end if;

```

Hide or show the actions, based on whether the workflow is enabled.

The commands and command list for the workflow actions should be visible in the Action Pane only when the workflow is enabled. The Initialize procedure for the list can be used to find out the state of the workflow, and then enable or disable the commands appropriately. The following is a portion of the Initialize procedure for the Leads list in the sample workflow integration. It queries the status of the Lead Approval workflow and enables or disables the workflow actions based on the result.

```

{Find whether the actions for workflow should be enabled}
if 'Workflow Enabled' of window Dummy of form Command_IG_Sample = false then
    hide command CL_ApprovalGroup;
    hide command Approve;
    hide command Submit;
    hide command RequestChange;
    hide command Reject;
    hide command Delegate;
    disable command CL_ApprovalGroup;
    disable command Approve;
    disable command Submit;
    disable command RequestChange;
    disable command Reject;
    disable command Delegate;
else
    show command CL_ApprovalGroup;
    show command Approve;
    show command Submit;
    show command RequestChange;
    show command Reject;
    show command Delegate;
    enable command CL_ApprovalGroup;
    enable command Approve;
    enable command Submit;
    enable command RequestChange;
    enable command Reject;
    enable command Delegate;
end if;

```

Add code to perform the workflow actions.

To process the new workflow actions, the ExecuteAction procedure must be updated. In the Leads list for the sample workflow integration the following case was added to the ExecuteAction procedure:

```
in [ACTION_WF_APPROVE of form ListObj_Leads, ACTION_WF_SUBMIT of form
ListObj_Leads, ACTION_WF_REQUESTCHANGE of form ListObj_Leads,
ACTION_WF_REJECT of form ListObj_Leads, ACTION_WF_DELEGATE of form
ListObj_Leads]
    call WFActionForList of form ListObj_Leads, list_object, nAction;
```

This code in the ExecuteAction procedure calls the new WFActionForList procedure that actually starts the process of performing a workflow action for a list. This new procedure is shown below. The procedure processes each marked item, updating its state and keeping track of any errors that occur. The workflow actions are processed as a group by the workflow engine in SharePoint.

```
inout ListObjState list_object;
in long nAction;

local long nErrStatus;
local integer nSuccessCount = 0;
local integer nFailureCount = 0;

local string message;
local integer err_num;

local boolean first_row;
local Ord n;
local ListStatusLineUserDefData Data;

{Get the first row that is marked}
{get first table(list_object:TableRef1) by number list_object:Index;}
get first table(list_object:TableRef1) by number 1;
first_row = true;

while err() = OKAY do

    {Read the record for the lead--in case the data in the list is stale}
    'Lead ID' of table IG_Leads_MSTR = 'Lead ID' of
    ➤ table(list_object:TableRef1);
    change table IG_Leads_MSTR;

    {Clear the message and error number for the current record}
    clear message;
    clear err_num;

    case err()
        in [MISSING]
            err_num = err();
            message = "Record for " + 'Lead ID' of
            ➤ table(list_object:TableRef1) + " was deleted by another user";
        in [LOCKED]
            err_num = err();
            message = "Record for " + 'Lead ID' of
            ➤ table(list_object:TableRef1) +
            ➤ " was locked and could not be updated.";
        in [OKAY]
            {Check the workflow status of the document first to see if it
```

```

should be processed.)
case nAction
  in[ACTION_WF_APPROVE, ACTION_WF_DELEGATE, ACTION_WF_REJECT,
    ➤ ACTION_WF_REQUESTCHANGE]
    if 'Workflow Approval Status' of table IG_Leads_MSTR <>
      ➤ WF_APPROVAL_STATUS_PENDINGAPPROVAL then
        err_num = 1001;
        message = "You can't approve, delegate, reject, or
          ➤ request a change for this lead until it has a status
          ➤ of Pending Approval.";
      end if;
  in[ACTION_WF_SUBMIT]
    case 'Workflow Approval Status' of table IG_Leads_MSTR
      in[WF_APPROVAL_STATUS_NOTACTIVATED,
        WF_APPROVAL_STATUS_NOTSUBMITTED,
        WF_APPROVAL_STATUS_PENDINGCHANGES,
        WF_APPROVAL_STATUS_REJECTED,
        WF_APPROVAL_STATUS_WORKFLOWENDED,
        WF_APPROVAL_STATUS_DEACTIVATED]

        'Workflow Approval Status' of table IG_Leads_MSTR =
          ➤ WF_APPROVAL_STATUS_SUBMITTED;
      else
        err_num = 1002;
        message = "This lead has already been submitted.";
      end case;
    end case;

if err_num <> 0 then
  {An error was logged, so don't update the IG_Leads_MSTR table
  or submit to workflow}
  release table IG_Leads_MSTR;
else
  {No errors, so continue processing the current row}
  'Workflow Priority' of table IG_Leads_MSTR =
  ➤ 'Workflow Priority' of window State;
  save table IG_Leads_MSTR;

  ActionStatusID of window State = list_object:ActionStatusID;

  {These values will be used by the workflow list controller}
  'Lead ID' of window State = 'Lead ID' of
  ➤ table(list_object:TableRef1);

  '(L) DateString' of window State =
  ➤ str(GetActionStatusDate(list_object) of form syListObj);
  '(L) TimeString' of window State =
  ➤ str(GetActionStatusTime(list_object) of form syListObj);

if first_row = true then
  {Start the workflow processing}
  first_row = false;
  {Need to specify the workflow action to take using the
  Dynamics GP constants}
  {This case statement re-maps the list actions to those
  constants}

```

```

case nAction
  in [ACTION_WF_SUBMIT]
    '(L) WFTransferInitiate' of window State =
      ➔ integer(WF_ACTION_SUBMIT);
  in [ACTION_WF_APPROVE]
    '(L) WFTransferInitiate' of window State =
      ➔ integer(WF_ACTION_APPROVE);
  in [ACTION_WF_DELEGATE]
    '(L) WFTransferInitiate' of window State =
      ➔ integer(WF_ACTION_DELEGATE);
  in [ACTION_WF_REJECT]
    '(L) WFTransferInitiate' of window State =
      ➔ integer(WF_ACTION_REJECT);
  in [ACTION_WF_REQUESTCHANGE]
    '(L) WFTransferInitiate' of window State =
      ➔ integer(WF_ACTION_REQUESTCHANGE);
end case;
run script '(L) WFTransferInitiate' of window State;
end if;

{Run this script once for each successful update to tell the
workflow controller to collect this information it needs}
run script '(L) WFTransfer' of window State;
if '(L) WFTransfer' of window State = false then
  {You can't approve, delegate, reject, or request a
change for this document because you are not an
assigned approver.}
  err_num = 9496;
  message = getmsg(9496);
end if;
end if;
end case;

{ Log errors that were encountered }
if err_num <> 0 then
  increment nFailureCount;
  n = ActionStatus_LogError(list_object:ActionStatusID, n+1, message,
➔ err_num,
  'Lead ID' of table(list_object:'TableRef1'), Data) of form syListObj;
else
  increment nSuccessCount;
end if;

{Get the next row that is marked}
get next table(list_object:TableRef1) by number list_object:Index;

if err() = EOF then
  {Tell the workflow controller that processing is finished.}
  case nAction
    in [ACTION_WF_SUBMIT]
      '(L) WFTransferComplete' of window State =
        ➔ integer(WF_ACTION_SUBMIT);
    in [ACTION_WF_APPROVE]
      '(L) WFTransferComplete' of window State =
        ➔ integer(WF_ACTION_APPROVE);
    in [ACTION_WF_DELEGATE]

```

```

        '(L) WFTransferComplete' of window State =
        ➤ integer(WF_ACTION_DELEGATE);
    in [ACTION_WF_REJECT]
        '(L) WFTransferComplete' of window State =
        ➤ integer(WF_ACTION_REJECT);
    in [ACTION_WF_REQUESTCHANGE]
        '(L) WFTransferComplete' of window State =
        ➤ integer(WF_ACTION_REQUESTCHANGE);
    end case;
    run script '(L) WFTransferComplete' of window State;
end if;
end while;

{Indicate that the processing is complete}
nErrStatus = ActionStatus_LogActionComplete(list_object:ActionStatusID,
nSuccessCount, nFailureCount) of form syListObj;
call List_MultiSelectActionCompleteEvent of form syListObj, list_object;

```

Add additional columns to the list to display workflow information.

Columns

Some additional columns containing workflow information must be added to the list. These columns are added, but not visible by default. You may want to make an additional view for the list to show the workflow columns. The following columns were added for the Leads list in the sample workflow integration:

- Approved Salesperson ID
- Workflow Approval Status
- Workflow Priority

The following is a portion of the CreateListColumnData procedure for the Leads list in the sample workflow integration. It adds the columns needed for workflow to the list.

```

{--- Approved Salesperson ID ---}
increment nSeq;
nStatus = CreateDefaultColumn(
    nListDictID,
    nListID,
    nSeq,
    resourceid(field 'Approved Salesperson ID'),
    COLSORTORDER_NONE{sort order},
    false{visible},
    250{width},
    0,
    0, 0, { no format field }
    0, {token ID}
    0{sort seq}) of form syListViewColObj;

{--- Workflow Approval Status ---}
increment nSeq;
nStatus = CreateDefaultColumn(
    nListDictID,
    nListID,
    nSeq,
    resourceid(field 'Workflow Approval Status'),
    COLSORTORDER_NONE{sort order},
    false{visible},

```

```

250{width},
0,
0, 0, { no format field }
0, {token ID}
0{sort seq}) of form syListViewColObj;

{--- Workflow Priority ---}
increment nSeq;
nStatus = CreateDefaultColumn(
    nListDictID,
    nListID,
    nSeq,
    resourceid(field 'Workflow Priority'),
    COLSORTORDER_NONE{sort order},
    false{visible},
    250{width},
    0,
    0, 0, { no format field }
    0, {token ID}
    0{sort seq}) of form syListViewColObj;

```

Define the names for the new workflow columns.

The GetColumnName procedure for the list must also be updated to include the names of the columns used for workflow. The following is a portion of this procedure for the Leads list in the sample workflow integration. This portion of the case statement returns the names for the new columns added for workflow.

```

in [resourceid('Approved Salesperson ID' of table IG_Leads_MSTR)]
    set sColName to "Approved Salesperson ID";
in [resourceid('Workflow Approval Status' of table IG_Leads_MSTR)]
    set sColName to "Workflow Status";
in [resourceid('Workflow Priority' of table IG_Leads_MSTR)]
    set sColName to "Workflow Priority";

```

Update the Refresh procedure to display the workflow icon for list items.

The Icon column of the list must also be updated to indicate when workflow information is attached to the specific row. The Icon column is updated in the Refresh procedure for the list. Updating this column typically involves executing pass-through SQL to update the temporary table used for the list. The following is a portion of the Refresh procedure for the Leads list in the sample workflow integration. It uses the SQL statement shown to update the InfoValue column of the list to the appropriate value if workflow information is attached for that record.

```
{Update the status column for workflow}
```

```
{Basic SQL statement used:
```

```
-----
```

```

update LISTTEMP
    set InfoValue = InfoValue + INFO_VALUE_WORKFLOWPRESENT
    where Workflow_Approval_Status > 1
}

```

```

SQLCommand = "update " + ListTempTable
+ " set " + ListTempTable + "." + InfoValueField + " = " +
ListTempTable + "." + InfoValueField + " + " +
str(InfoValueWorkflowPresent) + " where " + ListTempTable + "." +
WorkflowApprovalStatusField + " > 1";

```

Add form-level procedures needed for the list to support workflow.

Form-level procedures

Several additional procedures must be added to the list object form to support workflow operations. The following procedures must be added:

WFRequestReturn This procedure is called when the workflow action being performed by the list encounters an error. The procedure finds the correct records that will be displayed in the message bar, and adds information for the error that occurred.

```

in ActionStatusID nActionStatusID;
in string LeadID;
in string sDate;
in string sTime;
in string sErr;

local syListActionStatusHdrState HdrObj;
local syListActionStatusLineState LineObj;
local integer nStatus;
local date dt;
local time tm;
local ListStatusLineUserDefData UserDefData;

if empty(nActionStatusID) then
    abort script;
end if;

nStatus = Create(HdrObj, LineObj, table syListActionStatusHdr, table
    ➤ syListActionStatusLine, nActionStatusID, MODE_CHG) of form
    ➤ syListActionStatusObj;
if nStatus = OKAY then

    {Validate the date and time to make sure we have the correct button press}
    call GetDateAndTime of form syListActionStatusObj, HdrObj, dt, tm;

    if str(dt) = sDate and str(tm) = sTime then

        {Decrement the success count, increment the failure count on the hdr
        and write a new line.}
        call SetSuccessCount of form syListActionStatusObj, HdrObj,
            ➤ (GetSuccessCount(HdrObj)of form syListActionStatusObj - 1);
        call SetFailureCount of form syListActionStatusObj,
            ➤ HdrObj, (GetFailureCount(HdrObj)of form syListActionStatusObj + 1);

        nStatus = CommitHeader(HdrObj) of form syListActionStatusObj;
        if nStatus = OKAY then
            {Add new line}
            call AddNewLine of form syListActionStatusObj,
                LineObj,
                nActionStatusID,
                0,
                sErr,
                0,
                LeadID,
                UserDefData;

            nStatus = CommitLine(LineObj) of form syListActionStatusObj;

```



```

        assert OKAY=nStatus;
    end if;
end if;
end if;

```

WorkflowDialogsReturn This procedure is called after the user has chosen a workflow action from the drop-dialog, and the drop-dialog has been closed. The procedure starts the processing for the items marked in the list.

```
run script '(L) WFReturn' of window State;
```

Application assembly

Create the application assembly that allows the client workflow controller to interact with the integrating dictionary.

After the dictionary changes have been made, you will create an application assembly that allows the client workflow controller to access these dictionary resources. Do this using the Dictionary Assembly Generator (DAG.exe) included with the Visual Studio Tools SDK. The following example shows the basic command used to create the application assembly for the sample workflow integration:

```
dag.exe 3333 "c:\Program Files\Microsoft Dynamics\GP\Dynamics.set" /M
```

This creates an assembly named `Application.SampleIntegratingApp.dll` that you will ship with your client workflow controller. It also creates an XML file that is named `Application.SampleIntegratingApp.xml`, which is used by IntelliSense in Visual Studio. You will use this XML file as you're developing your client workflow controller, but you won't ship the XML file to customers.



Refer to the Visual Studio Tools for Microsoft Dynamics GP Programmer's Guide for details about creating an application assembly.

Chapter 7: Client Workflow Assembly

The client workflow assembly is a Microsoft .NET assembly that enables Dexterity-based Microsoft Dynamics GP forms to interact with the Workflow server. Information about the client workflow assembly is described in the following sections:

- [Overview](#)
- [Creating a Visual Studio project](#)
- [Creating a BusinessObjectKey](#)
- [Creating a form controller](#)
- [Creating a list controller](#)
- [Creating a form factory](#)
- [Building the client workflow assembly](#)

Overview

The client workflow assembly is a Visual Studio Tools for Microsoft Dynamics GP integration that adds the following workflow capabilities:

- Adds the workflow message bar, workflow controls, and workflow history bar to a Microsoft Dynamics GP form.
- Subscribes to workflow events raised by the Microsoft Dynamics GP form or list. Event handlers perform actions based on user interaction with the form or list.
- Uses the Workflow web service to initiate actions, retrieve workflow information, and provide the user with updated workflow status information.

Creating a Visual Studio project

Create a new Visual Studio project.

The client workflow assembly is a Microsoft .NET assembly you create using Visual Studio. The assembly works with Microsoft Dynamics GP forms or lists to provide workflow functionality.

To create a client workflow assembly, complete the following steps:

1. Create a new project.

Open Visual Studio. From the File menu, select File >> New >> Project. In the New Project window, select Visual C# as the Project type. In Templates, select Class Library from the list of Visual Studio installed templates.

Enter a name for the client workflow assembly. Review the Location and Solution Name, and click then OK.

2. Delete the Class1.cs file from the project.

Visual Studio creates a default class file named Class1.cs. From the View menu, select Solution Explorer. In Solution Explorer, delete Class1.cs from your project.

If a dialog window opens asking whether to delete the file, click OK.

3. Add references to the project.

From the Project menu, select Add References. The Add References window opens. Click the .NET tab. Hold the Ctrl key and select the following assemblies:

- System.Windows.Forms
- System.Drawing.

Click OK.

From the Project menu, select Add References. The Add References window opens. Click the Browse tab. Navigate to the Microsoft Dynamics GP client folder, typically found at the following location:

C:\Program Files\Microsoft Dynamics\GP

Hold the Ctrl key and click the following .dll files:

- Application.Dynamics.dll
- Microsoft.Dexterity.Bridge.dll
- Microsoft.Dexterity.Shell.dll
- Microsoft.Dynamics.Common.dll
- Microsoft.Dynamics.GP.Workflow.Client.dll
- Microsoft.Dynamics.Workflow.Common.dll

Click OK.

4. Set Reference properties

From Solution Explorer, expand the list of Reference and select all the References. In Properties, set Copy Local to False.

Creating a BusinessObjectKey

Add a BusinessObjectKey class to the project.

Each document you submit to workflow must have a unique ID. To identify your document, create a class that specifies a key that uniquely identifies individual documents. To ensure workflow can retrieve with your document, your key class must inherit from the Microsoft.Dynamics.Workflow.Common.BusinessObjectKey class.

A BusinessObjectKey is a general-purpose class for specifying the key values for any document type in Microsoft Dynamics GP. A BusinessObjectKey is composed of one or more KeyPart objects, each of which represent one segment of the key for the business document. For additional information on the BusinessObjectKey class, refer to [Business object keys](#) of [Chapter 16, "Using the Web Service."](#)

To create a key class for your business document, complete the following steps:

1. Create a class.

From the Project menu, choose Add Class. Select Class from the Add New Item window. Enter a name for your class and click Add. Visual Studio creates and opens a new class file. The following code sample shows the creation of a class named `LeadBusinessObjectKey`.

```
public class LeadBusinessObjectKey
{
}

```

2. Add namespace references.

Add `using` statements to provide convenient access to the classes and methods needed for this class. Include the following:

- `Microsoft.Dynamics.Workflow.Common`

3. Inherit from the `BusinessObjectKey` class.

Your class must inherit from the `BusinessObjectKey` class. The following code sample shows the `LeadBusinessObjectKey` inherits from `BusinessObjectKey`.

```
public class LeadBusinessObjectKey : BusinessObjectKey
{
}

```

4. Add data members that label your key values.

You need to provide a name for each segment of your document key. Use the database column name that the key segment value corresponds to. The following sample uses `LeadID` for lead documents.

```
private const string leadIdKeyName = "LeadID";

```

5. Add methods that create and retrieve the `KeyParts`.

Your key class must be able to store values that uniquely identify your business document. Store the values in the base class `KeyParts` collection. Use the strings you created earlier to name each `KeyPart`.

The following sample code creates two constructors. The first creates an empty `KeyPart`, gives the `KeyPart` a name, and adds the `KeyPart` to the base class `KeyParts` collection.

```
// Create an empty LeadBusinessObject key
public LeadBusinessObjectKey() : base()
{
    KeyParts.Add(new KeyPart<string>(leadIdKeyName, string.Empty));
}

```

The second constructor also creates a `KeyPart` and adds it to the `KeyPart` collection. This constructor allows the value of the `KeyPart` to be set when it is created.

```
// Create a LeadBusinessObject key that identifies a specific Sales Lead
public LeadBusinessObjectKey(string leadId) : base()
{
    KeyParts.Add(new KeyPart<string>(leadIdKeyName, leadId));
}
```

The following sample code creates a class property that updates or retrieves the value from the `LeadID` `KeyPart`.

```
// Property that uniquely identifies a specific sales lead
public string LeadId
{
    get
    {
        return ((KeyPart<string>)KeyParts[0]).PartValue;
    }
    set
    {
        ((KeyPart<string>)KeyParts[0]).PartValue = value;
    }
}
```

6. Save your class.

Creating a form controller

Add a form controller class to the project.

A workflow form controller is a class that responds to Microsoft Dynamics GP client events, manages communication with the Dynamics Workflow web service, and adds the workflow message bar, workflow controls, and workflow history bar to the Microsoft Dynamics GP client window. You create a separate form controller class for each Microsoft Dynamics GP form for which workflow is being implemented.

To create a new form controller, you need to complete the following tasks:

- Create the class.
- Add static data fields.
- Add the base class and helper methods.
- Add VS Tools event handlers.
- Add client workflow event handlers.
- Add properties.

Create the class

To create the form controller class, complete the following steps:

1. Create the class.

From the Visual Studio Project menu, choose `Add Class`. Select `Class` from the `Add New Item` window. Enter a name for your class and click `Add`.

The following sample code sample shows the creation of class named LeadWorkflowController.

```
class LeadWorkflowController
{
}
}
```

2. Add references to the required assemblies.

The form controller accesses many of the workflow fields and resources you include with application you integrated with Microsoft Dynamics GP. Add a reference to the application assembly you created. For information about creating an application assembly, see [Application assembly](#) on page 67.

From the Project menu, select Add References. The Add References window opens. Click the Browse tab. Browse to the Microsoft Dynamics GP folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GP

Select your application assembly. For example, the lead sample application adds a reference to the Application.SampleIntegratingApp.dll file. Click OK.

3. Add namespace references.

Add **using** statements to provide convenient access to the classes and methods needed for this class. Include the following:

- Microsoft.Dynamics.Workflow.Common
- Microsoft.Dexterity.Applications
- Microsoft.Dexterity.Applications.DynamicsDictionary
- Microsoft.Dynamics.Common
- Microsoft.Dynamics.GP.Workflow
- Microsoft.Dynamics.Workflow
- Microsoft.Dexterity.Applications.SampleIntegratingAppDictionary

Replace the Microsoft.Dexterity.Applications.SampleIntegratingAppDictionary namespace with the namespace from your application assembly

4. Inherit from ApprovalWorkflowController<> class.

Your workflow list controller class must inherit from the abstract base class:

Microsoft.Dynamics.GP.Workflow.ApprovalWorkflowController<WorkflowWrapper>

The following sample code creates a LeadWorkflowController class that inherits from the ApprovalWorkflowController class.

```
class LeadWorkflowController :
ApprovalWorkflowController<WorkflowWrapper>
{
}
}
```

5. Implement the abstract base class.

Your form controller must implement all of the methods and properties specified by the `ApprovalWorkflowController<>` class.



To implement the method and properties of the abstract base class, right click `ApprovalWorkflowController<WorkflowWrapper>` and choose **Implement Abstract Class**. Visual Studio will add the abstract base class methods and properties to your class.

Add data fields to the class.

Add static data fields

Now that you have created a new controller class, you must define the controllers data fields. You need static data members for the following:

- This controller class
- Your Microsoft Dynamics GP client form
- A workflow wrapper object
- Your workflow name



The workflow name must match the name that was specified in the workflow attribute of the workflow server class. If the names are not identical, the client will not be able to attach the workflow to your document. To learn more about naming your workflow, see [Creating a workflow type](#) on page 137.

The following sample code adds the required static data fields to the controller class. The `leadMaintenanceForm` is populated with a reference to the Lead Maintenance form. The `LeadWorkflowName` contains the workflow name.

```
private static LeadWorkflowController controller;

private static IgLeadMaintenanceForm leadMaintenanceForm =
    Microsoft.Dexterity.Applications.SampleIntegratingApp.
    Forms.IgLeadMaintenance;
private static WorkflowWrapper workflowWrapper;

// Make the workflow name public so it can be used by other objects that
// specify the workflow name
public static string LeadWorkflowName =
    "Sample Salesperson Approval Workflow";
```

Add required methods to the class.

Add base class and helper methods

To control the behavior of your controller, implement the base class methods and any other methods needed to support your workflow client. The following represent a minimum set of methods you will need to implement.

Create The base class requires you to provide a Create method. The Create method instantiates your form controller and attaches a workflow wrapper. The workflow wrapper adds the workflow message bar, workflow controls, and workflow history bar to your Microsoft Dynamics GP form. The following sample code implements the Create method for the lead sample application:

```
public static void Create(WorkflowWrapper wrapper)
{
    workflowWrapper = wrapper;

    if (LeadWorkflowController.controller == null)
    {
        LeadWorkflowController.controller = new LeadWorkflowController();
    }

    controller.Initialize(workflowWrapper);
}
```

SubscribeToApplicationEvents The base class requires you to override the SubscribeToApplicationEvents method. This method identifies Dynamic GP events that require action. These events are Visual Studio Tools events. The method must supply an event handler for each of the specified events.



To learn more about Visual Studio Tools events, refer to the Visual Studio Tools for Microsoft Dynamics GP Programmers Guide.

The following sample code for the Lead Maintenance form controller verifies that workflow is active, subscribes to three form events, and specifies the event handler for each event:

```
protected override void SubscribeToApplicationEvents()
{
    // Determine whether the workflow is active
    if (SampleIntegratingApp.Forms.CommandIgSample.Dummy.
        WorkflowEnabled.Value == true)
    {
        // If workflow is enabled, set the indicator flag on the Command form to
        // true. The value of the flag enables the form's Save and Submit button
        SampleIntegratingApp.Forms.CommandIgSample.Dummy.
            WorkflowEnabled.Value = true;

        // Subscribe to application events and associate a handler for each event
        leadMaintenanceForm.LeadMaintenance.LocalSubmit.ClickAfterOriginal +=
            new System.EventHandler(LocalSubmit_ClickAfterOriginal);
        leadMaintenanceForm.LeadMaintenance.WfGetInfo.ValidateAfterOriginal +=
            new System.EventHandler(WfGetInfo_ValidateAfter);
        leadMaintenanceForm.LeadMaintenance.OpenAfterOriginal +=
            new System.EventHandler(LeadMaintenance_OpenAfterOriginal);
    }
}
```

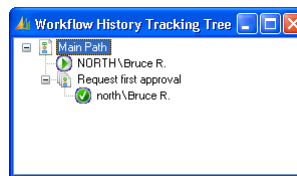
RetrieveSalesLeadTracking When the form loads a document, you need to retrieve workflow history information associated with that document. To retrieve and store this information, add a “helper” method that retrieves the document’s workflow history from the server and adds the history information to the form controller’s tracking information.

For the Lead Maintenance form, this “helper” method was named `RetrieveSalesLeadTracking`. You can choose a different name when you create a “helper” method for your document.



The `LeadWorkflowController` example calls `RetrieveSalesLeadTracking` in the event handler that updates the workflow message bar, workflow controls, and workflow history bar. The event occurs when new document opens in the Lead Maintenance form.

Add the history information to the controller’s workflow wrapper `TrackingInstance` collection. This allows the document’s history information to display in the Workflow History Tracking Tree.



The following sample code uses a workflow agent to retrieve the workflow history associated with the current lead document. The workflow agent is a “helper” class that you can use to work with the Dynamics GP Workflow web service. The history information populates the tracking history object. To display the history information, the history object is added to the form controller’s workflow wrapper.

```
private void RetrieveSaleLeadTracking()
{
    // Only retrieve tracking information if the current business object
    // is not null
    if (CurrentBusinessObject != null)
    {
        // Create a new workflow agent object
        WorkflowAgent workflowAgent = WorkflowHelper.GetWorkflowAgent();

        // Initialize a tracking object to null
        Tracking currentTracking = null;

        try
        {
            // Retrieve tracking information for the current sales lead
            System.Collections.ObjectModel.
                Collection<Tracking> trackingsList;
            trackingsList =
                workflowAgent.GetTrackingInformation(
                    CurrentBusinessObject.Key, WorkflowHelper.
                    GetAssociationKey(WorkflowName));
        }
    }
}
```

```

// Populate the tracking object with the results from the server
foreach (Tracking tracking in trackingsList)
{
    if (currentTracking == null)
    {
        currentTracking = tracking;
    }
    else
    {
        if (tracking.TimeTracking.Start.CompareTo(
            currentTracking.TimeTracking.Start) > 0)
        {
            currentTracking = tracking;
        }
    }
}
}
catch
{
    // If the call fails, set tracking info to null so the Workflow
    // History Tracking Tree is not displayed. Do not report this
    // as an error.
    workflowWrapper.TrackingInstance = null;
}

// Populate the workflow wrapper for the window with tracking
// information. This will cause the Workflow History Tracking Tree
// to be available in the Workflow History bar.
workflowWrapper.TrackingInstance = currentTracking;
}
else
{
    // If the current business object is empty, set tracking info to
    // null so the Workflow History Tracking Tree is not displayed.
    workflowWrapper.TrackingInstance = null;
}
}
}

```

Add the event handlers that add workflow functionality to the form.

Add VS Tools event handlers

In the `SubscribeToApplicationEvents` method you specified event handlers for form events. You must now implement these event handlers. These event handlers add workflow functionality to your Microsoft Dynamics GP form.

To add workflow to your form, you must implement event handlers that perform the following tasks:

- Update the workflow message bar, workflow controls, and history bar when the form loads a new document.
- Open the workflow submit dialog window. The workflow submit dialog window allows an approver to approve, reject, request changes or delegate the current document.

The following sample code implements an event handler that updates the workflow controller when a lead document is opened in the Lead Maintenance window. This event handler runs when the run script statement is used for the WFGetInfo hidden field in the Lead Maintenance window. This is one of the hidden fields that was added to support workflow.

Notice how the event handler updates the controller business document, updates the workflow message bar, workflow controls, and workflow history bar, and calls the RetrieveSalesLeadTracking method to load the document's workflow history.

```
void WfGetInfo_ValidateAfter(object sender, EventArgs e)
{
    ResetWrapperDisplay();

    // Populate the business object with the current record using the
    // key for the object
    SetCurrentBusinessDocument(BusinessObjectKey);
    if (CurrentBusinessObject != null)
    {
        // Use the base class method to get and display the workflow data
        DisplayWorkflowState();

        // If this is an existing sales lead, populate and display
        // the Workflow History Tracking Tree
        if (IsNew == false)
        {
            RetrieveSaleLeadTracking();
        }
    }
}
```

The following sample opens the workflow submit dialog window when an approver clicks one of the buttons in the workflow controls. Notice how the event handler validates the workflow status before displaying the dialog window.

```
void LocalSubmit_ClickAfterOriginal(object sender, EventArgs e)
{
    if (ApprovalWorkflowStatus !=
        (short)GPAApprovalWorkflowStatusValue.PendingApproval)
    {
        DisplaySubmitDialog();
    }
}
```

You may need to clear the workflow controllers data cache to force the controller to retrieve updated workflow information for a document. The following code sample resets the controller's data cache when the Lead Maintenance window is updated.

```
void LeadMaintenance_OpenAfterOriginal(object sender, EventArgs e)
{
    ResetWrapperDisplay();
}
```

Add client event handlers to the class.

Client workflow event handlers

Add event handlers that respond to a workflow action that a user performs using the form. The workflow form controller responds to workflow events that originate with the form. You will add code to these events to keep the Dynamics GP window in the proper state after the user has performed a workflow operation. The following table lists the client workflow events:

Event	Description
OnSubmitted	The originator submits the document to workflow for approval.
OnApproved	The approver clicks the Approved button on the workflow controls.
OnRequestedChange	The approver clicks the Request Change button on the workflow controls.
OnDelegated	The approver clicks the Delegate button on the workflow controls.
OnRejected	The approver clicks the Reject button on the workflow controls.

The following sample codes shows how to override the base class event handlers for each of these workflow events.

OnSubmitted Notice how the event handler updates the controller's workflow priority property, calls a Dexterity procedure to save the priority setting, and clears the Lead Maintenance window.

```
// When a lead assignment is submitted, update the workflow priority value,
// save that change and restart the Lead Maintenance form
protected override void OnSubmitted(string comment, int? daysToAct,
    short priority)
{
    WorkflowPriority = priority;
    Microsoft.Dexterity.Applications.SampleIntegratingApp.Forms.
        IgLeadMaintenance.Procedures.UpdateWorkflowPriority.
        Invoke(this.CurrentBusinessObject.Key.KeyParts[0].
            GetPartValueAsString(), priority);
    Microsoft.Dexterity.Applications.SampleIntegratingApp.
        Forms.IgLeadMaintenance.LeadMaintenance.LocalRestartForm.
        RunValidate();
}
```

OnApproved Notice how the event handler updates the controller's workflow priority property, calls a Dexterity procedure to save the priority setting, and clears the Lead Maintenance window.

```
// When a lead assignment is approved, update the workflow priority value,
// save that change and restart the Lead Maintenance form
protected override void OnApproved(string comment, int? daysToAct, short
priority)
{
    WorkflowPriority = priority;
    Microsoft.Dexterity.Applications.SampleIntegratingApp.Forms.
        IgLeadMaintenance.Procedures.UpdateWorkflowPriority.
        Invoke(this.CurrentBusinessObject.Key.KeyParts[0].
            GetPartValueAsString(), priority);
}
```

```

        Microsoft.Dexterity.Applications.SampleIntegratingApp.Forms.
            IgLeadMaintenance.LeadMaintenance.LocalRestartForm.RunValidate();
    }

```

OnRequestedChange Notice how the event handler updates the controller's workflow priority property, calls a Dexterity procedure to save the priority setting, and clears the Lead Maintenance window.

```

// When additional information is requested, update the workflow priority
// value, save that change and restart the Lead Maintenance form
protected override void OnRequestedChange(string comment, short priority)
{
    WorkflowPriority = priority;
    Microsoft.Dexterity.Applications.SampleIntegratingApp.Forms.
        IgLeadMaintenance.Procedures.UpdateWorkflowPriority.
        Invoke(this.CurrentBusinessObject.Key.KeyParts[0].
            GetPartValueAsString(), priority);
    Microsoft.Dexterity.Applications.SampleIntegratingApp.Forms.
        IgLeadMaintenance.LeadMaintenance.LocalRestartForm.RunValidate();
}

```

OnDelegated Notice how the event handler updates the controller's workflow priority property, calls a Dexterity procedure to save the priority setting, and clears the Lead Maintenance window.

```

// When a lead assignment is delegated, update the workflow priority value,
// save that change and restart the Lead Maintenance form
protected override void OnDelegated(string comment, short priority, string
delegateToUser)
{
    WorkflowPriority = priority;
    Microsoft.Dexterity.Applications.SampleIntegratingApp.Forms.
        IgLeadMaintenance.Procedures.UpdateWorkflowPriority.Invoke(
        this.CurrentBusinessObject.Key.KeyParts[0].GetPartValueAsString(),
        priority);
    Microsoft.Dexterity.Applications.SampleIntegratingApp.Forms.
        IgLeadMaintenance.LeadMaintenance.LocalRestartForm.RunValidate();
}

```

OnRejected Notice how the event handler calls a Dexterity procedure to clear the Lead Maintenance window.



When rejecting a document, the approver cannot change the workflow priority. As a result, the workflow controller does not need to update the workflow priority.

```

// When a lead assignment is rejected, restart the Lead Maintenance form
protected override void OnRejected(string comment)
{
    Microsoft.Dexterity.Applications.SampleIntegratingApp.Forms.
        IgLeadMaintenance.LeadMaintenance.LocalRestartForm.RunValidate();
}

```

Add properties

Add the required properties to the class.

The `ApprovalWorkflowController<WorkflowWrapper>` abstract base class requires you to override the following properties in your workflow controller:

Property	Description
<code>ApprovalWorkflowStatus</code>	Returns an integer that represents workflow status associated with the current business document. The integer value is from the <code>WorkflowApprovalStatus</code> enumeration.
<code>BusinessObjectKey</code>	Returns a business object key that uniquely identifies the current business document.
<code>DisplayOnlyWrapper</code>	A boolean value that specifies whether the controller displays the window's workflow wrapper but does not allow a user to perform workflow operations. Typically, the value of this property is set to <code>False</code> .
<code>Edited</code>	A boolean value that specifies whether the document has been changed.
<code>IsNew</code>	A boolean value that specifies whether the document is a new document.
<code>SynchronousServiceCall Success</code>	A boolean value that specifies whether the last synchronous call to the Workflow web service succeeded or failed.
<code>WorkflowName</code>	Returns a string value that specifies the workflow associated with this workflow controller.
<code>WorkflowPriority</code>	An integer value that represents the workflow priority. The integer value is from the <code>Priority</code> enumeration.

The following sample code shows how these required properties were implemented for the Lead Maintenance form controller. Notice how the `ApprovalWorkflowStatus`, `BusinessObjectKey`, `Edited`, `IsNew`, and `WorkflowPriority` properties use the Lead Maintenance form to store the property value.

```
// Returns the workflow status for a Sales Lead
protected override short ApprovalWorkflowStatus
{
    get
    {
        short statusValue;

        // Get the workflow approval status from the form
        string status = Microsoft.Dexterity.Applications.
            SampleIntegratingApp.Forms.IgLeadMaintenance.
            LeadMaintenance.WorkflowApprovalStatus.Value.ToString();

        // Set the status value for the return value
        // If status is empty return as Not Submitted otherwise return the
        // current status value
        if (string.IsNullOrEmpty(status))
        {
            statusValue = (short)GPApprovalWorkflowStatusValue.NotSubmitted;
        }
        else if (status == "0")
        {
            statusValue = (short)GPApprovalWorkflowStatusValue.Unknown;
        }
    }
}
```

```

        else
        {
            statusValue = Convert.ToInt16(status);
        }

        return statusValue;
    }
}

// The LeadBusinessObjectKey that uniquely identifies the Sales Lead
protected override BusinessObjectKey BusinessObjectKey
{
    get { return new LeadBusinessObjectKey(
        Microsoft.Dexterity.Applications.SampleIntegratingApp.
        Forms.IgLeadMaintenance.LeadMaintenance.LeadId.Value); }
}

// Controls whether to display only the workflow wrapper
protected override bool DisplayOnlyWrapper
{
    get { return false; }
}

// Specifies whether the lead information in the Lead Maintenance form has
// been changed
protected override bool Edited
{
    get {
        return Microsoft.Dexterity.Applications.SampleIntegratingApp.
            Forms.IgLeadMaintenance.LeadMaintenance.IsChanged; }
}

// Specifies whether the lead in the Lead Maintenance form is a new lead
protected override bool IsNew
{
    get
    {
        if (Microsoft.Dexterity.Applications.SampleIntegratingApp.Forms.
            IgLeadMaintenance.LeadMaintenance.DisplayExistingRecord == false)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
}

```



```

// Specifies whether a synchronous web service call has succeeded or failed
// For this example, we always return true
protected override bool SynchronousServiceCallSuccess
{
    get
    {
        return true;
    }
    // Set is required by the abstract base class but this class does not
    // support changes to this property
    set
    {
    }
}

// Specifies the name of the workflow
protected override string WorkflowName
{
    get
    {
        return LeadWorkflowName;
    }
}

// Specifies the workflow priority of the specified lead
protected override short WorkflowPriority
{
    get
    {
        return Microsoft.Dexterity.Applications.SampleIntegratingApp.
            Forms.IgLeadMaintenance.LeadMaintenance.WorkflowPriority;
    }
    set
    {
        Microsoft.Dexterity.Applications.SampleIntegratingApp.
            Forms.IgLeadMaintenance.LeadMaintenance.WorkflowPriority.
            Value = value; ;
    }
}

```

Creating a list controller

Add a list controller class to the project.

A workflow list controller is a class that responds to Microsoft Dynamics GP client events, and manages communication with the Dynamics Workflow web service for a Microsoft Dynamics GP list. You create a separate form controller class for each Microsoft Dynamics GP list for which workflow is being implemented.

To create a new form controller, you need to complete the following tasks:

- Create the class.
- Add static data fields.
- Add the base class and helper methods.
- Add VS Tools event handlers.
- Add client workflow event handlers.
- Add properties.

Create the class

To create the list controller class, complete the following steps:

1. Create a class.

From the Visual Studio Project menu, choose Add Class. Select Class from the Add New Item window. Enter a name for your class and click Add.

The following code sample shows the creation of a class named `LeadListWorkflowController`, which is the list controller for the Leads list in the sample integration.

```
class LeadListWorkflowController
{
}

```

2. Add references to the required assemblies.

The list controller accesses many of the workflow fields and resources you included with the application you integrated with Microsoft Dynamics GP. Add a reference to the application assembly you created.

From the Project menu, select Add References. The Add References window opens. Click the Browse tab. Browse to the Microsoft Dynamics GP client folder, typically found in the following location:

```
C:\Program Files\Microsoft Dynamics\GP
```

Select your application assembly. For example, the lead sample application includes a reference to the `Application.SampleIntegratingApp.dll` file. Click OK.

3. Add namespace references.

Add `using` statements to provide convenient access to the classes and methods needed for this class. Include the following:

- `System.Collections.ObjectModel`
- `Microsoft.Dynamics.Workflow.Common`
- `Microsoft.Dexterity.Applications`

In the using statement for the sample application, you can assign the namespace an alias. The following sample code shows how to assign App to the Microsoft.Dexterity.Applications namespace:

```
using App = Microsoft.Dexterity.Applications;
```

In the following code samples, App refers to the namespace of the Application.SampleIntegratingApp application assembly.

4. Inherit from the ListWorkflowController class.

Your workflow list controller class must inherit from the following abstract base class:

```
Microsoft.Dynamics.GP.Workflow.ListWorkflowController
```

The following sample code example creates a new LeadListWorkflowController class that inherits from the ListWorkflowController class.

```
class LeadListWorkflowController : ListWorkflowController
{
}
}
```

5. Implement the abstract base class.

Your list controller must implement all of the methods and properties specified by the ListWorkflowController class.



*To implement the methods and properties of the abstract base class, right click ListWorkflowController and choose **Implement Abstract Class**. Visual Studio adds the abstract base class methods and properties to your class.*

Add static data fields

Now that you have created a new list controller class, you must define the controller's data fields. You need a static data member that stores an instance of your controller class.

The following sample code sample adds the required static data field to the list controller class:

```
private static LeadListWorkflowController controller;
```

Add the base class methods

To control the behavior of your list controller, implement the base class methods and any other methods needed to support your workflow for the list. The following represent a minimum set of methods.

Create The Create method instantiates the list controller when your list is opened. The following sample code implements the Create method for the lead list controller. Notice how the method ends by calling the Initialize method of the base class:

```
public static void Create()
{
    if (LeadListWorkflowController.controller == null)
    {
        LeadListWorkflowController.controller =
            new LeadListWorkflowController();
    }

    controller.Initialize();
}
```

SubscribeToApplicationEvents The abstract base class requires the list controller to override the SubscribeToApplicationEvents method. This method identifies the Dynamic GP events for the list that require action. The method must supply an event handler for each of the specified events. SubscribeToApplicationEvents is automatically invoked during the creation of the controller.

The events are all Visual Studio Tools events. These events occur when a user's actions invoke the Dexterity procedures you added to support workflow functionality.

The following sample code verifies that workflow is active, subscribes to four events, and specifies the event handler for each event.

```
protected override void SubscribeToApplicationEvents()
{
    // Create a reference to the client application stateWindow
    App.SampleIntegratingAppDictionary.ListObjLeadsForm.StateWindow
stateWindow = App.SampleIntegratingApp.Forms.ListObjLeads.State;

    // Specify the event handlers for stateWindow events
stateWindow.WfResetWorkflowHandler.ValidateAfterOriginal +=
    new EventHandler(WfResetWorkflowHandler);
stateWindow.LocalWfTransferInitiate.ValidateAfterOriginal +=
    new EventHandler(TransferInitiateHandler);
stateWindow.LocalWfTransfer.ValidateAfterOriginal +=
    new EventHandler(TransferHandler);
stateWindow.LocalWfTransferComplete.ValidateAfterOriginal +=
    new EventHandler(TransferCompleteHandler);
}
```

TransmitSubmissionResultToApplication The abstract base class requires your list controller to override the `TransmitSubmissionResultToApplication` method. When a workflow operation is performed for the documents marked in the list, this method is called once for each document that encounters a workflow error. The code in this method invokes a Dexterity form procedure to log the error for display by the list.

```
protected override void TransmitSubmissionResultToApplication(
    AsyncCallbackUserState asyncCallbackUserState,
    Microsoft.Dynamics.Workflow.Common.BusinessObjectKey key,
    string errorMessage)
{
    // Create a LeadBusinessObjectKey that identifies the sales lead
    LeadBusinessObjectKey leadKey = (LeadBusinessObjectKey)key;

    // Call the forms WfRequestReturn procedure and pass the information
    // received from workflow This allows the client form to reflect
    // whether workflow succeeded or failed for each Sales Lead
    App.SampleIntegratingApp.Forms.ListObjLeads.Procedures.
        WfRequestReturn.Invoke(asyncCallbackUserState,
            ApplicationAsyncCallbackId,
            leadKey.LeadId,
            asyncCallbackUserState.ApplicationAsyncCallbackDate,
            asyncCallbackUserState.ApplicationAsyncCallbackTime,
            errorMessage);
}
```

Add VS Tools event handlers

In the `SubscribeToApplicationEvents` method, you specify event handlers for list events. You must now implement these event handlers. The event handlers add workflow functionality to your Microsoft Dynamics GP list.

The following sample code implements an event handler that forces the list controller to refresh its data cache. This forces the Dynamics GP client to reload the document's current workflow information.

```
private void WfResetWorkflowHandler(object sender, EventArgs args)
{
    ResetController();
}
```

The next three event handlers respond to events raised by the Dexterity `WfActionForList` procedure that was added to the list. For more information about this procedure, refer to [List integration](#) on page 53.

The `TransferInitiate` event occurs once and indicates that a workflow action was selected. The `TransferInitiateHandler` uses the `SetTargetedAction` method to specify the workflow action to perform.

In the following sample code, the event handler retrieves the value of the WFTransferInitiate hidden field from the list to specify the selected workflow action. The event handler also retrieves the value of the ActionStatusID, DateString, and TimeString hidden fields from the list.

```
// The application uses this event to initiate the approval workflow
// for Sales Leads
private void TransferInitiateHandler(object sender, EventArgs args)
{
    SetTargetedAction(
        (GPWorkflowAction)(App.SampleIntegratingApp.Forms.ListObjLeads.State.
            LocalWfTransferInitiate.Value),
        App.SampleIntegratingApp.Forms.ListObjLeads.State.ActionStatusId,
        App.SampleIntegratingApp.Forms.ListObjLeads.State.LocalDateString,
        App.SampleIntegratingApp.Forms.ListObjLeads.State.LocalTimeString);
}
```

The TransferHandler event occurs for each document in the list that was marked. The TransferHandler event handler adds the records to the list controller's collection.

The following sample code adds leads to the list controller collection. To perform the workflow action, the controller submits this collection of leads to the Dynamics GP Workflow web service. Notice how the event handler sets a value on the form to signal whether the entry was successfully added to the collection.

```
private void TransferHandler(object sender, EventArgs args)
{
    bool transferSucceeded = AddEntryForTargetedAction();
    App.SampleIntegratingApp.Forms.ListObjLeads.State.LocalWfTransfer.Value =
        transferSucceeded;
}
```

The TransferComplete event occurs once and initiates the processing of the list documents that were marked. The TransferCompletedHandler event handler sends the completed collection to workflow to be processed.

The following sample code shows how to use the InvokeTargetedAction method to have the Workflow web service perform the specified action on the collection of leads that was assembled by the TransferHandler event handler.

```
void TransferCompleteHandler(object sender, EventArgs e)
{
    InvokeTargetedAction();
}
```

Add properties

The `ListWorkflowController` abstract base class requires you to override the following properties in your list controller:

Property	Description
<code>ActionComment</code>	A string that specifies the comment the approver entered in the workflow submit dialog window.
<code>ActionDaysToAct</code>	An integer that represents the number of days the approvers entered in the workflow submit dialog window.
<code>ActionDelegateToUser</code>	A string that specifies the user that the approver selects as an alternate approver.
<code>ActiveWorkflowNames</code>	A collection of strings that specify the workflows available for the current document type.
<code>BusinessObjectKey</code>	A business object key that uniquely identifies the business document.
<code>SynchronousServiceCallSuccess</code>	A boolean value that specifies whether the last Workflow web service call succeeded.
<code>WorkflowName</code>	A string that identifies the specific workflow associated with the list controller.

The following sample code shows how to implement the required properties. Notice how the `ActionComment`, `ActionDaysToAct`, `ActionDelegateToUser`, `ActiveWorkflowNames`, `BusinessObjectKey`, and `SynchronousServiceCallSuccess` use fields on the `ListObjLeads` or `CommandIgSample` forms to store the property value. To identify its workflow, the list controller's `WorkflowName` property uses the static `LeadWorkflowName` data member from the `LeadWorkflowController` class.

```
// The comment entered in the drop down dialog
protected override string ActionComment
{
    get
    {
        return App.SampleIntegratingApp.Forms.
            ListObjLeads.State.LocalComment;
    }
}

// The number of days to act entered in the drop down dialog
protected override int? ActionDaysToAct
{
    get
    {
        return GPClientHelper.ConvertDaysToActValue(
            App.SampleIntegratingApp.Forms.ListObjLeads.
                State.LocalChangeDaysToAct,
            App.SampleIntegratingApp.Forms.ListObjLeads.State.LocalDaysToAct);
    }
}
```

```

// The user selected in the drop down dialog that identifies the person
// selected as an alternate approver
protected override string ActionDelegateToUser
{
    get
    {
        return App.SampleIntegratingApp.Forms.ListObjLeads.State.
            LocalDelegateUser;
    }
}

// A collection of strings indicating the enabled workflows for the
// current type
protected override System.Collections.ObjectModel.Collection<string>
ActiveWorkflowNames
{
    get
    {
        Collection<string> activeWorkflowNames = new Collection<string>();

        if (App.SampleIntegratingApp.Forms.CommandIgSample.Dummy.
            WorkflowEnabled.Value == true)
        {
            activeWorkflowNames.Add(WorkflowName);
        }

        return activeWorkflowNames;
    }
}

// The BusinessObjectKey that specifies a Lead
protected override Microsoft.Dynamics.Workflow.Common.BusinessObjectKey
BusinessObjectKey
{
    get
    {
        LeadBusinessObjectKey leadKey = new LeadBusinessObjectKey(
            App.SampleIntegratingApp.Forms.ListObjLeads.State.LeadId);
        return leadKey;
    }
}

// Passes the success or failure status for synchronous web service calls
// back to state window of the current form
protected override bool SynchronousServiceCallSuccess
{
    get
    {
        return App.SampleIntegratingApp.Forms.ListObjLeads.State.
            LocalWfVerifySuccess;
    }
}

```



```

        set
        {
            App.SampleIntegratingApp.Forms.ListObjLeads.State.
                LocalWfVerifySuccess.Value = value;
        }
    }

    // The name of the workflow for sales lead approvals
    protected override string WorkflowName
    {
        get
        {
            return LeadWorkflowController.LeadWorkflowName;
        }
    }
}

```

Creating a form factory

Add form factory classes to the project.

A form factory is Microsoft .NET managed code that runs immediately before a Dexterity window is created. A form factory can be used for many purposes, such as adding managed code controls to the window.

To add workflow functionality, you create form factories for each form or list that use your workflow. A workflow form factory performs the following tasks:

- Determines whether a specified workflow is active.
- Creates the workflow controller class for the form or list.

Create the form factory classes

You create a separate form factory class for each form or list controller that you implement. Add your form factory classes to your existing client workflow assembly project.

To create a form factory class, complete the following steps:

1. Create a class.

From the Visual Studio Project menu, choose Add Class. Select Class from the Add New Item window. Enter a name for your class and click Add. To add additional form factory classes, repeat this step or manually add the class to the file and namespace created by Visual Studio.

The following sample code shows the creation of the LeadWorkflowFormFactory class and the WorkflowStatusFormFactory class:

```

class LeadWorkflowFormFactory
{
}

class WorkflowStatusFormFactory
{
}

```

2. Add namespace references.

Add **using** statements to provide convenient access to the classes and methods needed for this class. Include the following:

- Microsoft.Dexterity.Shell
- Microsoft.Dynamics.GP.Workflow
- Microsoft.Dynamics.Workflow
- Microsoft.Dynamics.Workflow.Common
- Microsoft.Dexterity.Applications

3. Inherit from the FormFactory class.

Your form factory must inherit from the Microsoft.Dexterity.Shell.FormFactory class.

The following code sample adds inheritance to the LeadWorkflowFormFactory, and the WorkflowStatusFormFactory class:

```
class LeadWorkflowFormFactory : FormFactory
{
}

class WorkflowStatusFormFactory : FormFactory
{
}
```

4. Override the base class CreateDexForm method.

To add workflow functionality to a standard Dexterity window, you must override the CreateDexForm method. Your override must determine whether your workflow is active, and create the workflow controller for the window.

The following sample code creates a form factory class for the sample application's Lead Maintenance form. The form factory overrides the CreateDexForm method.

Notice how the form factory checks the value of the WorkflowEnabled field of the CommandIgSample form. This value indicates whether the workflow for leads is active. If the workflow is active, the form factory creates a workflow wrapper passes it to a new instance of the LeadWorkflowController. It then returns the wrapper. These steps add the workflow message bar, workflow controls, and workflow history bar to Lead Maintenance window. It also attaches the workflow controller to the window.

```
class LeadWorkflowFormFactory : FormFactory
{
    protected override DexForm CreateDexForm(string name,
        DexWindowType windowType,
        FormFlags flags,
        WindowState windowState,
        System.Drawing.Size minClientSize,
        System.Drawing.Size clientSize,
        bool hResizable,
        bool vResizable,
        short productId,
        short formId,
```

```

        short windowId,
        bool modified)
    {
        // Determine whether workflow is enabled for the company
        // If workflow is enabled, add the workflow wrapper to the form
        // If workflow is not enabled, do not add the workflow wrapper
        if(SampleIntegratingApp.Forms.CommandIgSample.Dummy.
            WorkflowEnabled.Value == true)
        {
            WorkflowWrapper wrapper = new WorkflowWrapper();
            LeadWorkflowController.Create(wrapper);
            return wrapper;
        }
        else
        {
            return base.CreateDexForm(name, windowType,
                flags, windowState,minClientSize, clientSize,
                hResizable, vResizable,productId, formId, windowId,
                modified);
        }
    }
}

```

5. Override the base class **CreateDexDialogForm** method.

To add a form factory to a modal Dexterity window, override the `CreateDexDialogForm` method. Use this override to determine whether your workflow is active, store that workflow status on the command form, and create a workflow controller for your list.

The following sample code creates a form factory for a Dexterity modal window. To add workflow to a list, you cannot use a client open event to execute the form factory. To execute a form factory for the list, the sample code uses a **Sample Integrating App** dialog window that runs when Microsoft Dynamics GP starts. Since the dialog window executes the form factory, override the base class `CreateDexDialogForm` method.

Notice how the form factory determines whether the workflow is enabled. The `WorkflowAgent` allows you to check the status of a specified workflow. In this example, a **WorkflowAssociationKey** is created and uses the `WorkflowName` property from the controller class to identify the workflow. The form factory stores the status in the `WorkflowEnabledField` of the `CommandIgSample` form. This value can be used by other form factories like the previous `LeadWorkflowFormFactory`.

If the workflow is enabled, a workflow controller is created for the Lead list. This allows the list to display the workflow controls and complete workflow actions. The method ends by allowing the dialog window to display.

```
class WorkflowStatusFormFactory : FormFactory
{
    protected override DexDialogForm CreateDexDialogForm(string name,
        DexWindowType windowType,
        FormFlags flags,
        WindowState windowState,
        System.Drawing.Size minClientSize,
        short productId,
        short formId,
        short windowId,
        bool modified)
    {
        // Create a workflow agent object
        WorkflowAgent workflowAgent = WorkflowAgent.CreateInstance(
            GPCClientHelper.GetWorkflowServiceUrl());

        try
        {
            // Create a workflow association key using this controller's
            // WorkflowName property. Use the key with the workflow agent
            // to determine whether the named workflow is activated or
            // deactivated
            WorkflowAssociationKey wfAssocKey = WorkflowHelper.
                GetAssociationKey(LeadWorkflowController.
                    LeadWorkflowName);
            bool isEnabled = workflowAgent.IsWorkflowEnabled(wfAssocKey);

            // If workflow is enabled, set the indicator flag on the
            // Command form to true. If workflow is not enabled, set the
            // indicator flag on the Command form to false. The Lead List
            // form checks the value of the Command form's WorkflowEnabled
            // flag to determine whether to enable or disable the Lead List
            // form's workflow buttons
            if (isEnabled == true)
            {
                SampleIntegratingApp.Forms.
                    CommandIgSample.Dummy.WorkflowEnabled.Value = true;

                // If the workflow is enabled, instantiate a
                // LeadListWorkflow controller object
                LeadListWorkflowController.Create();
            }
            else
            {
                SampleIntegratingApp.Forms.
                    CommandIgSample.Dummy.WorkflowEnabled.Value = false;
            }
        }
    }
}
```

```

catch (Exception err)
{
    // If an error occurs, indicate that workflow is not enabled
    SampleIntegratingApp.Forms.
        CommandIgSample.Dummy.WorkflowEnabled.Value = false;
}

return base.CreateDexDialogForm(name, windowType, flags,
    windowState, minClientSize, productId, formId, windowId,
    modified);
}
}

```

Building the client workflow assembly

Build the client workflow assembly.

Open your Visual Studio client workflow assembly solution. From the Build menu, choose Build. Visual Studio builds your client workflow assembly and places the assembly file in the project's "\bin\debug" folder.

When you deploy your workflow solution, you must install the client workflow assembly to the install folder of your Microsoft Dynamics GP client. For more information about deploying the client workflow assembly, see [Installing the client workflow assembly](#) on page 171.

Chapter 8: Web Service

Microsoft Dynamics GP Workflow uses Web Services for Microsoft Dynamics GP to retrieve documents. You must also use the Dynamics GP web service to access documents for any new workflow type you create. The following sections describe how to create a new web service that builds upon the Web Services for Microsoft Dynamics GP platform, and is suitable for use with workflow.

- [Creating a document type](#)
- [Creating a web service](#)
- [Securing the web service](#)
- [Testing the web service](#)

Creating a document type

Create an assembly for your business document.

To use a web service to retrieve your document, first create .NET assembly that represents your business document. To work with Web Services for Microsoft Dynamics GP, your assembly must include the following three classes.

Class	Description
Document type	The class that defines your business document data. It contains properties you populate to create the document.
Summary	An object that contains only the most important details of the document type.
Criteria	An object that contains the restrictions that define what is to be returned from a GetList method in the Dynamics GP web service.

Create a Visual Studio project

To create a .NET assembly to represent your document, create a new Visual Studio project. To create the project, complete the following procedure:

1. Create a new project.

Open Visual Studio. From the File menu, select File >> New >> Project. In the New Project window, select Visual C# in the Project types list. In Templates, select Class Library in the list of Visual Studio installed templates.

Enter a name for your project. Review the Location and Solution Name, and then click OK.

2. Delete the Class1.cs file from the project.

Visual Studio creates a default class file named Class1.cs. From the View menu, choose Solution Explorer. In Solution Explorer, delete Class1.cs from your project.

If a dialog window opens asking whether to delete the file, click OK.

3. Add references to the project.

Open the Project menu and choose Add References. The Add References window opens. Click the Browse tab and navigate to the Dynamics GP web services “bin” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

Select the following assemblies and click OK:

- Microsoft.Dynamics.Common.dll
- Microsoft.Dynamics.Common.Types.dll
- Microsoft.Dynamics.GP.BusinessLogic.dll

Create the document type class

The first part of the assembly is a class that defines the data members of your back-office document. This class allows the web service to create objects that represent your document.

To create a document type class, complete the following steps:

1. Create a class.

From the Visual Studio Project menu, choose Add New Item. Select Class from the list of Templates in the Add New Item window. Enter a name for your class and click Add. Visual Studio adds a class to your namespace.

The following sample code shows the creation of the Lead class that defines the lead document type for the sample application.

```
using System;
using System.Collections.Generic;
using System.Text

namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    class Lead
    {
    }
}
```

2. Add namespace references.

Add **using** statements to provide convenient access to the classes and methods needed for this class. Include the following:

- System.Runtime.InteropServices
- System.Xml.Serialization
- Microsoft.Dynamics.Common
- Microsoft.Dynamics.GP

Add a class to the project that describes the document.

3. Add class attributes.

Web Services for Microsoft Dynamics GP expects two attributes. You must add an attribute that makes the class serializable. You must also provide a GUID attribute that uniquely identifies your class. To generate a GUID for your class, open the Visual Studio Tools menu and choose Create GUID.

The following sample code adds the required attributes to the Lead class from the sample application.

```
namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    [Serializable]
    [GuidAttribute("2852BB26-3BA8-4663-9613-033327D7F3C2")]
    class Lead
    {
    }
}
```

4. Inherit from the BusinessObject class.

To leverage the existing Dynamics GP web services platform, have your class inherit from the Microsoft.Dynamics.Common.BusinessObject class. The BusinessObject class is the base class used by all back-office documents in Web Services for Microsoft Dynamics GP. Also, make your class public.

The following sample code adds the BusinessObject base class the Lead class.

```
namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    [Serializable]
    [GuidAttribute("2852BB26-3BA8-4663-9613-033327D7F3C2")]
    public class Lead : BusinessObject
    {
    }
}
```

5. Add the class data members.

Add private data members that represent the data from your document. To add money or date fields, use the classes supplied by Web Services for Microsoft Dynamics GP.

The following sample code adds the data members for lead documents. Notice how the Potential Revenue amount uses the Dynamics GP web services MoneyAmount and the date fields use the System.DateTime class. The potentialRevenue data member is not private. This allows the Currency and DecimalDigit properties of potentialRevenue to be updated.

```
namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    [Serializable]
    [GuidAttribute("2852BB26-3BA8-4663-9613-033327D7F3C2")]
    public class Lead : BusinessObject
    {
        private LeadKey key;
        private string name;
    }
}
```

```

        private string salespersonID;
        private string city;
        private string state;
        private string zip;
        private string address1;
        private string address2;
        private string phone1;
        private string phone2;
        private string fax;
        private LeadCategory? leadBusinessCategory;
        private string country;
        private string contact;
        MoneyAmount potentialRevenue;
        private bool? qualifiedLead;
        private string leadSource;
        private DateTime qualificationDate;
        private string workflowApprovalStatus;
        private string workflowPriority;
        private string approvedSalespersonID;
        private DateTime modifiedDate;
    }
}

```

6. Add a class property for each data member.

Add a class property that provides access to each data field. The property allows you to set or retrieve the value of each data field.

The following sample code adds properties for leads:

```

public LeadKey Key
{
    get{return key;}
    set{key = value;}
}
public string Name
{
    get{return name;}
    set{name = value;}
}
public string SalespersonID
{
    get{return salespersonID;}
    set{salespersonID = value;}
}
public string City
{
    get{return city;}
    set{city = value;}
}
public string State
{
    get{return state;}
    set{state = value;}
}

```

```

public string Zip
{
    get{return zip;}
    set{zip = value;}
}
public string Address1
{
    get{return address1;}
    set{address1 = value;}
}
public string Address2
{
    get{return address2;}
    set{address2 = value;}
}
public string Phone1
{
    get{return phone1;}
    set{phone1 = value;}
}
public string Phone2
{
    get{return phone2;}
    set{phone2 = value;}
}
public string Fax
{
    get{return fax;}
    set{fax = value;}
}
public LeadCategory? LeadBusinessCategory
{
    get{return leadBusinessCategory;}
    set{leadBusinessCategory = value;}
}
public string Country
{
    get{return country;}
    set{country = value;}
}
public string Contact
{
    get{return contact;}
    set{contact = value;}
}
public MoneyAmount PotentialRevenue
{
    get{return potentialRevenue;}
    set{potentialRevenue = value;}
}
public bool? QualifiedLead
{
    get{return qualifiedLead;}
    set{qualifiedLead = value;}
}
}

```

```

public string LeadSource
{
    get{return leadSource;}
    set{leadSource = value;}
}
public DateTime QualificationDate
{
    get{return qualificationDate;}
    set{qualificationDate = value;}
}
public string WorkflowApprovalStatus
{
    get{return workflowApprovalStatus;}
    set{workflowApprovalStatus = value;}
}
public string WorkflowPriority
{
    get{return workflowPriority;}
    set{workflowPriority = value;}
}
public string ApprovedSalespersonID
{
    get{return approvedSalespersonID;}
    set{approvedSalespersonID = value;}
}
public DateTime ModifiedDate
{
    get{return modifiedDate;}
    set{modifiedDate = value;}
}

```

7. Add a class constructor.

Create a class constructor. The constructor should initialize any data fields that cannot be null.

The following sample code shows the constructor for the Lead class. Notice how the Key and PotentialRevenue properties are initialized.

```

// Constructor
public Lead()
{
    Key = new LeadKey();

    PotentialRevenue = MoneyAmount.GetInstance(0m, "USD", 2);
}

```

8. Save the class.

Add a class to the project that defines a document key.

Create a key class

Web Services for Microsoft Dynamics GP uses key objects to identify specific documents. To use your documents with web services, you must create a key class for your document type class.

To add a key class to the project, complete the following steps:

1. Create a class.

Add a public class to the namespace and file of the document type class you created.

The following sample code adds a LeadKey class:

```
namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    [Serializable]
    [GuidAttribute("2852BB26-3BA8-4663-9613-033327D7F3C2")]
    public class Lead : BusinessObject
    {
        ...
    }

    public class LeadKey
    {
    }
}
```

2. Add the serializable class attribute.

Since the key will be used with web services, add the Serializable attribute.

```
[Serializable]
public class LeadKey
{
}
```

3. Inherit from the ReferenceKey class.

To allow the key to be used with the Dynamics GP web service, it must inherit from the Microsoft.Dynamics.GP.ReferenceKey class.

```
[Serializable]
public class LeadKey : ReferenceKey
{
}
```

4. Add a constructor that defines your document key.

The Reference Key class uses a properties list to store keys as Name and Value pairs. To specify a document, you create property objects that contain the data that identifies a document.

The following sample code shows a constructor for the LeadKey class. Notice how a Property object is created, initialized, and added to the ReferenceKey **properties** list.

```
[Serializable]
public class LeadKey : ReferenceKey
{
    // Constructor
    public LeadKey()
    {
        Property id = String.Empty;
        properties.Add(id);
    }
}
```

5. Add a class property that set or retrieves a key value.

Implement a class property that sets or retrieves the value of each Property object in the properties list.

The following sample code adds an Id property that sets or retrieves the value of the first element in the **properties** list.

```
[Serializable]
public class LeadKey : ReferenceKey
{
    // Constructor
    public LeadKey()
    {
        Property id = String.Empty;
        properties.Add(id);
    }

    // Id Property that specifies a sales lead document
    public string Id {
        get { return properties[0]; }
        set { properties[0] = value; }
    }
}
```

6. Add a GetInstance method.

Add a static GetInstance method that returns individual instances of your key class.

```
[Serializable]
public class LeadKey : ReferenceKey
{
    // Constructor
    public LeadKey()
    {
        Property id = String.Empty;
        properties.Add(id);
    }
}
```

```

// Id Property that specifies a sales lead document
public string Id {
    get { return properties[0]; }
    set { properties[0] = value; }
}

public static LeadKey GetInstance()
{
    return new LeadKey();
}
}

```

7. Save the class.

Enumerations

If your document type uses any custom enumerations, you must define those enumerations for the web service.

The following code sample adds the LeadCategory enumeration to match what is defined in the Lead Maintenance form. Notice how the enumeration also requires the Serializable attribute. In addition, each enumeration member requires an XmlEnum attribute that specifies an XML value to use with the web service. The value of RealEstate is set to one, this causes the enumeration values to match what is defined in the Lead Maintenance form.

```

[Serializable]
public enum LeadCategory
{
    [XmlEnum("RealEstate")]
    RealEstate = 1,
    [XmlEnum("Wholesale")]
    Wholesale,
    [XmlEnum("Retail")]
    Retail,
    [XmlEnum("Contractor")]
    Contractor,
    [XmlEnum("Educational")]
    Educational,
    [XmlEnum("Media")]
    Media,
    [XmlEnum("Software")]
    Software,
    [XmlEnum("Restaurant")]
    Restaurant
}

```

An enumeration can be added in any of the project files but must be in the same namespace as your document type class.

Add a class to the project that describes a document summary.

Create a summary class

To retrieve the most critical data for a document, Dynamics GP web services uses a Summary class. The class contains a limited number of data fields. The fields are specifically selected to provide a reviewer critical information about a document.

To add a Summary class to your project, complete the following steps:

1. Create a class.

Open the Visual Studio Project menu and choose Add Class. Select Class for the list of Templates in the Add New Item window. Enter a name for your class and click Add. Visual Studio adds a class to your namespace.

The following sample code shows a the LeadSummary class from the sample application:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    // The lead summary class contains critical data values from a lead
    // The class allows lists of data summary objects to be retrieved
    public class LeadSummary
    {

    }
}
```

2. Add namespace references.

Add using statements to provide convenient access to the classes and methods needed for this class. Include the following:

- Microsoft.Dynamics.Common

3. Add the serializable class attribute.

Add a class attribute specifying the class is serializable.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    // The lead summary class contains critical data values from a lead
    // The class allows lists of data summary objects to be retrieved
    [Serializable]
    public class LeadSummary
    {

    }
}
```


4. Inherit from the BusinessObject class.

All web services summary classes inherit from the `Microsoft.Dynamics.Common.BusinessObject` class.

The following sample code adds inheritance to the `LeadSummary` class:

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Dynamics.Common;

namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    // The lead summary class contains key data values from a lead
    // The class allows lists of data summary objects to be retrieved
    [Serializable]
    public class LeadSummary : BusinessObject
    {

    }
}
```

5. Add the class data members.

Add a private data field for each of the data fields you selected from your back-office document that you want included in the summary.

The following sample code defines the data fields for a lead summary object:

```
[Serializable]
public class LeadSummary : BusinessObject
{
    private LeadKey key;
    private string name;
    private string salespersonID;
    private LeadCategory leadCategory;
    private bool qualifiedLead;
    private string leadSource;
    private DateTime modifiedDate;
}
```

6. Add a class property for each data member.

Add a class property that allows you to set or retrieve the value of each data field.

```
[Serializable]
public class LeadSummary : BusinessObject
{
    private LeadKey key;
    private string name;
    private string salespersonID;
    private LeadCategory leadCategory;
    private bool qualifiedLead;
    private string leadSource;
    private DateTime modifiedDate;
```

```

public LeadKey Key
{
    get{return key;}
    set{key = value;}
}
public string Name
{
    get{return name;}
    set{name = value;}
}
public string SalespersonID
{
    get{return salespersonID;}
    set{salespersonID = value;}
}
public LeadCategory LeadCategory
{
    get{return leadCategory;}
    set{leadCategory = value;}
}
public bool QualifiedLead
{
    get{return qualifiedLead;}
    set{qualifiedLead = value;}
}
public string LeadSource
{
    get{return leadSource;}
    set{leadSource = value;}
}
public DateTime ModifiedDate
{
    get{return modifiedDate;}
    set{modifiedDate = value;}
}
}

```

7. Add a constructor.

Supply a constructor that initializes data members that cannot be null.

The following sample code initializes the Key property of the LeadSummary class. Notice how the constructor creates an empty LeadKey to initialize the LeadSummary's Key property.

```

// Constructor
// Initialize the Key object member
public LeadSummary()
{
    Key = new LeadKey();
    Key.Id = "";
}

```

8. Add a GetInstance method.

Web Services for Microsoft Dynamics GP requires a static GetInstance method that instantiates your summary object.

The following sample code adds a GetInstance method for the LeadSummary class.

```
// Instantiates a LeadSummary object
public static LeadSummary GetInstance()
{
    return new LeadSummary();
}
```

Create a list class

Add a List class to the project.

In Web Services for Microsoft Dynamics GP, the web service GetList method returns a list of summary objects that match a specified criteria. To ensure your web service's GetList method returns your summary documents, create a class that represents a list of your summary documents.

Add the class to the file and namespace where you created your summary class. Web services name each list class as:

`ArrayOf<type name>`

Where *type name* represents the class name of your summary class.

Your List class must also inherit from the System.Collections.Generic.List<> class. When you inherit from the List<> class, you must specify the type of the list.

```
// Class that encapsulates the Lead List return type
public class ArrayOfLeadSummary : List<LeadSummary>
```

The following sample code creates an ArrayOfLeadSummary class. Notice how base class declaration specifies the LeadSummary type. Since the list class uses the base class to perform its actions, it requires no additional fields, properties, or methods.

```
// Class that encapsulates the Lead List return type
public class ArrayOfLeadSummary : List<LeadSummary>
{
    public ArrayOfLeadSummary()
    {
    }
}
```

Add a criteria class to the project that enables data queries.

Create a criteria class

To enable GetList methods, you supply a criteria object that allows one or more summary objects to be retrieved. The properties of the criteria class allow you to specify ranges that will be used when documents are returned from the database. The web service GetList method returns summary objects for each document that meets the specified criteria.



The lead sample application does not implement the GetList method for leads.

1. Create a class.

From the Visual Studio Project menu, choose Add Class. Select Class from the list of Templates in the Add New Item window. Enter a name for your class and click Add. Visual Studio adds the class to your namespace.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    public class LeadCriteria
    {
    }
}
```

2. Add namespace references.

Add **using** statements to provide convenient access to the classes and methods needed for this class. Include the following:

- Microsoft.Dynamics.Common

3. Add the serializable class attribute.

Add a class attribute that specifies the class is serializable.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Dynamics.Common;

namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    [Serializable]
    public class LeadCriteria
    {
    }
}
```

4. Inherit from the Criteria class.

All web services criteria classes inherit from the `Microsoft.Dynamics.Common.Criteria` class. The following code sample shows a criteria class for leads.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Dynamics.Common;

namespace Microsoft.Dynamics.GP.Samples.SalesLeads
{
    [Serializable]
    public class LeadCriteria : Criteria
    {
    }
}
```

5. Add the class data members.

The criteria object allows a web service `GetList` web method to return a list of summary objects. To allow web method users to query your summary object data, add `Restriction` objects for each of your summary class's data members.

The following sample code adds `Restriction` objects for the `LeadCriteria` class. Notice how the type of each object corresponds to the data type of the `LeadSummary` object's fields. For more information on `Restriction` objects, refer to the *Web Service Programmers Guide*.

```
[Serializable]
public class LeadCriteria : Criteria
{
    private LikeRestriction<string> id;
    private LikeRestriction<string> name;
    private LikeRestriction<string> salespersonID;
    private LikeRestriction<LeadCategory?> leadCategory;
    private Restriction<bool?> qualifiedLead;
    private LikeRestriction<string> leadSource;
    private BetweenRestriction<DateTime?> modifiedDate;
}
```

6. Add a class property for each data member.

Add class properties that set or retrieve the value of each data field.

```
[Serializable]
public class LeadCriteria : Criteria
{
    private LikeRestriction<string> id;
    private LikeRestriction<string> name;
    private LikeRestriction<string> salespersonID;
    private LikeRestriction<LeadCategory?> leadCategory;
    private Restriction<bool?> qualifiedLead;
    private LikeRestriction<string> leadSource;
    private BetweenRestriction<DateTime?> modifiedDate;
```

```

public LikeRestriction<string> Id
{
    get{return id;}
    set{id = value;}
}
public LikeRestriction<string> Name
{
    get{return name;}
    set{name = value;}
}
public LikeRestriction<string> SalespersonID
{
    get{return salespersonID;}
    set{salespersonID = value;}
}
public LikeRestriction<LeadCategory?> LeadCategory
{
    get{return leadCategory;}
    set{leadCategory = value;}
}
public Restriction<bool?> QualifiedLead
{
    get{return qualifiedLead;}
    set{qualifiedLead = value;}
}
public LikeRestriction<string> LeadSource
{
    get{return leadSource;}
    set{leadSource = value;}
}
public BetweenRestriction<DateTime?> ModifiedDate
{
    get{return modifiedDate;}
    set{modifiedDate = value;}
}
}
}

```

7. Add a constructor.

Use the constructor to perform any initialization required for your class.

```

// Constructor
public LeadCriteria()
{
}

```

8. Override the BuildArrays method.

The Criteria base class requires you to override the BuildArrays method to specify the columns from the database table that correspond to your private data fields. You also build two arrays and add all your private data fields to each array.

The following sample code shows this for the lead criteria.

```
// Implements the required method from the Criteria base class.
// The class specifies the columns used to query summary data from the
// specified table. It specifies the restrictions that are supported by
// the criteria object.
protected override void BuildArrays()
{
    columns = new string[] { "LeadID", "LeadName", "SLPRSUID",
        "LeadBusinessCategory", "QualifiedLead", "LeadSource",
        "DEX_ROW_TS" };

    restrictions = new Restriction[] { Id, Name, SalespersonID,
        LeadCategory, QualifiedLead, LeadSource, ModifiedDate };

    convertToUpperCaseRestriction = new Restriction[] { Id, Name,
        SalespersonID, LeadCategory, QualifiedLead, LeadSource,
        ModifiedDate };
}
```

9. Override the GetWhereClause method.

You must also override the base class GetWhereClause method to perform adjustments that allow the GetList method to query data from the database.

The following sample code changes the value of the QualifiedLead property to use the boolean values stored in the IG001 table.

```
// Generates the Where clause for a query
public override string GetWhereClause()
{
    StringBuilder whereClause = new StringBuilder(base.GetWhereClause());

    // The QualifiedLead specifier is a bool field but uses the
    // values 1 and 2 instead of 0 and 1. The query values need to be
    // updated to reflect this difference in the data.
    if (QualifiedLead != null)
    {
        if (QualifiedLead.EqualValue == true)
        {
            whereClause.Replace("QualifiedLead = 1", "QualifiedLead = 2");
            whereClause.Replace("QualifiedLead = 0", "QualifiedLead = 1");
        }
        if (QualifiedLead.EqualValue == false)
        {
            whereClause.Replace("QualifiedLead = 0", "QualifiedLead = 1");
            whereClause.Replace("QualifiedLead <> 1",
                "QualifiedLead <> 2");
        }
    }
}
```

```
        return whereClause.ToString();
    }
}
```

Build the document assembly.

Build the document assembly

Open the Visual Studio Build menu, and Choose Build. Visual Studio builds your document assembly and places it in the project's "\bin\debug" folder.

Creating a web service

Create a new web service for your business document.

To use the Dynamics GP web service to retrieve new types of back-office documents, add a new web service that implements a GetByKey web method for your back-office document. Web Services for Microsoft Dynamics GP provides a framework you will use to add your new web service and web method.

Update eConnect

Add your document to eConnect.

The Dynamics GP web service relies on eConnect to retrieve data from a Dynamics GP database. eConnect is a collection of tools, components, and interfaces that allow applications to programmatically interact with Microsoft Dynamics GP.

eConnect provides the Transaction Requester to retrieve data for specified types. To retrieve a new types of data, you must define the data to return. eConnect supplies a table named eConnect_Out_Setup in each company database where you define the data for your type.

The DOCTYPE column of the eConnect_Out_Setup table contains the name of the document type. To work with the Dynamics GP web service, the entry in the DOCTYPE column must start with **WS** followed by the name of the class you created to define your document type.

The Transaction Requester requires you to specify the table to retrieve data from. It also requires you to specify the index column or columns for the table. You can perform a SQL join to include data from other tables. The transaction requester also requires you to specify the data columns you want to retrieve data from.

The following table describes the columns of the eConnect_Out_Setup table used with lead documents. The eConnect_Out_Setup table contains a number of additional columns that allow you to support more complex documents. For more information on eConnect and the Transaction Requester, see the Microsoft Dynamics GP eConnect Programmer's Guide

Column name	Description
DOCTYPE	Identifies the service.
TABLERNAME	Name of the table in the SQL Server database
ALIAS	Alias name for DOCTYPE which is used in the output XML document.
MAIN	Specifies whether this record is associated with the primary table or a child table.
INDEX1-15	Specifies the column name of the primary index for the specified table.
INDEXCNT	Specifies the number of index columns used with the table's index.
DATA1-180	Specifies the number of data columns that are used by the document.
DATA1-180	Specifies the names of the data columns that you want to appear in the XML document.

The following SQL script sample updates the eConnect_Out_Setup to retrieve lead data from the sample application. Notice how it populates the DOCTYPE with WSLead and it populates INDEX1 with LeadID column name. Also notice how DATA1-23 specify the remaining column names from the IG001 table.

```
INSERT INTO dbo.eConnect_Out_Setup
(DOCTYPE, TABLENAME, ALIAS, MAIN, INDEX1, INDEXCNT, DATAcnt, DATA1, DATA2, DATA3,
DATA4, DATA5, DATA6, DATA7, DATA8, DATA9, DATA10, DATA11, DATA12, DATA13, DATA14,
DATA15, DATA16, DATA17, DATA18, DATA19, DATA20, DATA21, DATA22, DATA23)
VALUES ('WSLead', 'IG001', 'Lead', 1, 'LeadID', 1, 23, 'LeadName', 'SLPRSNUM', 'CITY',
'STATE', 'ZIP', 'ADDRESS1', 'ADDRESS2', 'PHONE1', 'PHONE2', 'FAX',
'LeadBusinessCategory', 'COUNTRY', 'CONTACT', 'PotentialRevenue',
'QualifiedLead', 'LeadSource', 'QualificationDate', 'LeadPassword', 'NOTEINDEX',
'Workflow_Approval_Status', 'Workflow_Priority', 'Approved_Salesperson_ID',
'DEX_ROW_TS')
```

Create an XSLT file

Create an XSLT file for your business document.

The Dynamics GP web service platform will use the eConnect Transaction Requester to retrieve the data for your document. When the data is retrieved, it is represented as XML. The XML data nodes are identified by column names from the Dynamics GP data table.

When you created your document assembly, you added data fields and properties that represent the data from your back-office document. Since the web service returns an instance of your document object, you need to transform the XML returned by eConnect into an instance of your business document type.

The Dynamics GP web service platform performs the transform, but requires an XSLT file that maps the eConnect fields to your document type's properties.

To create an XSLT file, complete the following steps.

1. Create an XSLT file.

Start Visual Studio. From the File menu choose New >> File. In the New File window, select XSLT File and click Open. Visual Studio creates a file that contains the following:

```
<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <body>
        <!--
          This is an XSLT template file. Fill in this area with the
          XSL elements which will transform your XML to XHTML.
        -->
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

2. Edit the XSLT in the file.

Remove the existing `xsl:template` node. Your XSLT should match the following sample.

```
<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

</xsl:stylesheet>
```

3. Add the Dynamics GP namespaces.

Add the namespaces to the `xsl:stylesheet` node that Dynamics GP web services use to transform eConnect data.

The following code sample adds these namespaces to the `xsl:stylesheet` node that Visual Studio created.

```
<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ms="urn:schemas-microsoft-com:xslt"
  xmlns:gpxslt="uri://GreatPlainsTransformLibrary"
  xmlns:gputil="urn:Microsoft.Dynamics.GP.TransformUtilities"
  xmlns:mbs="http://schemas.microsoft.com/dynamics/2006/01"
  version="1.0">

</xsl:stylesheet>
```

4. Import the StandardLibrary.

To use the same XSLT templates and definitions that the Dynamics GP web service use, import Microsoft Dynamics GP XSLT StandardLibrary. Notice how the following XSLT sample uses an `xsl:import` node to add the library.

```
<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ms="urn:schemas-microsoft-com:xslt"
  xmlns:gpxslt="uri://GreatPlainsTransformLibrary"
  xmlns:gputil="urn:Microsoft.Dynamics.GP.TransformUtilities"
  xmlns:mbs="http://schemas.microsoft.com/dynamics/2006/01"
  version="1.0">

  <xsl:import href="Microsoft.Dynamics.GP.StandardLibrary.xslt"/>

</xsl:stylesheet>
```

5. Add the eConnect template.

Add the templates you will use with your document type. This allows your XSLT to work with the document as it is produced by the eConnect Transaction Requester. The web service framework requires you to specify the following template.

```
<xsl:template match = "root/eConnect">
  <xsl:apply-templates />
</xsl:template>
```

6. Add a document template.

Add a template that specifies the name of your specific document type.

The following sample adds a template for the Lead type used in the sample application. In the XSLT, "Lead" corresponds to the value supplied as ALIAS in the eConnect_Out_Setup table.

```
<xsl:template match = "root/eConnect">
  <xsl:apply-templates />
</xsl:template>
<xsl:template match ="Lead">

</xsl:template>
```

7. Add the document node.

Add a node to the template that specifies your document type. The following sample adds a node for a Lead document. Notice how the node is the name of the class that defines the document.

```
<xsl:template match ="Lead">
  <Lead>

  </Lead>
</xsl:template>
```

8. Add property nodes.

Add nodes to the document that specify the name of each property in your document class. The following sample adds a node for each of the properties in the Lead class. Each property is added as a child of the Lead node that was added in the previous step.

```
<xsl:template match ="Lead">
  <Lead>
    <Key>
      <Id>

      </Id>
    </Key>
    <Name>

    </Name>
    <SalespersonID>

    </SalespersonID>
    <City>
```

```
</City>
<State>

</State>
<Zip>

</Zip>
<Address1>

</Address1>
<Address2>

</Address2>
<Phone1>

</Phone1>
<Phone2>

</Phone2>
<Fax>

</Fax>
<LeadBusinessCategory>

</LeadBusinessCategory>
<Country>

</Country>
<Contact>

</Contact>

</PotentialRevenue>
<QualifiedLead>

</QualifiedLead>
<LeadSource>

</LeadSource>
<QualificationDate>

</QualificationDate>
<WorkflowApprovalStatus>

</WorkflowApprovalStatus>
<WorkflowPriority>

</WorkflowPriority>
<ApprovedSalespersonID>
```

```

        </ApprovedSalespersonID>
        <ModifiedDate>

        </ModifiedDate>
    </Lead>
</xsl:template>

```

9. Add values to each property node.

To complete the XSLT transform, populate each property node with a value retrieved by the eConnect Transaction Requester. The eConnect Transaction Requester uses the column name from the Dynamics GP source table to label each data element it retrieves. Use the column name to specify the eConnect Transaction Requester data value that populates each of your document's properties.

Add an `<xsl:value-of />` XSLT node to each property nodes you added in the previous step. In the `<xsl:value-of />` XSLT node, use the `select="column name"` attribute to specify the eConnect Transaction Requester column name. The following sample shows how to populate the Name property. LeadName is the column name from the IG001 table that stores the name of a lead.

```

<Name>
    <xsl:value-of select="LeadName" />
</Name>

```

10. Save the XSLT file.

After completing the map, save the XSLT file. Dynamics GP web service conventions require you to name the file with the following format:

```
<document type><operation>.xslt
```

For example, LeadGetByKey.xslt indicates the file contains the XSLT transform map for Lead documents and the GetByKey web method.

The following sample shows a complete XSLT transform map for the GetLeadByKey web method. This example maps columns from the IG001 table that was created for the Lead Maintenance sample application to the properties of the Lead document.

As you review the XSLT, notice the following features of the XSLT:

- The template includes two `<xsl:variable />` nodes. These nodes add the "isocode" and "decimaldigits" variables from the StandardLibrary. These variables supply values for the PotentialRevenue property. PotentialRevenue is a MoneyAmount that requires you to supply Currency and DecimalDigits values.
- The LeadBusinessCategory node uses `<xsl:choose \>` to convert the enumeration integer value found in the database to the corresponding LeadBusinessCategory enumeration string value.
- The QualifiedLead property maps the integer value retrieved from the database to the corresponding boolean value.

- The XSLT tests the date value mapped to the ModifiedDate property. The test ensures the value is not empty. The ModifiedDate is only populated if the DEX_ROW_TS value is not equal to the Microsoft Dynamics GP default date.

```

<xsl:template match ="Lead">
  <xsl:variable name="isocode">
    <xsl:value-of select="gputil:LocalCurrency(
      /root/mbs:Context/mbs:OrganizationKey/mbs:Id)"/>
  </xsl:variable>
  <xsl:variable name="decimaldigits"
    select="gputil:CurrencyDecimalDigits($isocode)"/>
<xsl:template match ="Lead">
  <Lead>
    <Key>
      <Id>
        <xsl:value-of select="LeadID"/>
      </Id>
    </Key>
    <Name>
      <xsl:value-of select="LeadName"/>
    </Name>
    <SalespersonID>
      <xsl:value-of select="SLPRSNID"/>
    </SalespersonID>
    <City>
      <xsl:value-of select="CITY"/>
    </City>
    <State>
      <xsl:value-of select="STATE"/>
    </State>
    <Zip>
      <xsl:value-of select="ZIP"/>
    </Zip>
    <Address1>
      <xsl:value-of select="ADDRESS1"/>
    </Address1>
    <Address2>
      <xsl:value-of select="ADDRESS2"/>
    </Address2>
    <Phone1>
      <xsl:value-of select="PHONE1"/>
    </Phone1>
    <Phone2>
      <xsl:value-of select="PHONE2"/>
    </Phone2>
    <Fax>
      <xsl:value-of select="FAX"/>
    </Fax>
    <LeadBusinessCategory>
      <xsl:choose>
        <xsl:when test="LeadBusinessCategory=1">RealEstate</xsl:when>
        <xsl:when test="LeadBusinessCategory=2">Wholesale</xsl:when>
        <xsl:when test="LeadBusinessCategory=3">Retail</xsl:when>
        <xsl:when test="LeadBusinessCategory=4">Contractor</xsl:when>
        <xsl:when test="LeadBusinessCategory=5">Educational</xsl:when>
        <xsl:when test="LeadBusinessCategory=6">Media</xsl:when>
      </xsl:choose>
    </LeadBusinessCategory>
  </Lead>
</xsl:template>

```

```

        <xsl:when test="LeadBusinessCategory=7">Software</xsl:when>
        <xsl:when test="LeadBusinessCategory=8">Restaurant</xsl:when>
    </xsl:choose>
</LeadBusinessCategory>
<Country>
    <xsl:value-of select="COUNTRY" />
</Country>
<Contact>
    <xsl:value-of select="CONTACT" />
</Contact>
<PotentialRevenue>
    <Currency>
        <xsl:value-of select="$isocode" />
    </Currency>
    <Value>
        <xsl:value-of select="PotentialRevenue" />
    </Value>
    <DecimalDigits>
        <xsl:value-of select="$decimaldigits" />
    </DecimalDigits>
</PotentialRevenue>
<QualifiedLead>
    <xsl:choose>
        <xsl:when test="QualifiedLead=1">0</xsl:when>
        <xsl:when test="QualifiedLead=2">1</xsl:when>
    </xsl:choose>
</QualifiedLead>
<LeadSource>
    <xsl:value-of select="LeadSource" />
</LeadSource>
<QualificationDate>
    <xsl:value-of select="QualificationDate" />
</QualificationDate>
<WorkflowApprovalStatus>
    <xsl:value-of select="Workflow_Approval_Status" />
</WorkflowApprovalStatus>
<WorkflowPriority>
    <xsl:value-of select="Workflow_Priority" />
</WorkflowPriority>
<ApprovedSalespersonID>
    <xsl:value-of select="Approved_Salesperson_ID" />
</ApprovedSalespersonID>
<xsl:if test="gputil:IsNotGreatPlainsDefaultDate(DEX_ROW_TS)">
    <ModifiedDate>
        <xsl:value-of select="DEX_ROW_TS" />
    </ModifiedDate>
</xsl:if>
</Lead>
</xsl:template>

```

Create a web service

To retrieve a lead document, create a new web service that implements the GetLeadByKey web method. The web method will use the previous eConnect and XSLT changes to retrieve a lead document.

Use Visual Studio to create a web service application.

1. Create a new project.

Open Visual Studio. From the File menu, select New >> Project. In the New Project window, click Visual C# from the Project types list. Select Web in the Project types list. Select ASP.NET Web Service Application from the Templates list. Enter a Name for your web service, specify a Location, and then click OK.

The following sample code shows the default web service that Visual Studio creates:

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

/// <summary>
/// Summary description for Service1
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    [WebMethod]
    public string HelloWorld()
    {
        return "Hello World";
    }
}
```

2. Add references to the project.

Open the Project menu and choose Add References. The Add References window opens. Click the Browse tab and browse to the Microsoft Dynamics GP web service "bin" folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\bin

Select the following assemblies and click OK:

- Microsoft.Dynamics.Common.dll
- Microsoft.Dynamics.Common.Types.dll
- Microsoft.Dynamics.GP.BusinessLogic.dll

Also add a reference to your document assembly. The following is the document assembly created for lead documents.

- Microsoft.Dynamics.GP.Samples.SalesLeads.dll

3. Add namespace references.

From Solution Explorer, right click the .asmx file and select View Code. Add **using** statements to provide convenient access to the classes and methods needed for this class. Include the following:

- Microsoft.Dynamics.Common
- Microsoft.Dynamics.GP

Add an additional **using** statement for your document type namespace. The following example is the namespace of the lead document assembly used by the sample application.

- Microsoft.Dynamics.GP.Samples.SalesLeads

4. Edit the web service.

Update the WebService and WebServiceBinding attributes to reflect your namespace and profile information. Delete the HelloWorld web method. Your code should resemble the following:

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using Microsoft.Dynamics.Common;
using Microsoft.Dynamics.GP;
using Microsoft.Dynamics.GP.Samples.SalesLeads;

/// <summary>
/// Summary description for Service1
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
}
}
```

5. Add a web method.

Add a new public web method to the web service. For workflow you need a single GetByKey web method. The method returns a single instance of a back-office document. Add the WebMethod attribute to the method.

The method should accept two parameters:

Parameter	Description
key	An instance of an object that uniquely identifies a business document.
context	A Web Services for Microsoft Dynamics GP Context object that has properties that specify how to execute a web service request.

The following sample code adds a `GetByKey` web method for leads. Notice how it takes a `LeadKey` object to specify the lead to be retrieved.

```
[WebMethod]
public Lead GetLeadByKey(LeadKey leadKey, Context context)
{
}
}
```

6. Use the business service to retrieve the document.

To use the eConnect and XSLT pieces you added to the Dynamics GP web service platform, create an instance of the `GreatPlainsBusinessService`. Use the business service `GetByKey` method to retrieve a specific document. You must provide the three parameters to the `GetByKey` method:

Parameter	Description
key	Use the key parameter that was sent to the web method.
context	Use the context object that was sent to the web method.
type	Specify the type of the object. This will be the name of the class you created to define your business document.

The business service `GetByKey` method will return a generic business object. Cast the object to the document type your web method returns. Return the object from your web method.

If an error occurs, catch the error, convert it to a SOAP exception and return the exception to the caller of the web method.

The following sample code uses the `GreatPlainsBusinessService` to retrieve a `Lead`.

```
[WebMethod]
public Lead GetLeadByKey(LeadKey leadKey, Context context)
{
    try
    {
        // Instantiate the GreatPlainsBusinessService. Use the service's
        // GetByKey method to retrieve a specified sales lead.
        GreatPlainsBusinessService service =
            GreatPlainsBusinessService.GetInstance();
        Lead salesLead =
            (Lead)service.GetByKey(leadKey, context, typeof(Lead));
        return salesLead;
    }
    catch (BusinessException err)
    {
        throw ExceptionManager.ConvertBusinessToSoapException(err);
    }
}
```

7. Build the web service

From the Build menu, choose Build. Visual Studio builds the web service .asmx file and its code-behind assembly. Visual Studio places the assembly in the "bin" folder of the project.

Securing the web service

Create an application that updates the Dynamics Security Service.

When you add your web service to the Web Services for Microsoft Dynamics GP platform, security for your web service will be managed by the Dynamics Security Service.

To allow users to access your web service you must add operations and tasks to the Dynamics Security Service that provide access to specified security roles. To add this security information, create an application that uses the Dynamics Security Service to create the security tasks and operations. You will run this application when you install your web service on the Web Services for Microsoft Dynamics GP server.

The “security helper” application performs several tasks. The application creates security operations and tasks in the Dynamics Security Service. It assigns the tasks and operations to the Superuser role. If you need to remove your web service, the application should be able to remove your security operations and tasks from the Dynamics Security Service.

Create a Visual Studio project

The application requires you to create a new Visual Studio project. To create the project, complete the following steps:

1. Create a new project.

Open Visual Studio. From the File menu, select File >> New >> Project. In the New Project window, choose Visual C# from the Project types list. In Templates, select Console Application from the list of Visual Studio installed templates.

Enter a name for your project. Review the Location and Solution Name and click OK.

2. Delete the Class1.cs file from the project.

Visual Studio creates a project that contains a file named Class1.cs. From the View menu, choose Solution Explorer. In Solution Explorer, delete Class1.cs from your project.

3. Add references to the project.

From the Project menu, choose Add References. The Add References window opens. Click the .NET tab. Select the following assemblies and click OK:

- System.Configuration
- System.EnterpriseServices
- System.Web.Services

From the Project menu, choose Add References. The Add References window opens. Click the Browse tab and navigate to the Dynamics GP web services “bin” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

Select the following assembly and click OK:

- Microsoft.Dynamics.Security.dll

4. Create the security context and key objects.

A “security helper” application requires a security context object and several types of object keys. For example, the “security helper” application for the Sales Lead sample web service uses the following objects.

```
private SecurityContext securityContext;
private ApplicationKey appKey;
private OperationKey opKey;
private OperationKey getByKeyOpKey;
private OperationKey getListOpKey;
private TaskKey taskKey;
private RoleKey roleKey;
```

To populate the security context and object keys, add an initialization method to your application. The following code examples show how to use an initialization method to instantiate the required objects.

Create a security context object.

Create a SecurityContext object and specify the Web Services for Microsoft Dynamics GP application.



To specify Web Services for Microsoft Dynamics GP, use this same application key GUID in your security context object.

```
securityContext = new SecurityContext();
appKey = new ApplicationKey();
appKey.Id = "25cc1a21-2cc4-4b13-a1c8-eeal86fb688a";
securityContext.ApplicationKey = appKey;
```

Create the operation keys.

Create a new operation key for each of your security operations. The following code example creates a basic operation key. The examples includes a GUID that was generated to uniquely identify this operation.

```
opKey = new OperationKey();
opKey.Id = "E5EA3889-BFFC-4817-AC06-E818AAFD0B13";
```

Create an operation key for each of your web service web methods. The Id of each operation key must include the GUID value from the GUIDAttribute of your web service document class. To uniquely identify each operation, append “GetByKey” and “GetList” to the GUID value.



Web services requires the key Ids to include “GetByKey” or “GetList”. If you do not include these in your key Id, you will not be able to access your data.

The following code example creates OperationKeys for the Sales Lead web service. Notice how the operation key Id uses the GUID from the lead document class followed by the name of the operation.

```
// Create a key for a Get by Key operation
// Use the GUID attribute of the type to identify the operation
getByKeyOpKey = new OperationKey();
getByKeyOpKey.Id = "2852BB26-3BA8-4663-9613-033327D7F3C2GetByKey";

// Create a key for the Get List operation
// Use the GUID attribute of the type to identify the operation
getListOpKey = new OperationKey();
getListOpKey.Id = "2852BB26-3BA8-4663-9613-033327D7F3C2GetList";
```

Create a task key.

A task in the Dynamics Security Service is an object that contains other security objects. A task object allows you to bundle related operations into a single object that can be assigned to a security role.

The following code example creates a key for a task. The example includes a GUID that was generated to uniquely identify this task.

```
taskKey = new TaskKey();
taskKey.Id = "A33871C6-8393-4040-803D-FA1EF7306136";
```



The example of the subscription helper application for sales leads uses the Id of this task to add sales lead web service access to the Dynamics Security Service workflow administrator role.

Create a role key.

Create a role key that specifies the Dynamics Security Service role that will be assigned the new operations and tasks.

The following code example creates a role key that specifies the Dynamics Security Service Superuser. Use this GUID whenever you need to identify the Superuser role.

```
roleKey = new RoleKey();
// The following GUID value identifies the Superuser role
roleKey.Id = "e18b321a-9548-48fb-b75a-dee0a618ddaa";
```

5. Create the security operations.

Instantiate an `OperationService` object. `OperationService` provides an interface with the Dynamics Security Service that allows you to create operations.

```
// Instantiate the OperationService object that enables you to work with /
// the Microsoft Dynamics Security service.
OperationService opService = OperationService.GetInstance();
```

Instantiate an `Operation` object for each operation you want to add. The following code example creates an array of three `Operation` objects.

```
// Create a security operation object for each operation you
// want to enable.
Microsoft.Dynamics.Security.Operation[] leadOps = new Operation[3];
```

Populate the `Operation` objects with the keys, a name, and a brief description.

```
// Populate a security object that enables a query operation for
// sales leads.
leadOps[0] = Microsoft.Dynamics.Security.Operation.GetInstance();
leadOps[0].Key = opKey;
leadOps[0].Name = "Query Sales Leads";
leadOps[0].Description = "Privilege to query sales leads";

// Populate a security object that enables a GetByKey operation for
// sales leads.
leadOps[1] = Microsoft.Dynamics.Security.Operation.GetInstance();
leadOps[1].Key = getByKeyOpKey;
leadOps[1].Name = "Get Sales Lead";
leadOps[1].Description = "Privilege to view a sales lead";
```

```
// Populate a security object that enables a GetList operation.
leadOps[2] = Microsoft.Dynamics.Security.Operation.GetInstance();
leadOps[2].Key = getListOpKey;
leadOps[2].Name = "Get Leads List";
leadOps[2].Description = "Privilege to view sales leads";
```

Use the `CreateOperation` method of the `OperationService` to add your new operations to the Dynamics Security Service. If the specified operation already exists, use the `UpdateOperation` method to update that operation.

```
foreach (Operation op in leadOps)
{
    try
    {
        // If the operation exists, update the existing operation.
        opService.UpdateOperation(securityContext, op);
    }
    catch (NonExistentSecurityObjectException)
    {
        // If the operation does not exist, add the operation to the
        // security service.
        opService.CreateOperation(securityContext, op);
    }
}
```

6. Create a security task object.

Create a security task object that includes your security operations. Instantiate a task object, and then populate that object with the task key, a name, and a brief description. To specify the operations that the task includes, add the security operation objects to the operations list of the task object. To add the task to the Dynamics Security Service, use the `CreateTask` method of the `TaskService` object.

The following code example creates the “View Sales Leads” security task and adds it to the Dynamics Security Service. Notice how the three previously created security operations are added to this task object.

```
// Instantiate the TaskService object that enables you to work with the
// Microsoft Dynamics Security service.
TaskService taskService = TaskService.GetInstance();

// Create a task object for sales leads.
// Populate the task object with the new sales lead security objects.
object.Task task = Task.GetInstance();
task.Key = taskKey;
task.Name = "View Sales Leads";
task.Description = "View Sales Leads";
task.Operations.Add(opKey);
task.Operations.Add(getByKeyOpKey);
task.Operations.Add(getListOpKey);
```

```

// Create or update the task.
try
{
    // If the task already exists, update the existing task object.
    taskService.UpdateTask(securityContext, task);
}
catch (NonExistentSecurityObjectException)
{
    // If the task does not exist, add the task object to
    // security service.
    taskService.CreateTask(securityContext, task);
}

```

7. Add the security operations to the Superuser role.

To ensure visibility and access to the new security operations, add each operation to the Superuser role. Create a Role object and add the new security operations to the operations list of the Role object. Use the UpdateRole method of the RoleService object to add the operations to the Superuser role.

```

// Instantiate the RoleService object that enables you to work with the
// Microsoft Dynamics Security service.
RoleService roleService = RoleService.GetInstance();

// Instantiate a role object that represents the web service security
// Superuser role.
Role role = roleService.GetRole(securityContext, roleKey);

// Only add a security operation object to the role if that object is not
// already assigned to that role.
bool newOpAdded = false;

foreach (Operation newOp in leadOps)
{
    if (role.Operations.BinarySearch(newOp.Key) < 0)
    {
        role.Operations.Add(newOp.Key);
        newOpAdded = true;
    }
}

// Only update the security service when an operation is
// added to the role object.
if (newOpAdded == true)
{
    roleService.UpdateRole(securityContext, role);
}

```

8. Create an application configuration file.

To update the Dynamics Security Service, your “security helper” application requires configuration parameters from your Web Service for Microsoft Dynamics GP installation. One way to retrieve the required parameters is to copy the contents of an existing web service configuration file into the configuration file for the “security helper” application.

For example, the following Web Services for Microsoft Dynamics GP installation configuration file contains the required parameters:

Microsoft.Dynamics.GP.InstallSystemSecurityMetadata.exe.config

The configuration file is typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices

The following code example copies the contents of the web service installation configuration file into the application configuration file of the “security helper” application.



To run the “security helper” application, the executable file and the application configuration file must be in the same folder as the specified configuration file.

```
// Copy the contents of the
// Microsoft.Dynamics.GP.InstallSystemSecurityMetadata.exe.config file
// to this application's configuration file.
ExeConfigurationFileMap fileMap = new ExeConfigurationFileMap();
fileMap.ExeConfigFilename =
    "Microsoft.Dynamics.GP.InstallSystemSecurityMetadata.exe.config";
Configuration config =
    ConfigurationManager.OpenMappedExeConfiguration(fileMap,
        ConfigurationUserLevel.None);
config.SaveAs("Microsoft.Dynamics.GP.Samples.InstallSalesLeadSecurityMeta
data.exe.config");
```

9. Removing operations from the Dynamics Security Service.

To support an uninstall of your web service, your “security helper” application should remove your security tasks and operations from the Dynamics Security Service. To remove your Dynamics Security Service changes, you must supply the same security context, operation key, task key, and role key GUID values that you used to add them to the Dynamics Security Service.

To remove the tasks and operations, reverse the order in which they were originally installed. The following sample code removes the security operations and tasks that were added for the Sales Lead sample web service.

```
// Remove the specified security operations from the Superuser role.
// Create an array of the operation key objects used by the sales lead
// web service.
OperationKey[] opKeys = new OperationKey[3];
opKeys[0] = opKey;
opKeys[1] = getByKeyOpKey;
opKeys[2] = getListOpKey;
```



```

// Instantiate the RoleService object that enables you to work with the
// Microsoft Dynamics Security service.
// Instantiate a role object that represents the web service security
// Superuser role.
RoleService roleService = RoleService.GetInstance();
Role role = roleService.GetRole(securityContext, roleKey);

// Only remove the security operation from the role if it has been
// assigned to the Superuser role.
bool opRemoved = false;

foreach (OperationKey key in opKeys)
{
    if (role.Operations.BinarySearch(key) >= 0)
    {
        role.Operations.Remove(key);
        opRemoved = true;
    }
}

// Only update the security service when an operation is
// removed from the role object.
if (opRemoved == true)
{
    roleService.UpdateRole(securityContext, role);
}

// Remove the sales lead security task.
// Instantiate a TaskService object that enables you to work with the
// Microsoft Dynamics Security service.
TaskService taskService = TaskService.GetInstance();
try
{
    // Delete the specified task
    taskService.DeleteTask(securityContext, taskKey);
}
catch (NonExistentSecurityObjectException)
{
    // If the task does not exist, no action is needed.
    // Trap the error and continue.
}

// Remove the three sales lead security operations.
// Instantiate the OperationService object that enables you to work
// with the Microsoft Dynamics Security service.
OperationService opService = OperationService.GetInstance();
foreach (OperationKey key in opKeys)
{
    try
    {
        opService.DeleteOperation(securityContext, key);
    }
}

```

```

        catch (NonExistentSecurityObjectException)
        {
            // If the operation does not exist, no action is needed.
            // Trap the error and continue.
        }
    }
}

```

10. Build the application.

From the Visual Studio Build menu, choose Build. Visual Studio builds the application and the application configuration file. Visual Studio places both files in the project “\bin\debug” folder.

Testing the web service

It is a good idea to test the web service to ensure you can use it to retrieve sales lead documents. You can test the web service with a console application that uses the GetLeadByKey web method to retrieve a specified lead document.

The following C# example tests the Sale Lead web service. The application requires you to supply a web reference to the Sales Lead web service. If the web service is working, the application retrieves the document with the Lead ID value of “1001” and displays the name of that lead.

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using WebServiceTest.SalesLeadWebService;

namespace WebServiceTest
{
    class Program
    {
        static void Main(string[] args)
        {
            CompanyKey companyKey;
            Context context;
            LeadKey leadKey;
            Lead lead;

            // Create an instance of the sales lead web service
            SampleSalesLeadWebService wsSalesLeads =
                new SampleSalesLeadWebService();

            // Be sure the default credentials are used
            wsSalesLeads.UseDefaultCredentials = true;

            // Create a context object with which to call the web service
            context = new Context();

            // Specify which company to use (sample company)
            companyKey = new CompanyKey();
            companyKey.Id = (-1);
        }
    }
}

```

```
// Set up the context object
context.OrganizationKey = (OrganizationKey)companyKey;
context.CultureName = "en-US";

// Create a sales lead key to specify the sales lead
leadKey = new LeadKey();
leadKey.Id = "1001";

// Retrieve the sales lead object
lead = wsSalesLeads.GetLeadByKey(leadKey, context);

// Display the customer name property from the sales lead object
MessageBox.Show("Customer name: " + lead.Name);
    }
}
}
```


Chapter 9: Server Workflow Assembly

The Server Workflow Assembly is a Microsoft .NET assembly that creates a workflow type for a document in Microsoft Dynamics GP. Information about the Server Workflow Assembly is described in the following sections:

- [Creating a Visual Studio project](#)
- [Creating a workflow type](#)
- [Adding business logic](#)
- [Signing your server workflow assembly](#)
- [Creating a document viewer](#)
- [Creating a workflow event subscription helper application](#)
- [Building the assembly and application](#)

Creating a Visual Studio project

Create a new server workflow assembly.

The Server Workflow Assembly is a Microsoft .NET assembly you create using Visual Studio. The assembly works with the Microsoft Dynamics GP workflow server to add workflow approval to your back-office document.

To create a Server Workflow Assembly, complete the following steps:

1. Create a new project.

Open Visual Studio. From the File menu, select File >> New >> Project. In the New Project window, select Visual C# from the Project types tree. In Templates, choose Class Library from the list of Visual Studio installed templates.

Enter a name for your assembly. Review the Location and Solution Name, and then click OK.

2. Delete the Class1.cs file from the project.

Visual Studio creates a default class file named Class1.cs. From the View menu, select Solution Explorer. In Solution Explorer, delete Class1.cs from your project.

If a dialog window opens asking whether to delete the file, click OK.

3. Add references to the project.

From the Project menu, select Add References. The Add References window opens. Click the .NET tab. Select the following assembly and click OK:

- System.Windows.Forms

From the Project menu, select Add References. The Add References window opens. Click the Browse tab. Navigate to the SharePoint server "ISAPI" folder, typically found in the following location:

C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions
\12\ISAPI

Select the following .dll file and click OK:

- Microsoft.SharePoint.dll

From the Project menu, select Add References. The Add References window opens. Click the Browse tab. Browse to the Microsoft Dynamics GP Workflow folder. Typically, the folder is in the following location:

C:\Program Files\Microsoft Dynamics\Workflow

Select the following .dll files and click OK:

- Microsoft.Dynamics.Common.dll
- Microsoft.Dynamics.Common.Types.dll
- Microsoft.Dynamics.GP.Formatter.dll
- Microsoft.Dynamics.GP.Workflow.dll
- Microsoft.Dynamics.Workflow.dll
- Microsoft.Dynamics.Workflow.Common.dll
- Microsoft.Dynamics.Workflow.Controls.dll

From the Project menu, select Add References. The Add References window opens. Click the Browse tab. Navigate to the SharePoint folder, typically found in this location:

C:\inetpub\wwwroot\wss\VirtualDirectories\<port#>\bin

For the port number, specify the port number of the web site where you installed Microsoft Dynamics GP Workflow. Select the following .dll file and click OK:

- Microsoft.Dynamics.GP.Workflow.Pages.dll



If you will use Web Services for Microsoft Dynamics GP with your workflow, include a reference to the Microsoft.Dynamics.GP.WebServices.Proxy.dll that is also found in this SharePoint folder.

Add a web reference to the web service.

4. Add a web reference

If you use any web service other than Web Services for Microsoft Dynamics GP to access document data, you must add a web reference to the web service. From the Visual Studio Project menu, choose Add Web Reference. The Add Web Reference Window will open. Enter the URL of your web service and click Go. Enter a name in the Web reference name field and click Add Reference.

If you set the URL Behavior property of your web reference to Dynamic, Visual Studio stores the web reference URL in an assembly configuration file. Visual Studio name the file using the following format:

<Assembly name>.dll.config

Assembly name will be the name you gave your project. For example, the lead server workflow assembly sample includes a configuration file named:

Microsoft.Dynamics.GP.Workflow.Samples.Server.dll.config

The configuration file is an XML document that contains parameters for the assembly. When you install your workflow server assembly, you must also include the assembly configuration file.

To update your assembly and use another web server, change the URL in the configuration file. When the assembly loads, it will use the URL in the configuration file to find the web service.

The following sample shows the contents of an assembly configuration file that contains the web reference URL:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="applicationSettings"
      type="System.Configuration.ApplicationSettingsGroup, System,
      Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089" >
      <section name="Properties.Settings"
        type="System.Configuration.ClientSettingsSection, System,
        Version=2.0.0.0, Culture=neutral,
        PublicKeyToken=b77a5c561934e089"
        requirePermission="false" />
      </sectionGroup>
    </configSections>
  <applicationSettings>
    <Properties.Settings>
      <setting name="Microsoft_Dynamics_GP_Workflow_Samples_
        SalesLeadWebService_SampleSalesLeadWebService"
        serializeAs="String">
        <value>http://localhost:8080/DynamicsGPWebServices
          /SampleSalesLeadWebService.asmx</value>
        </setting>
      </Properties.Settings>
    </applicationSettings>
  </configuration>
```

Creating a workflow type

Add a class to the project that defines your workflow type.

A workflow type is a .NET class that enables workflow functionality for a specified business document. The class includes properties, methods, and interfaces that enable Microsoft Dynamics GP workflow to retrieve and update individual documents.

To implement a new workflow type, use Visual Studio to add a class to your .NET assembly. Open the Project menu, choose Add Class. From the Add New Item window, select Class. Enter a name for your class and click Add. Visual Studio creates and opens a new class file. Add the following **using** statements:

```
using Microsoft.Dynamics.Workflow;
using Microsoft.Dynamics.Workflow.Controls;
using Microsoft.Dynamics.Workflow.Common;
using Microsoft.Dynamics.Common;
using Microsoft.SharePoint;
using GPWorkflow = Microsoft.Dynamics.GP.Workflow;
using SalesLeadWebService;
using System.Collections.ObjectModel;
using Microsoft.Dynamics.Workflow.Approval;
```

The sample code uses `GPWorkflow` as an alias for the `Microsoft.Dynamics.GP.Workflow` namespace. `SalesLeadWebService` refers to the web service that retrieves lead information for the sample workflow type from the Microsoft Dynamics GP SQL server.

Make the new class public:

```
public class LeadApprovalWorkflowSample
{
}
```

Add the `DynamicsWorkflow` attribute.

Add the `DynamicsWorkflow` attribute to your class. The `DynamicsWorkflow` attribute requires a GUID and a name that uniquely identify your workflow. Also, you can use the `DynamicsWorkflow` attribute to enable Microsoft Dynamics GP desktop alerts for your workflow.

To create the GUID, open the Tools menu in Visual Studio and choose Create GUID. Copy the GUID value from the Create GUID dialog window into your `DynamicsWorkflow` attribute. Since the GUID in the attribute is a string, replace the open and close braces with double quotes and convert any lowercase letter characters to uppercase.



The name you supply the `DynamicsWorkflow` attribute must be the same name you use as the `WorkflowName` property in your Client workflow assembly. If they are not identical, your workflow will not run.

Your `DynamicsWorkflow` attribute should resemble the following sample code:

```
[DynamicsWorkflow(
    "42614BD0-D17A-4284-917F-73EBD19509C3",
    "Sample Salesperson Approval Workflow",
    ApplicationSupportsDesktopAlerts = true)]
public class LeadApprovalWorkflowSample
```

Inherit from the `ApprovalWorkflow` class.

All Server workflow assemblies must inherit from the following class:

`Microsoft.Dynamics.GP.Workflow.ApprovalWorkflow`

Add the workflow interfaces.

In addition, the Server workflow class must implement the following interfaces:

- `Microsoft.Dynamics.Workflow.IDynamicsWorkflow`
- `Microsoft.Dynamics.Workflow.IDynamicsWorkflowHistory`

Your class should resemble the following sample code:

```
[DynamicsWorkflow(
    "42614BD0-D17A-4284-917F-73EBD19509C3",
    "Sample Salesperson Approval Workflow",
    ApplicationSupportsDesktopAlerts = true)]
public class LeadApprovalWorkflowSample :
    GPWorkflow.ApprovalWorkflow,
    IDynamicsWorkflow,
    IDynamicsWorkflowHistory
{
}
```

From the File menu, choose Save All, and then Exit to close Visual Studio.

Add a resources file

The next step is to add a resources file that defines values for your workflow type. To use the resources file, your workflow assembly's default namespace must be blank. Visual Studio will not allow you to set the default namespace to blank, so you must manually edit your project file.

Add a resource file to the project.

1. Edit your project file.

Open Windows Explorer and navigate to the project file for your server workflow assembly. Open your .csproj file with Notepad.

2. Delete the root namespace value.

Search the XML to locate the <RootNamespace> element. Remove the value of the element. The XML should be as follows:

```
<RootNamespace></RootNamespace>
```

3. Save your changes.

From the File menu, choose Save. Close Notepad.

Reopen your server workflow assembly project with Visual Studio. From the Project menu, choose Add New Item. In the Add New Item dialog window, select Resources File from the Templates list. Enter a name for your resources file and click Add.



The name of the resources file must match the name of your server workflow assembly. If the names do not match, your server workflow assembly will not run.

Add the required names and values to the resource file.

Open the resources file. To enable the Microsoft Dynamics Workflow site to display information about this workflow, enter the following names and values to the resource file:

Name	Value
BusinessObjectKey<GUID>	Enter a type name.
Category<GUID>	Enter Approval.
DisplayName<GUID>	Enter a brief descriptive phrase.
GroupName<GUID>	Enter a name for the group.
ShortDisplayName<GUID>	Enter a name value.
WorkflowType<GUID>	Enter a brief descriptive phrase.

Each name requires you to append your workflow GUID value. Use the GUID value from the DynamicsWorkflow attribute of your workflow server class. The following code sample demonstrates the name for the Category.

```
Category42614BD0D17A4284917F73EBD19509C3
```

Notice how the GUID matches the value from the DynamicsWorkflow attribute in the previous sample code.

Add filterable properties to the resource file.

Add resource file entries for the display names of the filterable properties of the workflow. Filterable properties allow you to specify the types of documents that require approval. In the name field, enter a value that uniquely identifies each resource value. The names of these resources do not require GUID values. In the value field, enter the text to display in the list boxes of the Configure Workflow Step page of the Microsoft Dynamics Workflow Administration site.



Adding the filterable properties to a resource file allows you to supply localized values for each property. For more information about how to use resource files to support localization, see the .NET Framework Developer's Guide.

The following example shows the resources added for leads.

Name	Value	Comment
BusinessObjectType42614BD0D17A4284917F73EBD19509C3	Lead	
Category42614BD0D17A4284917F73EBD19509C3	Approval	
DisplayName42614BD0D17A4284917F73EBD19509C3	Sales Lead	
GroupName42614BD0D17A4284917F73EBD19509C3	Lead Management	
ShortDisplayName42614BD0D17A4284917F73EBD19509C3	Leads	
WorkflowType42614BD0D17A4284917F73EBD19509C3	Salesperson approval	
Lead_Key_Id	Lead Key Id	
Lead_Salesperson	Salesperson	
Lead_PotentialRevenue	Potential Revenue	
Lead_QualificationDate	Qualified Date	
Lead_BusinessCategory_Contractor	Contractor	
Lead_BusinessCategory_Educational	Educational	
Lead_BusinessCategory_Media	Media	
Lead_BusinessCategory_RealEstate	Real Estate	
Lead_BusinessCategory_Restaurant	Restaurant	
Lead_BusinessCategory_Retail	Retail	
Lead_BusinessCategory_Software	Software	
Lead_BusinessCategory_Wholesale	Wholesale	
Bool_False	False	
Bool_True	True	
*		

Add the event handlers.

Add an event handler

The Workflow server raises an event when a specified action occurs. To associate business logic with that event, add an event handler to your workflow class. The following table lists the events that the workflow server raises:

Workflow Event	Description
BusinessObjectChanged	Occurs when the business document being managed by workflow has changed.
StatusChanged	Occurs at each point in the life of a workflow instance where the status changes on the workflow server. This may or may not match the workflow status stored by a workflow client.
StepCompleted	Occurs when a workflow step completes.
TaskCompleted	Occurs when a task is completed due to the action of an approver.
TaskCreated	Occurs when a new task is created.
TaskDeleted	Occurs when a task is deleted because approval is no longer required.
TaskRolledBack	Occurs each time actions is taken on a task by a user who does not have sufficient privileges to perform the action.
TasksEscalated	Occurs after a collection of tasks is escalated.
WorkflowCompleted	Occurs when a workflow instance successfully completes
WorkflowDeactivated	Occurs when a workflow is deactivated. This event causes all running workflow instances to be deactivated.
WorkflowError	Occurs when the Workflow server throws an exception.
WorkflowStarted	Occurs when a business document is submitted for approval.

For example, you use workflow to approve or reject changes to the salesperson assigned to a sales document. To identify when a reviewer in the workflow process approves or rejects a salesperson change, implement an event handler for the StatusChanged event. The StatusChanged event handler allows you to implement business logic when a specific status change occurs. Common workflow status changes include Submitted, Pending Approval, Pending Changes, Approved, or Rejected.

After you complete your server workflow assembly, create an event subscription that associates your event handler with the event that the workflow server raises. For information about how to subscribe to a workflow event, see [Creating a workflow event subscription helper application](#) on page 160.

The following code sample creates a `StatusChanged` event handler for lead documents. To implement the event handler, the sample adds a method named `OnStatusChanged` to the workflow class that was created earlier. The `OnStatusChanged` method responds whenever the workflow server raises a `StatusChanged` event for a lead. In this example, the event handler creates an object that contains the business logic for leads.

```
public static void OnStatusChanged(object sender,
    WorkflowStatusChangedEventArgs args)
{
    try
    {
        GPWorkflowSampleEventHandlers eventHandler = new
            GPWorkflowSampleEventHandlers();
        eventHandler.OnStatusChanged(sender, args);
    }
    catch (Exception ex)
    {
        throw ex;
    }
}
```

Add a resource handler

To simplify the retrieval of values from the resource file, add a method that retrieves a resource value from the assembly. The method should use the name of the resource to specify the value to retrieve. Use culture settings to support localization of the resources. The following code example uses the `Microsoft.Dynamics.Common.ResourceHelper` class to retrieve resource strings from the workflow sample assembly.

```
private static string GetString(string resourceId, CultureInfo cultureInfo)
{
    return Microsoft.Dynamics.Common.ResourceHelper.GetString(
        resourceId,
        Assembly.GetAssembly(typeof(LeadApprovalWorkflowSample)),
        cultureInfo
    );
}
```

Implement the `IDynamicsWorkflow` interface.

Add the `IDynamicsWorkflow` interface

To integrate with Microsoft Dynamics GP workflow, your server workflow class must implement the `IDynamicsWorkflow` interface. The `IDynamicsWorkflow` interface defines the methods the Microsoft Dynamics GP Workflow server uses to retrieve information about your workflow.



*To add the interface members to your class, right-click `IDynamicsWorkflow` and choose **Implement Interface**. Visual Studio adds the required interface members to your class. To define your workflow type, add code to each method that provides information about your document.*

The IDynamicsWorkflow interface requires you to implement the following methods:

- GetAvailableFilterableProperties
- GetBusinessObject
- GetOrganizationName
- GetSummaryInformation
- GetViewerName

GetAvailableFilterableProperties The following code sample implements the GetAvailableFilterableProperties method. This method defines the configuration fields you use to specify documents that require approval. To add a filterable property, you must supply the following information:

- The owner GUID. This GUID identifies the workflow class that contains these properties.
- A string value specifying the property name. Typically this is the property name from the object that is retrieved using the web service. For some data types, such as decimals, you can include a modifier that returns just the value.
- The display name for the property.

The following sample code specifies the filterable properties for leads. To identify the owner Id, the sample uses the GUID from the DynamicsWorkflow attribute that you added to the LeadApprovalSampleWorkflow class. To identify each property, the sample uses a string that matches the name of the corresponding property of the web service Lead class. To retrieve the display name, use the GetString method you added earlier to retrieve the appropriate display value from the resource file.

```
public IList<FilterableProperty> GetAvailableFilterableProperties(
    CultureInfo cultureInfo)
{
    IList<FilterableProperty> properties = new List<FilterableProperty>();

    // Specify the owner
    Guid ownerId = new Guid("42614BD0-D17A-4284-917F-73EBD19509C3");

    // Add a string property for the id
    // Retrieve the display name from the resource file
    properties.Add(new StringFilterableProperty(ownerId,
        "Key.Id", GetString("Lead_Key_Id", cultureInfo)));

    // Add a string property for the salesperson
    // Retrieve the display name from the resource file
    properties.Add(new StringFilterableProperty(ownerId, "SalespersonID",
        GetString("Lead_Salesperson", cultureInfo)));

    // Add a number property for revenue
    // Retrieve the display name from the resource file
    properties.Add(new NumberFilterableProperty(ownerId,
        "PotentialRevenue.Value",
        GetString("Lead_PotentialRevenue", cultureInfo)));

    // Add a date property
    // Retrieve the display name from the resource file
```

```

properties.Add(new DateFilterableProperty(ownerId, "QualificationDate",
    GetString("Lead_QualificationDate", cultureInfo)));

// Add an enumerated property for the lead category
// The second parameter must match the property name from the object
// Retrieve the display values from the resource file
EnumeratedFilterableProperty enumProperty = new
    EnumeratedFilterableProperty(ownerId, "LeadBusinessCategory",
        "Category");
enumProperty.EnumType = typeof(SalesLeadWebService.LeadCategory);

enumProperty.DisplayNames.Add(
    (int)SalesLeadWebService.LeadCategory.Contractor,
    GetString("Lead_BusinessCategory_Contractor", cultureInfo));
enumProperty.DisplayNames.Add(
    (int)SalesLeadWebService.LeadCategory.Educational,
    GetString("Lead_BusinessCategory_Educational", cultureInfo));
enumProperty.DisplayNames.Add(
    (int)SalesLeadWebService.LeadCategory.Media,
    GetString("Lead_BusinessCategory_Media", cultureInfo));
enumProperty.DisplayNames.Add(
    (int)SalesLeadWebService.LeadCategory.RealEstate,
    GetString("Lead_BusinessCategory_RealEstate", cultureInfo));
enumProperty.DisplayNames.Add(
    (int)SalesLeadWebService.LeadCategory.Restaurant,
    GetString("Lead_BusinessCategory_Restaurant", cultureInfo));
enumProperty.DisplayNames.Add(
    (int)SalesLeadWebService.LeadCategory.Retail,
    GetString("Lead_BusinessCategory_Retail", cultureInfo));
enumProperty.DisplayNames.Add(
    (int)SalesLeadWebService.LeadCategory.Software,
    GetString("Lead_BusinessCategory_Software", cultureInfo));
enumProperty.DisplayNames.Add(
    (int)SalesLeadWebService.LeadCategory.Wholesale,
    GetString("Lead_BusinessCategory_Wholesale", cultureInfo));

properties.Add(enumProperty);

// Add a boolean property for qualified leads
// The second parameter must match the property name from the object
// Retrieve the display values from the resource file
EnumeratedFilterableProperty boolProperty = new
    EnumeratedFilterableProperty(ownerId, "QualifiedLead",
        "Qualified Lead");
boolProperty.EnumType = typeof(bool);

boolProperty.DisplayNames.Add(0, GetString("Bool_False", cultureInfo));
boolProperty.DisplayNames.Add(1, GetString("Bool_True", cultureInfo));

properties.Add(boolProperty);

return properties;
}

```

GetBusinessObject Your workflow type must be able to retrieve a specified document. The following code sample retrieves a specified lead. Notice how a LeadKey object is created to identify the lead. The sample uses the Microsoft Dynamics GP web service context object to specify the company and currency information.

The sample also uses SharePoint elevation of privileges to retrieve the specified record. Elevation of privileges causes the DynamicsGPWorkflow application pool identity to be used when the sample Sales Lead web service is called. During installation, you assigned this login to the Workflow Administrator role in the Dynamics Security Service.

```
public object GetBusinessObject(BusinessObjectKey businessObjectKey,
    Microsoft.Dynamics.Common.OrganizationKey organizationKey)
{
    Lead lead = null;
    LeadKey leadKey = new LeadKey();

    // Instantiate the sample web service for sales leads
    SampleSalesLeadWebService leadWebService =
        new SampleSalesLeadWebService();
    leadWebService.UseDefaultCredentials = true;

    // Use the lead key to specify the sales lead
    leadKey.Id = businessObjectKey.GetKeyPartValue<string>("LeadID");

    // Create context object for use with the web service
    SalesLeadWebService.Context context = new SalesLeadWebService.Context();

    // Populate the context object's properties
    SalesLeadWebService.CompanyKey companyKey =
        new SalesLeadWebService.CompanyKey();
    companyKey.Id =
        ((Microsoft.Dynamics.Common.CompanyKey)organizationKey).Id;
    context.OrganizationKey = companyKey;

    context.CurrencyType = SalesLeadWebService.CurrencyType.Transactional;

    try
    {
        // Utilize SharePoint elevation of privileges to retrieve the
        // specified sales lead
        SPSecurity.RunWithElevatedPrivileges(
            new SPSecurity.CodeToRunElevated(delegate()
            {
                // Use the web service to retrieve the specified sales lead
                lead = leadWebService.GetLeadByKey(leadKey, context);
            }));
    }
    catch (Exception ex)
    {
        throw ex;
    }
    return lead;
}
```

GetOrganizationName To identify the Microsoft Dynamics GP company the workflow type is accessing, implement a `GetOrganizationName` method. Notice how the method takes an organization key argument and uses a base class method to return the company name.

```
public string GetOrganizationName(Microsoft.Dynamics.Common.OrganizationKey
    organizationKey)
{
    return GetCompanyName(organizationKey);
}
```

GetSummaryInformation The workflow type also retrieves a summary version of a specified business document. A summary document is an abbreviated version that contains important fields from the document. Summary information is also included with workflow notifications.

The following code sample retrieves the business document and populates the `BusinessSummaryObjectInformation` with details of the lead document:

```
// Returns a summary object for a specified Sales Lead
public BusinessObjectSummaryInformation
GetSummaryInformation(BusinessObjectKey businessObjectKey,
    object businessObject, Microsoft.Dynamics.Common.OrganizationKey
    organizationKey, CultureInfo cultureInfo)
{
    BusinessObjectSummaryInformation summaryInfo =
        new BusinessObjectSummaryInformation();

    // Cast the businessObject parameter to a Sales Lead.
    SalesLeadWebService.Lead lead = (SalesLeadWebService.Lead)businessObject;

    // Populate the summary info object with the Sales Lead ID
    summaryInfo.RecordId = new SummaryInformation("LeadSummaryRecordId",
        businessObjectKey.VisibleId, 0, "LeadSummaryRecordId");

    // Add summary information values for the current Sales Lead
    summaryInfo.SummaryInformation.Add(new SummaryInformation("LeadName",
        lead.Name, 1, "Name"));
    summaryInfo.SummaryInformation.Add(new SummaryInformation("LeadContact",
        lead.Contact, 2, "Contact"));
    summaryInfo.SummaryInformation.Add(new SummaryInformation("LeadCity",
        lead.City, 3, "City"));

    return summaryInfo;
}
```

GetViewerName The server workflow assembly specifies a web page that can be used to display the business document. This viewer is used by a workflow web client to display the business document information. The following code sample specifies the file name of an `.aspx` file that displays a lead.

```
public string GetViewerName()
{
    return "Dynamics.Workflow.GP.Samples.SalesLeadViewer.aspx";
}
```


Implement the IDynamicsWorkflowHistory interface.

Add the IDynamicsWorkflowHistory interface

To integrate with Microsoft Dynamics GP workflow, your class must implement the IDynamicsWorkflowHistory interface. The IDynamicsWorkflowHistory interface defines the methods that the Microsoft Dynamics GP Workflow server uses to save, or retrieve workflow history information.



To add the IDynamicsWorkflowHistory interface, you must first install Service Pack 3 or later for Workflow for Microsoft Dynamics GP. If you use an earlier version of workflow, you cannot add the IDynamicsWorkflowHistory interface to your class.

The IDynamicsWorkflowHistory interface requires the following methods:

- FindAllTrackingHistory
- FindAllWorkflowHistory
- SaveWorkflowHistory
- SaveWorkflowTrackingHistory

The IDynamicsWorkflowHistory interface allows you to choose how your workflow type stores workflow history. Use one of the following techniques.

- Use the Microsoft Dynamics GP archive. The approval workflow base class includes methods you can use to save or retrieve workflow history information. Your workflow history is included in the same archive as the Microsoft Dynamics GP workflow. The sample code that follows demonstrates how to use the base class methods.
- Add code to the methods that save or retrieve workflow history from a specified data store. The interface enables you to use a data or file store that resides outside of Microsoft Dynamics GP.

FindAllTrackingHistory Implement a pair of overloaded methods that retrieve the workflows associated with the specified document. The following code sample shows how to use the FindAllTrackingHistory methods from the ApprovalWorkflow base class to retrieve the workflows associated with the document.

```
Collection<TrackingHistory> IDynamicsWorkflowHistory.FindAllTrackingHistory(
    Guid WorkflowSetupID,
    Guid WorkflowImplementationID)
{
    return FindAllTrackingHistory(WorkflowSetupID, WorkflowImplementationID);
}

Collection<TrackingHistory> IDynamicsWorkflowHistory.FindAllTrackingHistory(
    BusinessObjectKey businessObjectKey,
    Microsoft.Dynamics.Common.OrganizationKey organizationKey)
{
    return FindAllTrackingHistory(businessObjectKey, organizationKey);
}
```

FindAllWorkflowHistory Implement a pair of overloaded methods that retrieve information about the individual steps associated with each workflow for a document. The following code sample uses the FindAllWorkflowHistory methods from the ApprovalWorkflow base class to retrieve history information for each workflow that is associated with the document.

```
Collection<WorkflowHistory> IDynamicsWorkflowHistory.FindAllWorkflowHistory(
    BusinessObjectKey businessObjectKey,
    Microsoft.Dynamics.Workflow.WorkflowAssociationKey associationKey)
{
    return FindAllWorkflowHistory(businessObjectKey, associationKey);
}

Collection<WorkflowHistory> IDynamicsWorkflowHistory.FindAllWorkflowHistory(
    Guid wfCorrelationID,
    int OrgID)
{
    return FindAllWorkflowHistory(wfCorrelationID, OrgID);
}
```

SaveWorkflowHistory Implement a method that saves the workflow history for the document. The following code sample shows how to use the SaveWorkflowHistory method from the ApprovalWorkflow base class to save workflow history information associated with the document.

```
void IDynamicsWorkflowHistory.SaveWorkflowHistory(Guid wfCorrelationID,
    BusinessObjectKey businessObjectKey,
    Microsoft.Dynamics.Workflow.WorkflowAssociationKey associationKey,
    System.Collections.ObjectModel.Collection<WorkflowHistory>
        workflowHistory)
{
    SaveWorkflowHistory(wfCorrelationID,
        businessObjectKey,
        associationKey,
        workflowHistory);
}
```

SaveWorkflowTrackingHistory Implement a method that saves information about the workflow instance and the document. The following code sample uses the SaveWorkflowTrackingHistory method of the ApprovalWorkflow base class to save information about the current workflow.

```
void IDynamicsWorkflowHistory.SaveWorkflowTrackingHistory(
    Guid dynamicsWorkflowID,
    Microsoft.Dynamics.Workflow.Approval.TrackingHistory trackingHistory)
{
    SaveWorkflowTrackingHistory(dynamicsWorkflowID, trackingHistory);
}
```

Adding business logic

Add a class for the event handlers.

The server workflow assembly allows you to add custom business logic to a workflow event. To create a class for your workflow business logic, open the Visual Studio Project menu and choose Add Class. Select Class from the Templates list in the Add New Item window. Enter a name for your class, and then click Add.

Inherit from the EventHandlerImplementation class.

Your new class must inherit from the Microsoft.Dynamics.GP.Workflow.EventHandlerImplementation class. Add methods and properties to the class that implement the business logic for the event.

For example, lead documents require approval when the assigned salesperson changes. The event handler must perform three operations:

- Always update the data table to save the new workflow status.
- If the approver approves the change, update the field that saves the current salesperson as the approved salesperson.
- If the approver rejects the change, update the salesperson to be the value of the last approved salesperson.

The following code sample implements the event handler operations for the lead workflow example. Notice the use of the Sales Lead web service to retrieve existing lead data. Also, notice how the Update method of the EventHandlerImplementation base class uses database parameters, database connections, and SQL statements to save changes to the database.

```
class GPWorkflowSampleEventHandlers : EventHandlerImplementation
{
    // Implement a method to perform the status change to a lead
    // Always update the Workflow_Approval_Status field to reflect the change
    // that has occurred
    internal void OnStatusChanged(object sender,
        WorkflowStatusChangedEventArgs args)
    {
        try
        {
            // Retrieve the company and business object keys
            Microsoft.Dynamics.Common.CompanyKey companyKey =
                (Microsoft.Dynamics.Common.CompanyKey)args.
                DynamicsActivationArgs.OrganizationKey;
            DynamicsActivationArgs activationArgs = args.DynamicsActivationArgs;
            BusinessObjectKey businessObjectKey =
                args.DynamicsActivationArgs.
                BusinessObjectSubmissionInformation.BusinessObjectKey;

            // Retrieve the object Id
            string leadId =
                businessObjectKey.GetKeyPartValue<string>("LeadID");

            // Create a SQL statement to perform the operation
            string updateLead = "UPDATE IG001 SET Workflow_Approval_Status =
                @Workflow_Approval_Status WHERE LeadID = @LeadID";
```

```

// Get the connection to GP database
DbProviderFactory dbFactory =
    Connection.GetInstance().DbProviderFactory;

// Create parameters for the SQL statement
Collection<DbParameter> parameters =
    new Collection<DbParameter>();

// Create a parameter for the field name
// The parameter value argument should be the integer equivalent
// of the current WorkflowApprovalStatus enum value
DbParameter fieldParam = CreateDbParameter(dbFactory,
    "@Workflow_Approval_Status",
    DbType.String, ParameterDirection.Input,
    ((int)args.WorkflowApprovalStatus).ToString());
parameters.Add(fieldParam);

// Create a parameter for the Id
DbParameter leadParam = CreateDbParameter(dbFactory, "@LeadID",
    DbType.String, ParameterDirection.Input, leadId);
parameters.Add(leadParam);

// Perform the database operation
Update(companyKey, updateLead, parameters);

// If the status is approved or pending approval, update the
// Approved_Salesperson_ID to the value of SLSPRSNID
WorkflowApprovalStatus approved =
    WorkflowApprovalStatus.Approved;
WorkflowApprovalStatus rejected =
    WorkflowApprovalStatus.Rejected;

if (args.WorkflowApprovalStatus.Equals(approved))
{
    ApproveSalespersonChange(companyKey, leadId);
}
// If the status is rejected, update the SLSPRSNID to the value of
// the Approved_Salesperson_ID. This rollsback the proposed change
// to the last approved salesperson value.
else if (args.WorkflowApprovalStatus.Equals(rejected))
{
    RejectSalespersonChange(companyKey, leadId);
}
else
{
    // no op
}
}
catch (Exception ex)
{
    throw ex;
}
}

```

```

// Creates a database parameter object that can be used by the Update
// statements
private DbParameter CreateDbParameter(DbProviderFactory factory,
    string paramName,
    DbType paramType,
    ParameterDirection paramDirection,
    object paramValue)
{
    // Instantiate a Db parameter object
    DbParameter parameter = factory.CreateParameter();

    // Populate the object properties
    parameter.ParameterName = paramName;
    parameter.DbType = paramType;
    parameter.Direction = paramDirection;
    parameter.Value = paramValue;

    // return the parameter object
    return parameter;
}

// The approver accepted the salesperson change
// Update the Approved_Salesperson_ID field to be the value of the
// SLPRSNID field
private void ApproveSalespersonChange(
    Microsoft.Dynamics.Common.CompanyKey compKey,
    string leadId)
{
    SalesLeadWebService.CompanyKey companyKey;
    SalesLeadWebService.Context context;

    // Create an instance of the web service
    SampleSalesLeadWebService wsSampleService =
        new SampleSalesLeadWebService();

    // Make sure that default credentials are being used
    wsSampleService.UseDefaultCredentials = true;

    // Create a context with which to call the web service
    context = new SalesLeadWebService.Context();

    // Specify which company to use (lesson company)
    companyKey = new SalesLeadWebService.CompanyKey();
    companyKey.Id = (-1);

    // Set up the context
    context.OrganizationKey =
        (SalesLeadWebService.OrganizationKey)companyKey;
    context.CultureName = "en-US";

    // Create a lead key object to specify the lead to retrieve
    LeadKey myLeadKey = new LeadKey();
    myLeadKey.Id = leadId;
}

```

```

try
{
    // use the web service to retrieve the specified lead object
    Lead myLead = wsSampleService.GetLeadByKey(myLeadKey, context);

    // Create a SQL statement to perform the operation
    string updateLead = "UPDATE IG001 SET Approved_Salesperson_ID =
        @Approved_Salesperson_ID WHERE LeadID = @LeadID";

    // Get the connection to GP database
    DbProviderFactory dbFactory =
        Connection.GetInstance().DbProviderFactory;

    // Create parameters for the SQL statement
    Collection<DbParameter> parameters =
        new Collection<DbParameter>();

    // Create a parameter for the field name
    // The parameter value argument should be the integer equivalent
    // of the current WorkflowApprovalStatus enum value
    DbParameter fieldParam = CreateDbParameter(dbFactory,
        "@Approved_Salesperson_ID",
        DbType.String,
        ParameterDirection.Input,
        myLead.SalespersonID);
    parameters.Add(fieldParam);

    // Create a parameter for the Id
    DbParameter leadParam = CreateDbParameter(dbFactory,
        "@LeadID", DbType.String, ParameterDirection.Input, leadId);
    parameters.Add(leadParam);

    // Perform the database operation
    Update(compKey, updateLead, parameters);
}
catch (Exception err)
{
    throw err;
}

// The approver rejected the salesperson change
// Update the SLPRSNID field to be the value of the
// Approved_Salesperson_ID field
private void RejectSalespersonChange(Microsoft.Dynamics.Common.CompanyKey
    compKey, string leadId)
{
    SalesLeadWebService.CompanyKey companyKey;
    SalesLeadWebService.Context context;

    // Create an instance of the web service
    SampleSalesLeadWebService wsSampleService =
        new SampleSalesLeadWebService();

```

```

// Make sure that default credentials are being used
wsSampleService.UseDefaultCredentials = true;

// Create a context with which to call the web service
context = new SalesLeadWebService.Context();

// Specify which company to use (lesson company)
companyKey = new SalesLeadWebService.CompanyKey();
companyKey.Id = (-1);

// Set up the context
context.OrganizationKey =
    (SalesLeadWebService.OrganizationKey)companyKey;
context.CultureName = "en-US";

// Create a lead key object to specify the lead to retrieve
LeadKey myLeadKey = new LeadKey();
myLeadKey.Id = leadId;

try
{
    // Use the web service to retrieve the specified lead object
    Lead myLead = wsSampleService.GetLeadByKey(myLeadKey, context);

    // If the ApprovedSalespersonID is not empty, change SalespersonID
    // to the value of ApprovedSalespersonID. This rolls back the
    // salesperson change to the last approved salesperson.
    if(myLead.ApprovedSalespersonID.Length > 0)
    {
        // Create a SQL statement to perform the operation
        string updateLead = "UPDATE IG001 SET SLPRSNID =
            @SLPRSNID WHERE LeadID = @LeadID";

        // Get the connection to GP database
        DbProviderFactory dbFactory =
            Connection.GetInstance().DbProviderFactory;

        // Create parameters for the SQL statement
        Collection<DbParameter> parameters =
            new Collection<DbParameter>();

        // Create a parameter for the field name
        // The parameter value argument should be the integer
        // equivalent of the current WorkflowApprovalStatus enum value
        DbParameter fieldParam = CreateDbParameter(dbFactory,
            "@SLPRSNID",
            DbType.String,
            ParameterDirection.Input,
            myLead.ApprovedSalespersonID);
        parameters.Add(fieldParam);
    }
}

```

```

// Create a parameter for the Id
DbParameter leadParam = CreateDbParameter(dbFactory,
    "@LeadID",
    DbType.String,
    ParameterDirection.Input,
    leadId);
parameters.Add(leadParam);

// Perform the database operation
Update(compKey, updateLead, parameters);
    }
}
catch (Exception err)
{
    throw err;
}
}
}

```

Signing your server workflow assembly

Add a strong name key to the project.

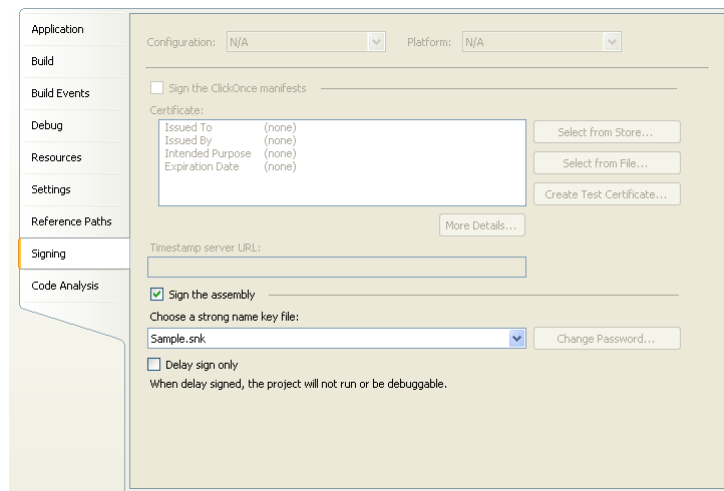
Add a strong name key to your project. The strong name key ensures your assembly is uniquely named and allows the .NET framework to ensure the assembly has not been changed. To provide a strong name key, use Visual Studio to sign your server workflow assembly.

In addition, the server workflow assembly must be added to the workflow server's global assembly cache (GAC). The GAC requires your assembly to be signed using a strong name key. Signing will create a strong name key (.snk) file for your assembly. To sign your server workflow assembly, use the following procedure.

1. Open the Visual Studio Project Designer.

With your project node selected in Solution Explorer, open the Project menu, choose Properties (or right-click the project node in Solution Explorer, and click Properties).

2. In the Project Designer, click the Signing tab.



3. Select the Sign the assembly check box.

Specify an existing key file. In the Choose a strong name key file drop-down list, select <Browse...>.



To enable development and testing, use the signing procedure to create a temporary strong name key. When prompted with the Choose a strong name key file drop-down list, select <New> and create a new strong name key.

4. Select the key file.

In the Select File dialog box, navigate to your key file or enter its path in the File name box. Click Open to select it.

Visual Studio creates the strong name key and adds the .snk file to your server workflow assembly project.

Creating a document viewer

Add a web page to the project that serves as a document viewer.

The Microsoft Dynamics Workflow site and other web based workflow clients provide links that allow you to view the workflow document. The link opens a SharePoint web page that displays information from the business document. To create the link's URL, workflow calls the `IDynamicsWorkflow.GetViewerName` method from the server workflow assembly.

When you create a new workflow, you must create a web page that displays data from your document. To simplify the development of the viewer, use the same viewer base class as the document viewers of the existing Dynamics GP workflows.



The filename of your document viewer must match the name specified by your `GetViewerName` method. If they are not identical, the links to your document viewer will create errors when you try to view the document.

The following example is the lead document viewer from the SDK sample application:

Microsoft Dynamics Workflow

Welcome Bruce Rivard | My Site | My Links | Site Actions

Home

View All Site Content

- Tasks
- Documents
- Reports
- My Delegation
- My Alerts
- Administration
- Recycle Bin

Lead Name:	Proseware, Inc.		
Lead Id:	1003	Category:	Software
Potential Revenue:	\$25,000.00	Qualified Lead:	True
Source:	Web site submission		
Contact Information:	Sally		
Address 1:	123 Longhorn Road	Address 2:	
City:	Chicago	State:	IL
Zip:	68234		
Assigned salesperson:	IAN M.		
Approved salesperson:	IAN M.		

Include the document viewer in the Visual Studio solution of your server workflow assembly. To create a new document viewer, complete the following steps:

1. Create a Layouts folder.

In the Visual Studio Solution Explorer, right-click your workflow server class. Choose Add >> New Folder from the menu. Name the folder Layouts.

2. Add a file to the Layouts folder.

In Solution Explorer, right-click the Layouts folder you created. Choose Add >> New Item. Select HTML Page from the Templates presented by the Add New Item window. Enter a name for your file and click Add.

3. Rename the file.

In Solution Explorer, right-click the HTML file you created and choose Rename. Enter the name of your document viewer. Your document viewer must be a .aspx file. Press Enter to save your filename.

4. Edit the file.

Open the file you created. Delete all file contents.

With the file renamed and empty, you can begin to build your document viewer. Your document viewer uses the same basic structure as the viewers for the existing Dynamics GP workflows.

```
<%@ Page Language="C#" MasterPageFile="DynamicsWorkflow.master"
Inherits="Microsoft.Dynamics.GP.Workflow.DocumentViewer,Microsoft.Dynamics.G
P.Workflow.Pages,Culture=neutral" EnableSessionState="false"
AutoEventWireup="true"%>
<%@ Register TagPrefix="Dynamics" Namespace="Microsoft.Dynamics.GP.Workflow"
Assembly="Microsoft.Dynamics.GP.Workflow.Pages, Version=10.0.0.0,
Culture=neutral" %>
<%@ Register Tagprefix="DynCommon"
Namespace="Microsoft.Dynamics.Workflow.Common"
Assembly="Microsoft.Dynamics.Workflow.Common, Version=10.0.0.0,
Culture=neutral"%>
<%@ Register Tagprefix="GPProxy" Namespace="Microsoft.Dynamics.GP.Proxy"
Assembly="Microsoft.Dynamics.GP.WebServices.Proxy, Version=10.0.0.0,
Culture=neutral"%>
<%@ Import Namespace="Microsoft.Dynamics.Workflow" %>
<%@ Import Namespace="Microsoft.Dynamics.GP.Workflow" %>
<%@ Register Tagprefix="SharePoint"
Namespace="Microsoft.SharePoint.WebControls" Assembly="Microsoft.SharePoint,
Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="Utilities" Namespace="Microsoft.SharePoint.Utilities"
Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" %>

<script runat="server">

</script>
<asp:Content ID="Content3" ContentPlaceHolderID="PlaceholderPageImage"
runat="server">
    <SharePoint:ViewIcon Width="145" Height="54"
        Src="/_layouts/images/generic.png" runat="server"/>
</asp:Content>
```

```

<asp:Content ID="Content1" ContentPlaceHolderId="PlaceHolderMain"
  runat="server">
  <div runat="server" id="HelpTextSection" class="ms-standard-header
    dyn-section-overview-title">
    <h3 runat="server" id="WorkflowTypeLabel" />

  </div>
</asp:Content>

```

The following sample code adds methods to the documents script node. When the page loads, it stores the Id of the lead. The page overrides the base class `GetBusinessObjectKey` to return a `BusinessObjectKey` that uniquely identifies the current lead. The `CurrentDocument` property provides access to the data in the business document. This `<script>` node is appended to the .aspx file from the previous sample code.

```

<script runat="server">
  private string leadId;

  void Page_Load(Object sender, EventArgs e)
  {
    if (!this.IsPostBack)
    {
      leadId = this.CurrentDocument.Key.Id;
    }
  }

  protected override Microsoft.Dynamics.Workflow.Common.
    BusinessObjectKey GetBusinessObjectKey()
  {
    if (string.IsNullOrEmpty(this.Request.Params["LeadID"]))
    {
      DisplayInvalidParametersMessage();
    }
    Microsoft.Dynamics.Workflow.Common.BusinessObjectKey key =
      Microsoft.Dynamics.Workflow.Common.
        BusinessObjectKey.CreateInstance();
    key.KeyParts.Add(new KeyPart<string>("LeadID",
      this.Request.Params["LeadID"]));
    return key;
  }

  public SalesLeadWebService.Lead CurrentDocument
  {
    get
    {
      return (SalesLeadWebService.Lead)this.BusinessObject;
    }
  }
</script>

```

The following sample code adds a table to web page <div> element. The table displays several fields containing lead data. Notice how the CurrentDocument property is used to access the specific data fields from the lead document. The <div> node completes the .aspx file that was used in the previous code examples.

```
<div runat="server" id="HelpTextSection" class="ms-standard-header
  dyn-section-overview-title">
<h3 runat="server" id="WorkflowTypeLabel" />
  <table class="ms-descriptiontext dyn-section-overview-extended"
    align=left>
    <tr>
      <td style="width: 150px">
        <strong><asp:Literal ID="Literal3" runat="server" Text="Lead Name:">
        </asp:Literal></strong>
      </td>
      <td>
        <strong><%=this.CurrentDocument.Name%></strong>
      </td>
    </tr>
    <tr>
      <td style="width: 150px">
        <asp:Literal ID="Literal1" runat="server" Text="Lead Id:">
        </asp:Literal>
      </td>
      <td>
        <%=leadId %>
      </td>
      <td style="width: 150px">
        <asp:Literal ID="Literal2" runat="server" Text="Category:">
        </asp:Literal>
      </td>
      <td>
        <%=this.CurrentDocument.LeadBusinessCategory%>
      </td>
    </tr>
    <tr>
      <td style="width: 150px">
        <asp:Literal ID="Literal4" runat="server" Text="Potential Revenue:">
        </asp:Literal>
      </td>
      <td>
        <%=this.CurrentDocument.PotentialRevenue.Value.ToString("C")%>
      </td>
      <td style="width: 150px">
        <asp:Literal ID="Literal15" runat="server" Text="Qualified Lead:">
        </asp:Literal>
      </td>
      <td>
        <%=this.CurrentDocument.QualifiedLead%>
      </td>
    </tr>
```

```

<tr>
  <td style="width: 150px">
    <asp:Literal ID="Literal8" runat="server" Text="Source:">
    </asp:Literal>
  </td>
  <td>
    <%=this.CurrentDocument.LeadSource%>
  </td>
</tr>
<tr></tr>
<tr></tr>
<tr></tr>
<tr>
  <td style="width: 150px">
    <strong>
      <asp:Literal ID="Literal5" runat="server"
        Text="Contact Information:">
      </asp:Literal></strong>
    </td>
    <td>
      <%=this.CurrentDocument.Contact%>
    </td>
</tr>
<tr>
  <td style="width: 150px">
    <asp:Literal ID="Literal7" runat="server" Text="Address 1:">
    </asp:Literal>
  </td>
  <td>
    <%=this.CurrentDocument.Address1%>
  </td>
  <td style="width: 150px">
    <asp:Literal ID="Literal9" runat="server" Text="Address 2:">
    </asp:Literal></td><td><%=this.CurrentDocument.Address2%>
  </td>
</tr>
<tr>
  <td style="width: 150px">
    <asp:Literal ID="Literal11" runat="server" Text="City:">
    </asp:Literal></td><td><%=this.CurrentDocument.City%>
  </td>
  <td style="width: 150px">
    <asp:Literal ID="Literal12" runat="server" Text="State:">
    </asp:Literal>
  </td>
  <td>
    <%=this.CurrentDocument.State%>
  </td>
</tr>

```

```

<tr>
  <td style="width: 150px">
    <asp:Literal ID="Literal13" runat="server" Text="Zip:">
    </asp:Literal>
  </td>
  <td>
    <%=this.CurrentDocument.Zip%>
  </td>
</tr>
<tr></tr>
<tr></tr>
<tr></tr>
<tr>
  <td style="width: 250px"><strong>
    <asp:Literal ID="Literal14" runat="server" Text="Assigned
      salesperson:">
    </asp:Literal></strong>
  </td>
  <td>
    <%=this.CurrentDocument.SalespersonID%>
  </td>
</tr>
<tr>
  <td style="width: 250px"><strong>
    <asp:Literal ID="Literal6" runat="server" Text="Approved
      salesperson:">
    </asp:Literal></strong>
  </td>
  <td>
    <%=this.CurrentDocument.ApprovedSalespersonID%>
  </td>
</tr>
</table>
</div>

```

Creating a workflow event subscription helper application

Create an application that performs the workflow event subscription.

To perform your workflow business logic, your workflow type subscribes to events raised by the workflow server. You subscribe to the specific events for which you have added business logic. For example, the lead sample workflow has business logic for the StatusChanged event, so it must subscribe to that event.

When you subscribe to a workflow event, you associate the workflow event with an event handler in your server workflow assembly. The DynamicsWorkflowEvents.config file contains all of the workflow subscriptions for your Workflow server. To create a new subscription, you must add your workflow event and event handler to the DynamicsWorkflowEvents.config file.

The workflow framework provides a WorkflowEventManager that you will use to add subscriptions to the DynamicsWorkflowEvents.config file. To use the WorkflowEventManager, create an executable that uses the WorkflowEventManager. You must run this application when you install your workflow server assembly on the Microsoft Dynamics GP Workflow server.

Create a Visual Studio project

To create the application to subscribe to workflow events, create a new Visual Studio project. You can create the project as a separate solution or add it to your server workflow assembly solution.

To create the project, complete the following steps:

1. Create a new project.

To create a subscription application, add a new Visual Studio project to your server workflow assembly solution. In Visual Studio, open the File menu and choose New >> Project. From the New Project window, select the desired project type from the list of templates. Enter a name and specify a project location. In the Solution list, select Add to Solution. Click OK.

2. Add references to the project.

From the Project menu, select Add References. The Add References window opens. Click the Browse tab. Navigate to the Microsoft Dynamics GP “Workflow” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\Workflow

Select the following .dll file and click OK:

- Microsoft.Dynamics.GP.Workflow.dll

Also, add a reference to your server workflow assembly project. From the Project menu, choose Add References. The Add References window opens. Click the Projects tab. Select your server workflow assembly project from the Project Name list. Click OK.

Open the class file

The following code sample shows the class Visual Studio creates for a console application.

```
class Program
{
    static void Main(string[] args)
    {

    }
}
```

Add the following **using** statements. Include the namespace of your server workflow assembly. Notice how the following code sample adds the namespace of the Microsoft.Dynamics.GP.Workflow.Samples.Server assembly created for the lead sample workflow.

```
using Microsoft.Dynamics.Workflow;
using Microsoft.Dynamics.GP.Workflow.Samples.Server;
```

Use the DynamicsWorkflowEventManager

To update the DynamicsWorkflowEvents.config file with a new subscription, instantiate the Microsoft.Dynamics.Workflow.WorkflowEventManager object and use its RegisterEventHandler method. The RegisterEventHandler method requires you to supply the following parameters:

Parameter	Description
WorkflowEvent	An enumeration value that specifies the workflow event. The WorkflowEvent enumeration contains the following: WorkflowStarted StatusChanged TaskCreated TaskCompleted TasksEscalated TaskDeleted TaskRolledBack StepCompleted WorkflowError WorkflowCompleted WorkflowDeactivated BusinessObjectChanged
DynamicsWorkflow	The workflow class that handles the specified event.
SoftwareVendor	A string that identifies the company that created the workflow class.
Type	Specifies the class that contains the subscription's event handler.
MethodName	A string that specifies the name of the event handler.

The following code sample uses the RegisterEventHandler to create a subscription to the StatusChanged event for the lead sample application. Notice how the targeted workflow is specified by namespace, type name, and the name of the event handler method.

```
// Create an new event handler in the DynamicsWorkflowEvent
// configuration file.
WorkflowEventManager.RegisterEventHandler(
    WorkflowEvent.StatusChanged,
    DynamicsWorkflow.FindByName("Sample Salesperson Approval Workflow"),
    "GPWorkflowSample",
    typeof(LeadApprovalWorkflowSample),
    "OnStatusChanged");
```

Add installation tasks

You run your event subscription helper application when you install your server workflow assembly on the Workflow for Dynamics GP server. You may want to use the application to perform other routine installation tasks.

For example, the Microsoft Dynamics GP Workflow SDK sample application uses a web service to retrieve lead documents. To authorize workflow to use the web service, the appropriate Dynamics Security Service operations and tasks must be added to the Workflow Administrator role. To learn more about creating Dynamics Security Service tasks, see [Securing the web service](#) of [Chapter 8, "Web Service."](#)

To perform this additional task, add a method to your event subscription helper application that updates the Dynamics Security Service. Use this method to update the Dynamics Security Service Workflow Administrator role.

Before you add code to this method, you must first add several new reference to your project. From the Visual Studio Project menu, choose Add References. Click the .NET tab and select the following assembly, and then click OK:

- System.Web.Services.dll

From the Project menu, choose Add References. Click the Browse tab. Browse to the “Workflow” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\Workflow

Add the following .dll files:

- Microsoft.Dynamics.Workflow.dll
- Microsoft.Dynamics.Security.AdminServices.Proxy.dll

You also need to add the following **using** statements.

```
using System.Collections;
using Microsoft.Dynamics.Security.AdminServices.Proxy;
using System.Web.Services.Protocols;
```

The following code sample adds the Dynamics Security Service task that was created during the installation of the web service to the Workflow Administrator role. Notice how GUID values are used to specify the application, the workflow administrator role, and the “View leads” task.



The code sample contains a URL for the Dynamics Security Service. The URL assumes the Dynamics Security Service is installed on this server. To provide greater flexibility, add a key to the application configuration file that specifies the URL. This allows you to update the Dynamics Security Service where ever it is installed.

```
// Method adds the View leads task to the workflow admin role
static void AddDSSOperation()
{
    DynamicsSecurityAdminService securityService =
        new DynamicsSecurityAdminService();
    securityService.Url =
        @"http://localhost:49152/DynamicsAdminService.asmx";
    securityService.UseDefaultCredentials = true;

    SecurityContext securityContext = new SecurityContext();
    ApplicationKey appKey = new ApplicationKey();
    appKey.Id = "25cc1a21-2cc4-4b13-a1c8-eeal86fb688a";
    securityContext.ApplicationKey = appKey;

    // Use a role key to specify the workflow administrator role
    // The GUID value identifies the workflow admin
    RoleKey roleKey = new RoleKey();
    roleKey.Id = "196871c6-984c-4021-ab5d-424fda14b083";

    Role role = securityService.GetRoleByKey(roleKey, securityContext);
```

```

// Create a task key
// Specify the GUID that uniquely identifies the View Leads task
TaskKey taskKey = new TaskKey();
taskKey.Id = "A33871C6-8393-4040-803D-FA1EF7306136";

// Create an arraylist of task keys from the role's existing task keys
ArrayList taskList = new ArrayList();
foreach(TaskKey t in role.Tasks)
{
    taskList.Add(t);
}

// Add the task key for the View Leads task
taskList.Add(taskKey);

// Replace the role's existing array with the array that contains
// the View Leads task key
role.Tasks = (TaskKey[])taskList.ToArray(typeof(TaskKey));

// Update the workflow admin role to include the new task
securityService.UpdateRole(role, securityContext);
}

```

To run this method, call the `AddDSSOperation` method from the console application's `Main` method.

```

class Program
{
    static void Main(string[] args)
    {
        try
        {
            // Subscribe to the Status Changed event
            WorkflowEventManager.RegisterEventHandler(
                WorkflowEvent.StatusChanged,
                DynamicsWorkflow.FindByName(
                    "Sample Salesperson Approval Workflow"),
                "GPWorkflowSample",
                typeof(LeadApprovalWorkflowSample),
                "OnStatusChanged");

            // Update the Dynamics Security Service
            AddDSSOperation();
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error completing administrative tasks");
            Console.WriteLine("Error Message: " + ex.Message);
            Console.WriteLine("Stack Trace: " + ex.StackTrace);
            Console.WriteLine("Press any key to quit");
            Console.ReadLine();
        }
    }
}

```

Add an application configuration file to the project.

Create an application configuration file

To update the DynamicsWorkflowEvents.config file, you need to supply the file path to that configuration file. Use an application configuration file to store this file path.

To create an application configuration file, open the Project menu and choose Add New Item. Select Application Configuration File from the Add New Item window Templates list. Click Add.

From Solution Explorer, open the App.config file. The file contains an empty configuration.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
</configuration>
```

In `<configuration>` add an `<appSettings>` node that adds a `BusinessObjectConfigurationPath` key. The key value specifies the file path to the DynamicsWorkflowEvents.config file. Typically, this will be in the following location:

```
C:\Inetpub\wwwroot\wss\VirtualDirectories\port#\bin\DynamicsWorkflowEvents.config
```

You need to supply the port number where you installed Dynamics GP Workflow web service.

The following code sample shows an application configuration file with a key that specifies the location of the DynamicsWorkflowEvents.config file.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="BusinessObjectsConfigurationPath"
      value="C:\Inetpub\wwwroot\wss\VirtualDirectories\10072\bin\
      DynamicsWorkflowEvents.config" />
  </appSettings>
</configuration>
```

Building the assembly and application

Build the assembly and the subscription application.

Use Visual Studio to build your server workflow assembly and your subscription application. Open your solution file with Visual Studio. From the Visual Studio Build menu, choose Build Solution. Visual Studio builds your server workflow assembly and your subscription application. Visual Studio places the assembly and application in the “\bin\debug” folder of each project.

Chapter 10: Deploying the New Workflow

After building the components for the new workflow, you must deploy each component. The following sections contain instructions for deploying and registering your workflow components:

- [Installing the application](#)
- [Installing the workflow document type](#)
- [Installing the web service](#)
- [Installing the server workflow assembly](#)
- [Installing the client workflow assembly](#)
- [Viewing the workflow](#)

Installing the application

Install the back-office application.

Install your back-office application for Microsoft Dynamics GP. Add your application install files to the Dynamics GP client folder, typically found at the following location:

C:\Program Files\Microsoft Dynamics\GP

Install your back-office application using your normal install procedure. When installation is complete, close and reopen the Microsoft Dynamics GP client to verify the success of your back-office application installation.

Installing the workflow document type

Install the new document type assembly.

Install the assembly that defines your document type. Place the file in the Web Services for Microsoft Dynamics GP “bin” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\bin

Installing the web service

Install the web service.

To install a new web service, you must place your web service files into several locations on the server where you installed Web Services for Microsoft Dynamics GP.

Update the eConnect Transaction Requester

Add your document type to eConnect_Out_Setup table in the company database. Typically, you will run a SQL script that inserts your document type into the eConnect_Out_Setup table. To see an example of a SQL script that adds a document type to the eConnect Transaction Requester, refer to [Creating a web service](#) on page 114

Install the XSLT file

Install your XSLT file to the Dynamics GP web services “XSLT” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\bin\XSLT

Install the web service

Install your web service .asmx file to the Dynamics GP web service “WebServices” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices

To help secure your web service, use the security properties of the .asmx file to specify access permissions. Add the names of users or groups that use your web service to the access list of your .asmx file.

For example, you typically want your web service to be accessed by the same users as the Microsoft Dynamics GP web services. To change your .asmx file security settings, right-click that file, and choose Properties. When the Properties window opens, click the Security tab, and then click Add. From the Select Users, Computers, or Groups window, type Authenticated Users into the Enter the object names to select box, and then click OK. Mark the Allow check box for the Read and Read & Execute permissions. Click OK to close the Properties window. Repeat these steps for any other users or groups that you want to access your web service.

Place your web service code-behind file in the “bin” folder, typically in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\bin

Secure the web service

Place your web service security metadata helper application in the Dynamics GP web services “GPWebServices” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\GPWebServices

To add new security operations and tasks to roles in the Dynamics Security Service, run your web service security metadata helper application.



The Dynamics Security Service uses Active Directory Application Mode (ADAM) to store information about web service security roles, operations, and tasks. To allow the security metadata helper application to add operations and tasks to ADAM, you must be logged in as user who is an ADAM administrator. If you run the security metadata helper application using a login that is not an ADAM administrator, ADAM will return a “System.UnauthorizedAccessException” error. By default, the user who installed Dynamics GP web services is an ADAM administrator. To add another user as an ADAM administrator, see the Web Services Installation and Administration Guide.

Use the Dynamics Security Console to add your new security tasks and operations to the appropriate Dynamics Security Service roles.

Installing the server workflow assembly

Install the server workflow assembly and assembly configuration file.

You must install the server workflow assembly, the server workflow assembly configuration file, and the event subscription helper application on the server where you installed Microsoft Dynamics GP Workflow.



Before installing the server workflow assembly configuration file, be sure the file contains the URL of the web service you will use with your server workflow assembly.

Install your server workflow assembly, and server workflow assembly configuration file to the following folders:

- The workflow folder, typically found at the following location:
C:\Program Files\Microsoft Dynamics\Workflow.
- The Dynamics GP web services “bin” folder, typically found in the following location:
C:\inetpub\wwwroot\wss\VirtualDirectories\port#\bin

You must supply the port number where you installed the Microsoft Dynamics Workflow web service.

Add the assembly to the Global Assembly Cache

Add the server workflow assembly to the global assembly cache.

You must also install your server workflow assembly into the global assembly cache (GAC). You can add your assembly .dll file to the global assembly cache by dragging the .dll file to the appropriate directory, or by using the Gacutil Tool.

If you want to drag the file, open a second instance of Microsoft Windows Explorer. In the new Windows Explorer, open the “Assembly” folder, typically found in the following location:

C:\Windows\Assembly

Drag and drop your server workflow assembly into the Assembly folder.

To use the Gacutil Tool, complete the following procedure.

1. Open the Visual Studio Command Prompt.

In the Program Group for Microsoft Visual Studio, point to Visual Studio Tools and choose Visual Studio Command Prompt. A command prompt will be displayed.

2. Change the working location to the folder where you placed your server workflow assembly.

Using the command prompt, change the working location to the Dynamics GP web service “Workflow” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\Workflow

3. Use the Gacutil Tool to load the assembly in the global assembly cache.

Use the following command to add your server workflow assembly to the global assembly cache:

```
gacutil -I "c:\Program Files\Microsoft Dynamics\Workflow\
ServerWorkflowAssembly.dll"
```

Replace *ServerWorkflowAssembly* with the name of your assembly.

Register the workflow

To add your workflow to your Microsoft Dynamics GP Workflow server, you register your workflow assembly and workflow type with the Workflow server. Before using the following registration commands, verify that the specified workflow assembly has been installed in the “Workflow” folder:

C:\Program Files\Microsoft Dynamics\Workflow



When using the following registration utilities, the filename specified by /ASSEMBLY= parameter should not include the file extension. For example, use Microsoft.Dynamics.GP.Workflow.Samples.Server to represent an assembly named Microsoft.Dynamics.GP.Workflow.Samples.Server.dll.

Register the workflow schedule.

Use Microsoft.Dynamics.Workflow.Install.RegistrationSchedule.exe to register your workflow type with Microsoft Dynamics GP Workflow server. Open a command prompt and set the working directory to the “Workflow” folder:

C:\Program Files\Microsoft Dynamics\Workflow

Enter the following command. In the /ASSEMBLY parameter, use the name of your server workflow assembly. Press Enter.

```
Microsoft.Dynamics.Workflow.Install.RegisterSchedule.exe /ACTION=LOAD
/DATABASE=DYNAMICS /ASSEMBLY=AssemblyName
```

Register the workflow assembly.

Use Microsoft.Dynamics.Workflow.Install.ApplicationServer.exe to register your new workflow with the workflow server.

Enter the following command. For the /ASSEMBLY parameter, specify the name of your workflow assembly. For the /CLASSES parameter, specify the name of your workflow type. Press Enter.

```
Microsoft.Dynamics.Workflow.Install.ApplicationServer.exe /ACTION=LOAD
/ASSEMBLY=AssemblyName /CLASSES=WorkflowType /COREPERFORMANCECOUNTERS=true
/WORKFLOWPERFORMANCECOUNTERS=true /DYNAMICSEVENTLOGSOURCES=true
```



To simplify installation, create a batch file that contains the preceding commands.

Install the subscription application.

Install the subscription application

Install the application and the application configuration files to the workflow “Workflow” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\Workflow.

Execute the subscription application.

Subscribe to the workflow events

You created an event subscription helper application that adds your workflow event handlers to the DynamicsWorkflowEvents.config file. You may also use the application to add web service operations to the Workflow Administrator role in the Dynamics Security Service. To complete these two steps, use the following procedure to run your subscription application.

1. Open a Command Prompt.

From the Start menu, open Programs >> Accessories >> Command Prompt. A command prompt will be displayed.

2. Change the working location to the folder where you placed your subscription application.

Using the command prompt, change the working location to the “Workflow” folder, typically found in the following location:

C:\Program Files\Microsoft Dynamics\Workflow

3. Run your subscription application.

Use the following command to run your subscription application:

SubscriptionApplication.exe

Replace *SubscriptionApplication* with the name of your application.



If you also use the event subscription helper application to update Dynamics Security Service roles, you must ensure the web service security application has been installed and run. The event subscription helper application cannot update roles until the security operations and tasks have been added to the Dynamics Security Service.

Install the document viewer

Install the document viewer you created in the SharePoint “Layouts” folder, typically found in the following location:

C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\templates\layouts

Installing the client workflow assembly

Install the client workflow assembly with the Microsoft Dynamics GP client.

To install the client workflow assembly, you must place the assembly in the install folder of the Microsoft Dynamics GP client and add entries to the Dynamics GP client’s Dynamics.exe.config configuration file.



You must install your client workflow assembly on every Microsoft Dynamics GP client that requires access to workflow.

Install the assembly

Copy your client workflow assembly to the Dynamic GP client folder on the Dynamics GP client machine. Typically, the folder is in the following location:

C:\Program Files\Microsoft Dynamics\GP

Add the form factory registration to the client configuration file.

Register the client workflow assembly

To enable workflow on the Microsoft Dynamics GP client, you must register your form factory or factories with the client. Use the following procedure to register the form factories in your client workflow assembly with the Microsoft Dynamics GP client.

1. Backup the Dynamics.exe.config file.

Browse to the Dynamics GP client folder, typically in the following location:

C:\Program Files\Microsoft Dynamics\GP

Make a copy of the Dynamics.exe.config file. Store the back up copy in a safe location.

2. Edit the Dynamics.exe.config file.

Open the Dynamics.exe.config file in a text editor.

3. Add your form factory registrations.

The Dynamics.exe.config file contains a formFactories node that contains the existing form factory registrations for Dynamics GP. To register your form factory, you must add the following information:

Parameter	Description
name	A name that uniquely identifies the workflow.
productId	Specifies the product Id of the form associated with the form factory.
formId	Specifies the resource Id of the form associated with the form factory.
windowId	Specifies the resource Id of the window associated with the form factory.
factoryType	Specifies the form factory class and the name of the client workflow assembly that contain the form factory class.

You add a form factory for each Microsoft Dynamics GP client form that your client workflow assembly supports. The following code sample adds a form factory for the sample application Lead Maintenance form and the modal dialog window. Notice how the factoryType specifies the fully-qualified name of the form factory class followed by the name of the client workflow assembly.

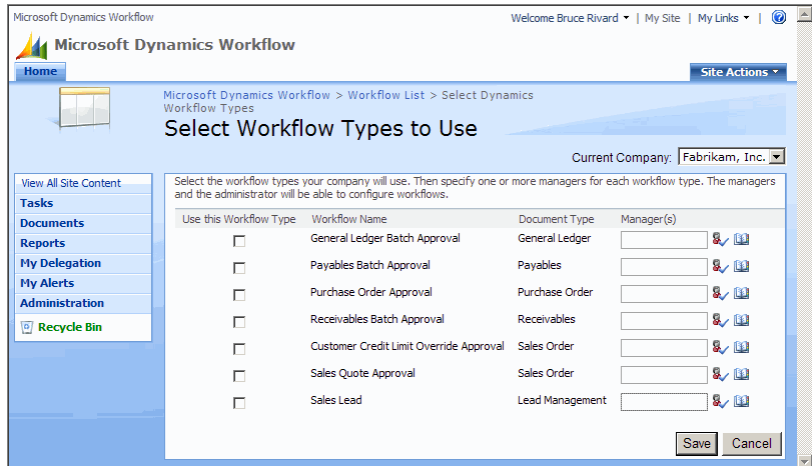
```
<add name="Workflow-SalesLeads" productId="3333" formId="22000"
factoryType="Microsoft.Dynamics.GP.Workflow.Samples.Client.LeadWorkflowFo
rmFactory,Microsoft.Dynamics.GP.Workflow.Samples.Client" />
<add name="Workflow-SampleWorkflowStatus" productId="3333" formId="22002"
windowId="22001"
factoryType="Microsoft.Dynamics.GP.Workflow.Samples.Client.WorkflowStatus
FormFactory,Microsoft.Dynamics.GP.Workflow.Samples.Client" />
```

Viewing the workflow

Verify the workflow is installed.

Before viewing your workflow, open the Start menu on the server where you installed your web service. Choose Run and enter "iisreset".

Open the Microsoft Dynamics Workflow web site. Click Administration. From the Workflow List, click Select Workflow Type to Use. When Select Workflow Types to Use opens, you should see your workflow in the list of Workflow Names.



Part 3: Extending an Existing Workflow

This portion of the documentation describes how to extend an existing workflow. If you have created an integration for one of the document types in Microsoft Dynamics GP that has a workflow defined for it, you may want the data for your integration to be accessible to the workflow. The following topics describes how to extend an existing workflow:

- [Chapter 11, “Designing a Workflow Extension,”](#) describes a workflow extension and the things you should consider when you are creating one.
- [Chapter 12, “Web Service Extension,”](#) explains how to create a web service extension so that additional data from your Microsoft Dynamics GP integration will be available for the workflow extension.
- [Chapter 13, “Workflow Extension Assembly,”](#) describes how to create the workflow extension assembly. This assembly makes the data from an integrating application available to be used with an existing workflow.
- [Chapter 14, “Deploying the Workflow Extension,”](#) explains how to create an installation helper application and use it to deploy a workflow extension.

Chapter 11: Designing a Workflow Extension

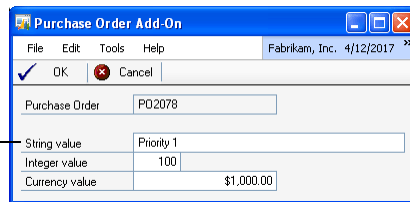
If you have created an integration for one of the document types in Microsoft Dynamics GP that has a workflow defined for it, you may want the data for your integration to be accessible to the workflow. This portion of the documentation describes how to create a workflow extension for an existing workflow. The workflow extension allows this additional data from the integrating application to be used in a workflow. The following topics are discussed:

- [Making data available to workflow](#)
- [Workflow extension assembly](#)
- [Data access](#)

Making data available to workflow

If you have an integration for Microsoft Dynamics GP that works with one of the documents that can have workflow defined for it, you may want the data for your integration to be available to that workflow. For instance, the following sample integration created with Dexterity manages additional data for each purchase order document in Microsoft Dynamics GP.

This integrating application tracks additional data for each purchase order document.



When you make data from your integration available to workflow, the data can be used by the workflow administrator as they set up the approval steps for the workflow.

Data made available by the workflow extension can be used when defining a workflow step.

Require approval when:

Select the situations when approval is required.

Choose to move to the next step when conditions for this step are not met. For example, when no users are required to approve this step you want the workflow to move to the next step to see if approval is required there.

Always

Only

Where is greater than (Remove)

And

Move to the next step if conditions aren't met to require approval for this step.

The additional data can also be made available in the business document summary that is included with each workflow notification.

Workflow extension assembly

The workflow extension assembly is a separate assembly that you will create and register with the Dynamics workflow. It defines the additional properties that you are making available to an existing workflow. It also defines the data that you want to have included in the summary document for the workflow notification.

Carefully consider what specific data values you want to make available to use with an existing workflow. Limit the number of values you add to those that make sense to use when defining steps for the workflow. A different set of data values may be appropriate for the summary document included with the workflow notification.

The sample purchase order integration included with the Microsoft Dynamics GP Workflow SDK makes the following additional properties available for use when creating workflow steps:

- SampleInt — an integer value
- SampleString – a string value
- SampleCurrency – a currency (decimal) value

You will learn about creating a workflow extension assembly in [Chapter 13, “Workflow Extension Assembly,”](#) which describes the process in detail.

Data access

Your Microsoft Dynamics GP integration will use the standard techniques described in the Microsoft Dynamics GP Integration Guide to store the additional data and keep it synchronized with the corresponding documents in the accounting system.

The workflow extension assembly you create must be able to access the data for your Microsoft Dynamics GP integration. Since each workflow uses the Dynamics GP web service to access data for the core Dynamics GP documents, any additional data for those documents will also be accessed through this web service. The web service extension capability of the Dynamics GP web service is used to make the additional data from your integration available with the core web service document accessed by Microsoft Dynamics Workflow.

The sample purchase order integration included with the Microsoft Dynamics GP Workflow SDK adds an Extension object to the ExtensionList for each purchase order document. The Extension object contains the data values the extension is making available to workflow.

You will learn about creating a web service extension that can be used with workflow in [Chapter 12, “Web Service Extension.”](#) Detailed information about using web service extensions for the Microsoft Dynamics GP web service can be found in the Microsoft Dynamics GP Web Service Programmer’s Guide.

Chapter 12: Web Service Extension

To make an integrating application's data accessible to a workflow, you must add the data as an extension for the corresponding document in the Microsoft Dynamics GP web service. The information provided in this documentation describes the essential steps required to add an integrating application's data to the corresponding document in the Dynamics GP web service.

In a full web service integration, you would likely implement additional web service events in addition to the "Retrieved" event that is used by your workflow extension to retrieve data for the workflow. Detailed information about using web service extensions for the Microsoft Dynamics GP web service can be found in the Microsoft Dynamics GP Web Service Programmer's Guide.

Information about creating a web service extension for use with a workflow extension is divided into the following sections:

- [Creating a Visual Studio project](#)
- [Defining the extension data](#)
- [Adding the web service event handler](#)
- [Building the web service extension](#)
- [Registering the web service extension](#)
- [Testing the web service extension](#)

Creating a Visual Studio project

The web service extension assembly is a Microsoft .NET assembly that you create using Visual Studio. The assembly works with the Web Services for Microsoft Dynamics GP installation to make additional data available with documents retrieved through the web service.

To create a web service extension assembly, complete the following steps:

1. Create a new project.

Open Visual Studio. From File menu, select File >> New >> Project. In the New Project window, select Visual C# as the project type. In Templates, select Class Library from the list of Visual Studio installed templates.

Enter a name for the web service extension assembly. Review the Location and Solution Name, and then click OK.

2. Add references to the Dynamics GP web service assemblies.

Add references to the following assemblies from the "Bin" folder for the Dynamics GP web service.

- Microsoft.Dynamics.Common
- Microsoft.Dynamics.Common.Types
- Microsoft.Dynamics.GP.BusinessLogic

These assemblies are typically found in this location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

3. Add namespace references.

Add **using** statements to provide convenient access to the classes and methods needed for the web service extension assembly. Include the following:

- Microsoft.Dynamics.Common
- Microsoft.Dynamics.GP
- System.Data.SqlClient
- System.Xml

4. Specify the name for the namespace.

In the code for the class library, specify the name for the namespace. Use a name that indicates who created the web service extension or what type of extended data is being made available. In the example included with the Workflow SDK, the namespace “POExtension” is used.

Defining the extension data

You will add a class defined in the web service extension code to indicate what additional data you want to add to the main document in the Dynamics GP web service. For this sample, three additional properties were added to a purchase order document. The following is the PurchaseOrderExtension class that defines these properties for the sample web service extension:

```
public class PurchaseOrderExtension
{
    private string sampleString;
    private int sampleInt;
    private decimal sampleCurrency;

    public PurchaseOrderExtension() { }

    public PurchaseOrderExtension(string StringValue, int IntValue,
    decimal CurrencyValue)
    {
        this.SampleCurrency = CurrencyValue;
        this.sampleInt = IntValue;
        this.sampleString = StringValue;
    }

    public decimal SampleCurrency
    {
        get { return sampleCurrency; }
        set { sampleCurrency = value; }
    }

    public int SampleInt
    {
        get { return sampleInt; }
        set { sampleInt = value; }
    }

    public string SampleString
    {
        get { return sampleString; }
        set { sampleString = value; }
    }
}
```

The `PurchaseOrderExtension` object will be serialized and added as an XML fragment for the Extension object in the Dynamics GP web service. The web service extension for your integration should follow this same basic pattern.

Adding the web service event handler

The web service event handler is a static class containing methods that respond to requests from the Dynamics GP web service each time a web service operation is performed. For the workflow extension, you must implement the “Retrieved” event handler. The event handler for the web service extension used with workflow must retrieve the integrating application’s data that corresponds to the main document. The event handler creates and populates the object defined in the previous section. Then it attaches that data to the Extension object. Finally, the Extension object is added to the collection of Extension objects that are included with the document.

The following is the event handler for the sample web service extension. Notice that the class is defined as static. The “Retrieved” static method will be called each time a purchase order document is retrieved from the web service. The event handler retrieves information from the “POPAddOn” table defined by the sample integration. This data is used to populate the `PurchaseOrderExtension` object, which is added to the Extensions collection and returned with the purchase order document.



The extension was given the name “POPAddOn” when it was added to the Extensions collection. This name will be needed when the workflow extension assembly is created.

```
public static class PurchaseOrderExtensionEventHandler
{
    public static void Retrieved(object sender, BusinessObjectEventArgs e)
    {
        PurchaseOrder purchaseOrder;
        Connection connection;

        if (e.BusinessObject.GetType() == typeof(PurchaseOrder))
        {
            purchaseOrder = (PurchaseOrder)e.BusinessObject;

            // Get the connection to the database for the current company
            connection = Connection.GetInstance();

            // The SQL command to retrieve the additional Purchase Order
            // information
            string selectCommand = "SELECT SampleString, SampleInteger,
            SampleCurrency FROM POPAddOn WHERE PONUMBER='" +
            purchaseOrder.Key.Id + "'";

            SqlDataAdapter adapter = new SqlDataAdapter(selectCommand,
            (SqlConnection)connection.GetConnection(
            e.Context.OrganizationKey));

            DataTable table = new DataTable();
            adapter.Fill(table);

            if (table.Rows.Count > 0)
            {
                // Get the data from the SQL result
```

```

PurchaseOrderExtension purchaseOrderExtension = new
PurchaseOrderExtension(
table.Rows[0].ItemArray[0].ToString(),
Convert.ToInt32(table.Rows[0].ItemArray[1]),
Convert.ToDecimal(table.Rows[0].ItemArray[2]));

// Build the Extension object to return from the web service
Extension poExtension = new Extension("POAddOn",
purchaseOrderExtension);

// Add the extension to the Purchase Order object
e.BusinessObject.Extensions.Add(poExtension);
}
}
}
}
}

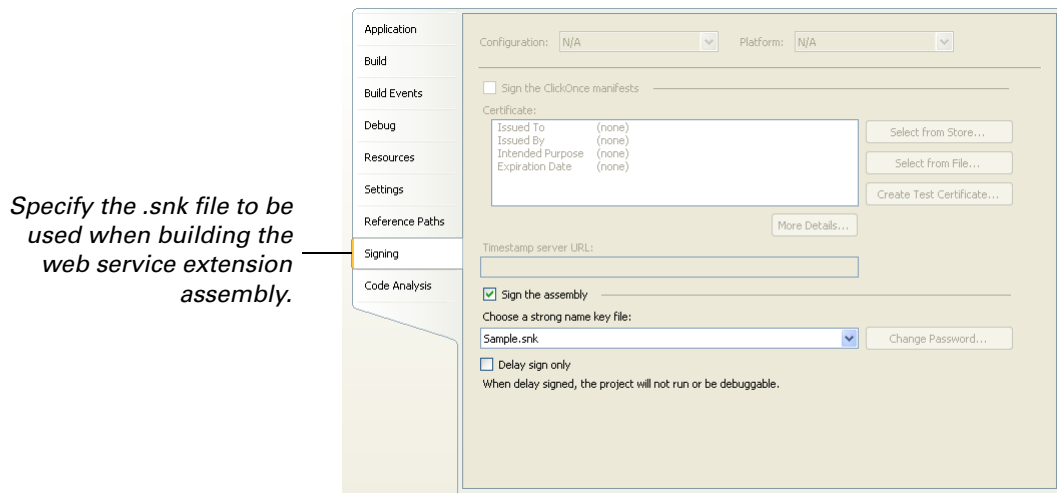
```

This completes the code required for the web service extension assembly.

Building the web service extension

An additional step is required before you build the web service extension assembly. When used with a workflow extension, a copy of the web service extension assembly must be added to the global assembly cache (GAC). To do this, the web service extension assembly must be compiled using a strong name key.

To specify a strong name key, display the properties for the web service extension project. Select the Signing tab and mark the “Sign the assembly” check box. You can generate a strong name key (.snk) file at the point, or use a .snk file you previously created.



It's a good idea to use the same .snk file for the projects in your web service extension and workflow extension.

You can now build the web service extension assembly. The completed web service extension assembly should be copied to the same location as the other assemblies for the Dynamics GP web service. Typically, this will be the following:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

Registering the web service extension

To have events for the Dynamics GP web service be run, they must be registered. The file named **BusinessObjectFile.config** contains the registrations for all of the additional events that are run by the Dynamics GP web service. This file is in XML format. It is located in the Dynamics GP web service “bin” folder, typically found in this location:

C:\Program Files\Microsoft Dynamics\GPWebServices\WebServices\Bin

You will add an entry to this file so that the “retrieved” event for the web service document to which you added additional data will retrieve that data.



Refer to the Microsoft Dynamics GP Web Service Programmer’s Guide for complete details about registering a web service extension.

DictionaryEntry elements

The **BusinessObjectFile.config** contains one `<DictionaryEntry>` element for each web service object that can have events registered for it. For convenience, the entries are in alphabetical order so they can be found more easily.

Not all objects have `<DictionaryEntry>` entries in this configuration file. For instance, the Batch object isn’t included in the default version of this file. If you are extending the batch workflows and have a web service extension for the Batch object, you will need to add a `<DictionaryEntry>` element for it. The easiest way to do this is by copying an existing element.

The `<Key>` for the `<DictionaryEntry>` element is the complete name of the business object. The complete name has the prefix `Microsoft.Dynamics.GP`, followed by the name of the object. The name of the object corresponds to the class name you see in the Dynamics GP Web Service Reference. As an example, the complete name for the purchase order object is: `Microsoft.Dynamics.GP.PurchaseOrder`.

Event elements

A `<DictionaryEntry>` element will have one or more `<Event>` elements, each describing an event for the business object. Each `<Event>` element has the following:

EventName The name of the event being registered. This corresponds to one of the events listed in the Microsoft Dynamics GP Web Service Programmer’s Guide.

EventHandlerType Specifies the type of event handler needed for the event.

SoftwareVendor Identifies who added the event.

Type The qualified name that indicates the namespace and static class containing the event handler method for the event.

StaticMethod The name of the static method that will be run for the event. The signature for this static method must be appropriate for the type of event.

Assembly The name of the web service extension assembly that contains the static method for the event.

Execute A boolean value that allows an event to be turned on or off. The value must be set to true for an event to be processed.

Example

The following example is the web service event registration for the sample web service extension. This web service extension works with purchase order documents, so entry in the `BusinessObjectFile.config` will be added to this section:

```
<DictionaryEntry>
  <Key xsi:type="xsd:string">Microsoft.Dynamics.GP.PurchaseOrder</Key>
  <Value xsi:type="BusinessObjectConfiguration">
```

The following is the registration for the “Retrieved” event for the purchase order document.

```
<Event>
  <EventName>Retrieved</EventName>
  <EventHandlerType>
    <Type>Microsoft.Dynamics.Common.BusinessObjectEventHandler</Type>
    <Assembly>Microsoft.Dynamics.Common</Assembly>
  </EventHandlerType>
  <EventHandler>
    <SoftwareVendor>MicrosoftDocumentation</SoftwareVendor>
    <Type>POExtension.PurchaseOrderExtensionEventHandler</Type>
    <StaticMethod>Retrieved</StaticMethod>
    <Assembly>POExtension</Assembly>
    <Execute>true</Execute>
  </EventHandler>
</Event>
```

Be sure that the static method and assembly names match the names that you used when you created the web service extension assembly.

Testing the web service extension

It’s a good idea to test the web service extension before you use it with a workflow extension. You can do this using a standard web service call to the Dynamics GP web service to retrieve a document for which you have added additional data. The data you added should be included in the Extensions collection for the document you retrieved.



You should use the “iisreset” command to be sure that your web service extension has been loaded into memory.

The following C# example tests the sample web service extension. It retrieves a purchase order document, and then looks through the Extensions collection to find out whether an extension with the name “POAddOn” has been added. If it has, the XML fragment containing the extension data is displayed.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using ExtensionTest.DynamicsGPService;

namespace ExtensionTest
{
  class Program
  {
```

```

static void Main(string[] args)
{
    CompanyKey companyKey;
    Context context;
    PurchaseTransactionKey purchaseOrderKey;
    PurchaseOrder purchaseOrder;
    Extension[] extensions;

    // Create an instance of the web service
    DynamicsGP wsDynamicsGP = new DynamicsGP();

    // Be sure the default credentials are used
    wsDynamicsGP.UseDefaultCredentials = true;

    // Create a context with which to call the web service
    context = new Context();

    // Specify which company to use (sample company)
    companyKey = new CompanyKey();
    companyKey.Id = (-1);

    // Set up the context object
    context.OrganizationKey = (OrganizationKey)companyKey;
    context.CultureName = "en-US";

    // Create a purchase transaction key to specify the purchase order
    purchaseOrderKey = new PurchaseTransactionKey();
    purchaseOrderKey.Id = "PO2074";

    // Retrieve the purchase order object
    purchaseOrder = wsDynamicsGP.GetPurchaseOrderByKey(
        purchaseOrderKey, context);

    // Get the extensions
    extensions = purchaseOrder.Extensions;

    // Look at the extension objects
    foreach (Extension ex in extensions)
    {
        if (ex.ExtensionId == "POAddOn")
        {
            // Display the purchase order object's extension inner XML
            MessageBox.Show(ex.DocExtension.InnerXml);
        }
    }
}
}
}

```


Chapter 13: Workflow Extension Assembly

The workflow extension assembly makes the data from an integrating application available to be used with an existing workflow. Information about creating a workflow extension assembly is divided into the following sections:

- [Creating a Visual Studio project](#)
- [Adding a resources file](#)
- [Adding the workflow extension code](#)
- [Building the workflow extension](#)

Creating a Visual Studio project

The workflow extension assembly is a Microsoft .NET assembly that you create using Visual Studio. The assembly works with the Microsoft Dynamics GP Workflow installation to make additional data available to an existing workflow.

To create a workflow extension assembly, complete the following steps:

1. Create a new project.

Open Visual Studio. From File menu, select File >> New >> Project. In the New Project window, select Visual C# as the project type. In Templates, select Class Library from the list of Visual Studio installed templates.

Enter a name for the workflow extension assembly. Review the Location and Solution Name, and then click OK.

2. Add references to the required assemblies.

Add references to the following assemblies used for Dynamics GP Workflow.

- Microsoft.Dynamics.Common
- Microsoft.Dynamics.Common.Types
- Microsoft.Dynamics.Workflow
- Microsoft.Dynamics.Workflow.Controls

These assemblies are typically found in this location:

C:\Program Files\Microsoft Dynamics\Workflow

Add references to the following additional assemblies used for Dynamics GP Workflow:

- Microsoft.Dynamics.Workflow.Common
- Microsoft.Dynamics.GP.WebServices.Proxy

These assemblies are typically found in the “bin” folder of the virtual directory used by the Workflow web service:

C:\Inetpub\wwwroot\wss\VirtualDirectories\port#\bin

You will need to know the port number you are using for the virtual directory that the Workflow web service is installed into.

Finally, add a reference to the web service extension assembly containing the class that defines the extended data. For the sample web service extension this is a reference to the POExtension assembly.

3. Add namespace references.

Add **using** statements to provide convenient access to the classes and methods needed for the workflow extension assembly. Include the following:

- Microsoft.Dynamics.Workflow
- Microsoft.Dynamics.Workflow.Common
- Microsoft.Dynamics.Workflow.Controls
- Microsoft.Dynamics.GP.Proxy
- System.Globalizaton
- System.Reflection

You will also want to add a **using** statement that references the web service extension assembly. For the sample workflow extension, this is a reference to the POExtension assembly.

4. Save and close the solution.

Save all of the files and close the solution.

5. Manually edit the project file.

To make the workflow extension assembly work properly, you must manually edit the project file used for the extension. Use a text editor such as Notepad to open the project (.csproj) file you just created.

Locate the <RootNamespace> element, and remove any value from it so that it appears in the file as:

```
<RootNamespace></RootNamespace>
```

Save the changes to the file and then re-open the solution in Visual Studio.

Adding a resources file

The names of the values made available to the workflow through the workflow extension assembly are stored in a resources file. To add a resource file to the project, complete the following procedure.

1. Add a resources file.

Choose Add Component from the Project menu. In the list of templates, choose Resources File.

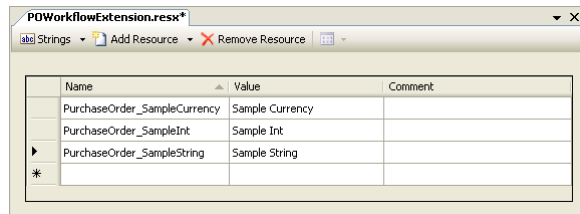
2. Name the resources file.

The resources file name must exactly match the name of the workflow extension assembly (with exception of the file extension). For example, the sample workflow extension assembly is named POWorkflowExtension.dll, so the resources file is named POWorkflowExtension.resx.

Click Add to add the file to the project.

3. Add content to the resources file.

Open the resources file and add an entry for each data item you want to make available to workflow through the workflow extension. The Name uniquely identifies the item, while the Value indicates the name that will be displayed for the item. The following illustration shows the resources file defined for the sample workflow extension.



Save the changes to the resources file.

Adding the workflow extension code

The code for the workflow extension is defined in a single public class. You will implement three handler methods in this class to do the following:

- Return available properties from your workflow extension.
- Return the base business object with the additional information added by the web service extension.
- Return information about the new properties you are making available, for use in a summary document for the business object.

The main class will look like the following example:

```
public class WorkflowExtension
{
}

```

GetAvailableProperties

This method returns all of the property values you want your workflow extension to make available to the workflow. These properties will be available to use when defining workflow steps.

The property definitions use a GUID value to define which extension defined them. You will need to generate your own GUID value for the properties that you are adding. Do this using a tool such as GUIDGen, which is included with Visual Studio.

The following example is the GetAvailableProperties method defined for the sample workflow extension. It defines a GUID that will be used to identify which extension added the additional properties. Each new property must include the following:

- The owner GUID
- A string value specifying how the property is accessed. Typically this is the property name in the class defined for the web service extension. For some data types, such as decimals, it can include a modifier to return just the value.
- The display name for the property. This value is retrieved from the resources file defined earlier.

The `GetAvailableProperties` method uses a private helper method named `GetString()` to retrieve information from the resource file. This helper method is also shown below.

```
public static void GetAvailableProperties(object sender,
GetAvailablePropertiesEventArgs args)
{
    Guid ownerId = new Guid("1D4F9BFB-2138-4794-9F8A-CB87BD346379");

    // Add the SampleString property
    args.Properties.Add(new StringFilterableProperty(ownerId, "SampleString",
GetString("PurchaseOrder_SampleString", args.CultureInfo)));

    // Add the SampleCurrency property
    args.Properties.Add(new NumberFilterableProperty(ownerId,
"SampleCurrency", GetString("PurchaseOrder_SampleCurrency",
args.CultureInfo)));

    // Add the SampleInt property
    args.Properties.Add(new NumberFilterableProperty(ownerId, "SampleInt",
GetString("PurchaseOrder_SampleInt", args.CultureInfo)));
}

private static string GetString(string resourceId, CultureInfo cultureInfo)
{
    return Microsoft.Dynamics.Common.ResourceHelper.GetString(resourceId,
Assembly.GetAssembly(typeof(WorkflowExtension)), cultureInfo);
}
```

GetBusinessObject

This method returns the base business object along with the additional information added by the web service extension. This method also uses a GUID value to identify which workflow extension is adding to the workflow. You can use the same GUID value that you created for the additional properties.

The following is the `GetBusinessObject` method for the sample workflow extension. It is adding to the Purchase Order workflow, so it is retrieving the ID of the workflow by using the workflow name. Use the following names when looking up the workflow:

- Dynamics GP Purchase Order Approval Workflow
- Dynamics GP Sales Quote Approval Workflow
- Dynamics GP Customer Credit Limit Override Approval Workflow
- Dynamics GP Payables Batch Approval Workflow
- Dynamics GP Receivables Batch Approval Workflow
- Dynamics GP General Ledger Batch Approval Workflow

The `GetBusinessObject` method uses a private helper method named `GetPOExtension` to retrieve the additional purchase order extension information that is being made available to the workflow. This private method is also shown below.

```
public static void GetBusinessObject(object sender,
GetBusinessObjectEventArgs args)
{
    // Need to define ownership of the object added
    Guid ownerId = new Guid("1D4F9BFB-2138-4794-9F8A-CB87BD346379");
    Guid primaryObjectId = DynamicsWorkflow.FindByName(
"Dynamics GP Purchase Order Approval Workflow").Key.Id;
    PurchaseOrder purchaseOrder = (PurchaseOrder)args.Objects
[primaryObjectId];
    args.Objects.Add(ownerId, GetPOExtension(purchaseOrder));
}

private static POExtension.PurchaseOrderExtension
GetPOExtension(PurchaseOrder purchaseOrder)
{
    // Retrieve the Purchase Order Extension from the purchase order document
    foreach (Extension extension in purchaseOrder.Extensions)
    {
        if (extension.ExtensionId == "POAddOn")
        {
            Microsoft.Dynamics.Common.Extension e = new
Microsoft.Dynamics.Common.Extension();
            e.ExtensionId = "POAddOn";
            e.DocExtension = extension.DocExtension;
            e.Deserialize(typeof(POExtension.PurchaseOrderExtension));
            return (POExtension.PurchaseOrderExtension)e.Obj;
        }
    }
    return null;
}
```

GetSummaryInformation

The `GetSummaryInformation` method returns display information for the additional properties being added to a workflow. This information will be included in summary documents for the workflow, which are available with the workflow notification. The values defined in the `GetSummaryInformation` method are the only ones added to the summary information for the workflow document.

The following is the `GetSummaryInformation` method defined in the sample workflow extension. It adds the integer and string values from the workflow extension to the summary information for the purchase order document. The `GetSummaryInformation` method uses the same `GetString()` helper method that the `GetAvailableProperties` method does.

```
public static void GetSummaryInformation(object sender,
GetSummaryInformationEventArgs args)
{
    POExtension.PurchaseOrderExtension poExt = GetPOExtension(
(PurchaseOrder)args.BusinessObject);
    int priority = args.SummaryInformation.SummaryInformation.Count + 1;
```

```

args.SummaryInformation.SummaryInformation.Add(new
SummaryInformation("SampleInt", poExt.SampleInt.ToString(), priority++,
GetString("PurchaseOrder_SampleInt", args.CultureInfo));

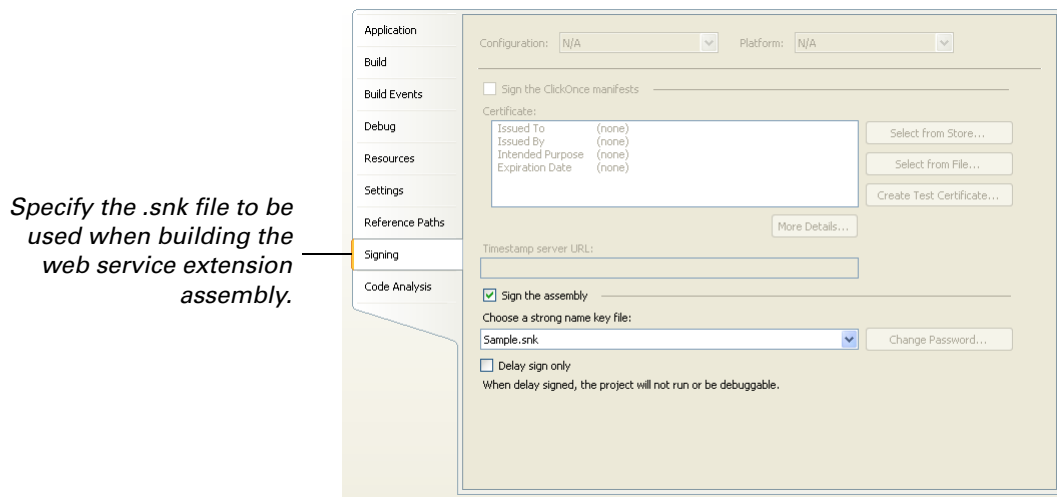
args.SummaryInformation.SummaryInformation.Add(new
SummaryInformation("SampleString", poExt.SampleString, priority++,
GetString("PurchaseOrder_SampleString", args.CultureInfo));
}

```

Building the workflow extension

An additional step is required before you build the workflow extension assembly. The workflow extension assembly must be added to the global assembly cache (GAC). To do this, the workflow extension assembly must be compiled using a strong name key.

To specify a strong name key, display the properties for the workflow extension project. Select the Signing tab and mark the “Sign the assembly” check box. You can generate a strong name key (.snk) file at the point, or use a .snk file you previously created.



It's a good idea to use the same .snk file for the projects in your web service extension and workflow extension.

You can now build the workflow extension assembly.

Chapter 14: Deploying the Workflow Extension

Deploying a workflow extension assembly is a multi-step process. To make the process easier, you will create a “helper” application that will register your workflow extension with the Microsoft Dynamics GP Workflow. Information about deploying a workflow extension is contained in the following sections:

- [Web service extension](#)
- [Adding assemblies to the global assembly cache](#)
- [Installation helper](#)
- [Registering the workflow extension](#)

Web service extension

The web service extension assembly should be deployed and operating properly before you attempt to deploy the workflow extension. Refer to [Building the web service extension](#) on page 182 and [Registering the web service extension](#) on page 183 for details about deploying the web service extension that you will be using with your workflow extension.

Adding assemblies to the global assembly cache

Both the workflow extension assembly and the web service extension assembly must be added to the global assembly cache (GAC). You can add your assembly to the global assembly cache by dragging the .dll file to the appropriate folder, or by using the Gacutil Tool.

To add the file by dragging, open an instance of Microsoft Windows Explorer. In the new Windows Explorer, open the folder C:\Windows\Assembly. Drag the workflow extension assembly and the web service extension assembly to the Assembly folder.

To use the Gacutil Tool, use the following procedure.

1. Open the Visual Studio Command Prompt.

In the Program Group for Microsoft Visual Studio, point to Visual Studio Tools and choose Visual Studio Command Prompt. A command prompt will be displayed.

2. Specify the working location.

Using the command prompt, change the working location to the folder where you have created the workflow extension assembly or the web service extension assembly.

3. Use the Gacutil Tool to load the assembly in the global assembly cache.

Use the following command to add your server workflow assembly to the global assembly cache:

```
gacutil -I <Complete Path>
```

Replace <Complete Path> with the complete path and filename of the assembly you are adding to the global assembly cache.

Installation helper

The registration for a workflow extension is complex. To help with this process, you will create an “installation helper.” This is a command-line application that you will use to register the workflow extension.

To create the installation helper application, complete the following steps:

1. Create a new project.

Open Visual Studio. From File menu, select File >> New >> Project. In the New Project window, select Visual C# as the project type. In Templates, select Console Application from the list of Visual Studio installed templates.

Enter a name for the application. Review the Location and Solution Name, and then click OK.

2. Add references to the required assemblies.

Add a reference to the following assembly used for Dynamics GP Workflow:

- Microsoft.Dynamics.Workflow

This assembly is typically found in the location:

C:\Program Files\Microsoft Dynamics\Workflow

Also, add a reference to the workflow extension assembly that you want to register with workflow.

3. Add a namespace reference.

Add a **using** statement to provide convenient access to the classes and methods in the following assembly:

- Microsoft.Dynamics.Workflow

4. Add the Register method.

The Register method is a static method that performs the registration for the workflow extension. Add this method to the main class for the command line application. The following example shows this method used to register the sample workflow extension.

```
static void Register()
{
    WorkflowEventManager.RegisterStepConditionExtension(
        DynamicsWorkflow.FindByName("Dynamics GP Purchase Order Approval
        Workflow"),
        "Sample",
        typeof(WorkflowExtension));
}
```

Notice that the workflow the extension is to be used with is retrieved based on its name. Use one of the following values to specify the workflow you are extending:

- Dynamics GP Purchase Order Approval Workflow
- Dynamics GP Sales Quote Approval Workflow
- Dynamics GP Customer Credit Limit Override Approval Workflow
- Dynamics GP Payables Batch Approval Workflow
- Dynamics GP Receivables Batch Approval Workflow
- Dynamics GP General Ledger Batch Approval Workflow

The second parameter to the RegisterStepConditionExtension method is a string that identifies who is registering the workflow extension.

5. Add code to the Main method.

In the Main method for the application, add a call to the Register() method you just created.

```
static void Main(string[] args)
{
    Register();
}
```

6. Build the application.

Choose Build Solution to build the installation application.

7. Add an application configuration file.

To work properly, the installation helper application must have an application configuration (app.config) file that specifies where a workflow configuration file is located.

You can use Visual Studio to add an application configuration file. To do this, choose Add New Item in Project menu. In the list of available templates, choose Application Configuration File. Name the file App.config and click Add.

Use the editor in Visual Studio to make the App.config file look like the following:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="BusinessObjectsConfigurationPath" value =
      "C:\Inetpub\wwwroot\wss\VirtualDirectories\10072\bin\
      DynamicsWorkflowEvents.config" />
  </appSettings>
</configuration>
```

The standard path to the Workflow web service is shown in this example. You will need to update the path to use the port number (likely something other than 10072) that is being used for the Workflow web service.

Registering the workflow extension

To register your workflow extension, locate and run the installation helper application you created. This application will locate the DynamicsWorkflowEvents.config file, and add a large entry to it that defines the necessary events for your workflow extension. This config file can be found in the location specified by the App.config file used by the installation helper.

Part 4: Dynamics Workflow Web Service

This portion of the documentation describes how to connect to and use the Dynamics Workflow web service from an external application. The following topics are discussed:

- [Chapter 15, “Connecting to the Web Service.”](#) explains how to create a connection to the Dynamics Workflow web service.
- [Chapter 16, “Using the Web Service.”](#) provides techniques and examples that will be helpful as you use the Dynamics Workflow web service.

Chapter 15: Connecting to the Web Service

To perform actions with the Microsoft Dynamics Workflow web service from an external application, your application must create a connection to it.



If you are accessing the Dynamics Workflow web service through the client workflow controller, be sure to use the client “helper” assembly to access the web service, rather than the techniques described here.

The following topics describe how to connect to this web service:

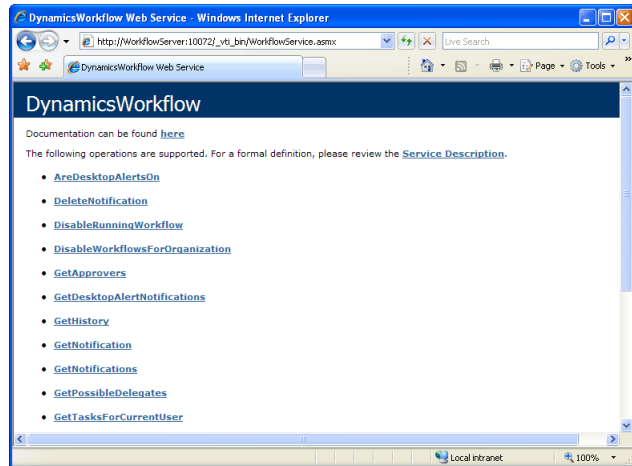
- [Web service URL](#)
- [Workflow web service proxy](#)
- [Web service namespace](#)
- [Creating a web service instance](#)

Web service URL

The Dynamics Workflow web service is accessible through a standard web browser. Simply supply the URL to the **WorkflowService.asmx** file that defines the web service interface. This web service is installed into the same location as several web services used by Office SharePoint Server. By default, the URL to this file will be:

`http://machine_name:port/_vti_bin/WorkflowService.asmx`

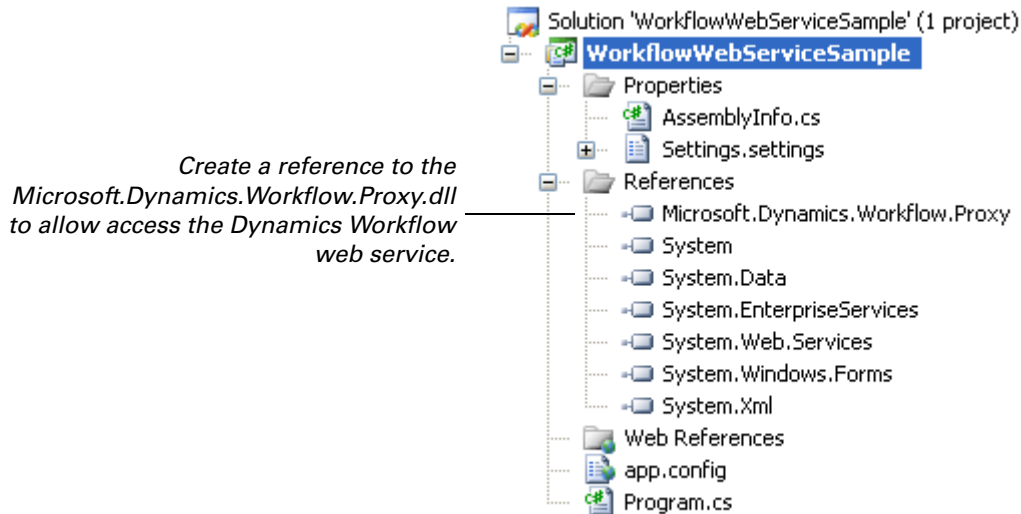
You will need to substitute the machine name and the port number for your Office SharePoint Server installation. When you view this file, you will see a list of the operation (web methods) available in the service. This is shown in the following illustration.



Click the individual operations to see the SOAP messages that are sent when the operations are performed and results are returned. For most web services, you would be able to click Service Description to see the entire Web Service Description Language (WSDL) file that completely describes the web service. Because the Dynamics Workflow web service is installed in the same location as several Office SharePoint Server web services, security restrictions will prevent the WSDL from being displayed. This means the best way to access the Dynamics Workflow web service is to use a proxy assembly, which is described in the next section.

Workflow web service proxy

To access the Dynamics Workflow web service, you will use a proxy assembly. This assembly is named **Microsoft.Dynamics.Workflow.Proxy.dll**, and is found in the Microsoft Dynamics GP installation folder. You will create a reference in your Visual Studio project to access this proxy assembly. This will provide access to a set of methods that closely match those described in the Dynamics Workflow web service WSDL document.



You will need to include the web service proxy assembly with your application that accesses the Dynamics Workflow web service.

Web service namespace

In Visual Studio, the classes defined in the Dynamics Workflow proxy assembly will be added to a separate namespace in the project. To make it easier to reference the classes, methods, and enumerations from the Dynamics Workflow web service, you will want to add this namespace to your application code. For instance, the following C# statement will add this namespace to the current application.

```
using Microsoft.Dynamics.Workflow.Proxy;
```

Adding the **using** statement will keep you from having to fully-qualify the classes, methods, and enumerations you refer to in the Dynamics Workflow web service proxy.

Creating a web service instance

After you have created a reference to the Dynamics Workflow proxy assembly, you will create an instance of the service so you can access the web service methods. The "DynamicsWorkflow" class in the proxy assembly represents the base web service. You will create an instance of this class that will provide access to the web service methods.

The following example shows the C# code required to create an instance of the Dynamics Workflow web service.

```
// Create an instance of the web service
DynamicsWorkflow wsWorkflow = new DynamicsWorkflow();
```

The web service instance also provides access to properties that control how the web service is called. For instance, when accessing the Dynamics Workflow web service you can specify that the current user's login credentials will be used for the web service call. The following C# code shows how this is done for a project created in Visual Studio.

```
// Be sure that default credentials are being used
wsWorkflow.UseDefaultCredentials = true;
```

The web service instance also specifies the URL of the web service to be accessed. You will use the URL of the Dynamics Workflow web service for your workflow installation. The following C# code shows how the URL is specified.

```
// Specify the URL used to access the Workflow web service
wsWorkflow.Url = "http://WorkflowServer:10072/_vti_bin/WorkflowService.asmx";
```

You will need to substitute the server name and port number that are appropriate for your workflow installation.

Chapter 16: Using the Web Service

This portion of the documentation provides information you will need when using the Dynamics Workflow web service. The following topics are discussed:

- [Dynamics Workflow Web Service Reference](#)
- [Dynamics Workflow web service example](#)
- [Business object keys](#)
- [BusinessObjectKey reference](#)

Dynamics Workflow Web Service Reference

The Dynamics Workflow Web Service Reference is an online help file that provides detailed information about the commonly-used methods, classes, and enumerations available in the Dynamics Workflow web service proxy assembly. Use this comprehensive reference as you learn about the object model for the Dynamics Workflow web service. The links in the help file make it easy to browse through the properties of an object and see the other objects related to it.

Dynamics Workflow web service example

The following is a basic example that demonstrates how to access the Dynamics Workflow web service through the web service proxy assembly. This C# example retrieves all of the workflow tasks for the current user and displays them in a dialog. Notice how the example creates an instance of the web service from the proxy assembly, and then specifies the URL of the Dynamics Workflow web service to access.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using Microsoft.Dynamics.Workflow.Proxy;

namespace WorkflowWebServiceSample
{
    class Program
    {
        static void Main(string[] args)
        {
            CompanyKey companyKey;
            WorkflowTask[] tasks;

            // Create an instance of the web service
            DynamicsWorkflow wsWorkflow = new DynamicsWorkflow();

            // Specify the URL used to access the Workflow web service
            wsWorkflow.Url =
                "http://WorkflowServer:10072/_vti_bin/WorkflowService.asmx";

            // Be sure that default credentials are being used
            wsWorkflow.UseDefaultCredentials = true;

            // Create the company key for the sample company
            companyKey = new CompanyKey();
            companyKey.Id = (-1);
        }
    }
}
```

```

// Get tasks for this user, including completed tasks
tasks = wsWorkflow.GetTasksForCurrentUser(companyKey, true);

// Display the list of tasks
StringBuilder taskList = new StringBuilder();
foreach (WorkflowTask t in tasks)
{
    taskList.AppendLine(t.AssignedTo + " -- " + t.Description);
}
MessageBox.Show(taskList.ToString());
}
}
}

```

Business object keys

Several of the methods available for the Dynamics Workflow web service require identifying a specific document in Microsoft Dynamics GP for which an action is being performed. The various types of documents will have different keys that are used to uniquely identify each document. The Dynamics Workflow web service has the **BusinessObjectKey** class, which is a general-purpose class for specifying the key values for any document type in Microsoft Dynamics GP.

A **BusinessObjectKey** is composed of one or more **KeyPart** objects, each of which represent one segment of the key for the business document. Each **KeyPart** object has the following properties:

Name This identifies the segment of the key. For Microsoft Dynamics GP documents, its value will typically be the name of the database column that the key segment value corresponds to.

PartValue This is the actual value of the key segment. Its datatype will vary, depending on the type of data used for the key segment. The Dynamics Workflow web service defines several **KeyPart** classes, one for each datatype that can be used for a key segment. For example, the **KeyPartOfString** class is used for key segments that have a string datatype.

An appropriate **KeyPart** object must be created and its value specified for each segment of the key that uniquely identifies a Microsoft Dynamics GP document. The **KeyPart** objects are then assembled by adding them to the **BusinessObjectKey** object that will be used for the Dynamics Workflow web service call.

For example, a Sales Order Processing document in Microsoft Dynamics GP uses two key segments to uniquely identify the document. The first is the Document Number, which corresponds to the "SOPNUMBE" column in the table that stores the documents. The second is the Document Type, which specifies what type of sales document (quote, invoice, etc.) is being referred to. This segment corresponds to the "SOPTYPE" column in the table that stores the documents.

The following C# example shows how the `BusinessObjectKey` for the Sales Quote QT4005 is created. Notice that `KeyPart` segments are created for each segment of the key, and then assembled into the `BusinessObjectKey`.

```
BusinessObjectKey key;
KeyPart[] keyParts;
KeyPartOfString docNumber;
KeyPartOfString docType;

// Build the key for the Sales Quote QT4005
key = new BusinessObjectKey();

// Build the key parts
// Document number
docNumber = new KeyPartOfString();
docNumber.Name = "SOPNUMBE";
docNumber.PartValue = "QT4005";

// Document type
docType = new KeyPartOfString();
docType.Name = "SOPTYPE";
docType.PartValue = "Quote";

// Assemble the key parts
keyParts = new KeyPart[2];
keyParts[0] = docNumber;
keyParts[1] = docType;

// Add the key parts to the key
key.KeyParts = keyParts;
```

BusinessObjectKey reference

The following tables describe the `BusinessObjectKey` objects that are needed to refer to specific Microsoft Dynamics GP documents that can be used with workflow. Use this information if you need to manually create a `BusinessObjectKey` object to refer to a document.



In most cases, you won't need to manually create a `BusinessObjectKey` object. It will be available as a property in an existing workflow object, such as `WorkflowTask`.

General Ledger Batch document

A General Ledger Batch document is identified by the following key segments:

Segment name	Type	Value
BACHNUMB	String	The batch number
BCHSOURC	String	"GL_Normal" for general entry "GL_Clearing" for clearing entry
CREATEDDATE	DateTime	A datetime value composed of the date portion of the Created Date (CREATDDT) column and the time portion of the Time (TIME1) column for the row in the SY00500 table.

Payables Batch document

A Payables Batch document is identified by the following key segments:

Segment name	Type	Value
BACHNUMB	String	The batch number
BCHSOURC	String	"PM_Trxent" for payables transaction entry "XPM_Cchecks" for computer checks "PM_Payment" for manual payment
CREATEDDATE	DateTime	A datetime value composed of the date portion of the Created Date (CREATDDT) column and the time portion of the Time (TIME1) column for the row in the SY00500 table.

Purchase Order document

A Purchase Order document is identified by the following key segment:

Segment name	Type	Value
PoNumber	String	The purchase order document number

Receivables Batch document

A Receivables Batch document is identified by the following key segments:

Segment name	Type	Value
BACHNUMB	String	The batch number
BCHSOURC	String	"RM_Sales" for transaction entry "RM_Cash" for cash receipts
CREATEDDATE	DateTime	A datetime value composed of the date portion of the Created Date (CREATDDT) column and the time portion of the Time (TIME1) column for the row in the SY00500 table.

Sales Order documents

A Sales Order document is identified by the following key segments:

Segment name	Type	Value
SOPNUMBE	String	The sales order document number
SOPTYPE	String	The sales order document type. The value will be one of the following: Quote Order Invoice Return Backorder FulfillmentOrder

Appendix

The following appendixes are provided:

- [Appendix A, “Troubleshooting,”](#) describes how to handle problems that prevent your integration from working properly.
- [Appendix B, “Debugging,”](#) provides advice on debugging the client workflow assembly and the server workflow assembly.
- [Appendix C, “Changing Passwords,”](#) describes a process for updating your workflow and SharePoint service when a password change occurs.

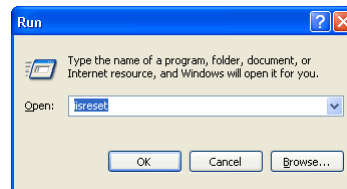
Appendix A: Troubleshooting

You may encounter problems that prevent your workflow integration from working properly. The following is a list of solutions to some problems you may encounter.

- [Resetting the workflow server](#)
- [Verifying authorization](#)

Resetting the workflow server

While you develop and test your server and client workflow assemblies, the Workflow server may perform slowly or display unexpected results on the Dynamics GP client. To restore the workflow server, open the Start menu on the Workflow server and choose Run. Enter `iisreset` and click OK.



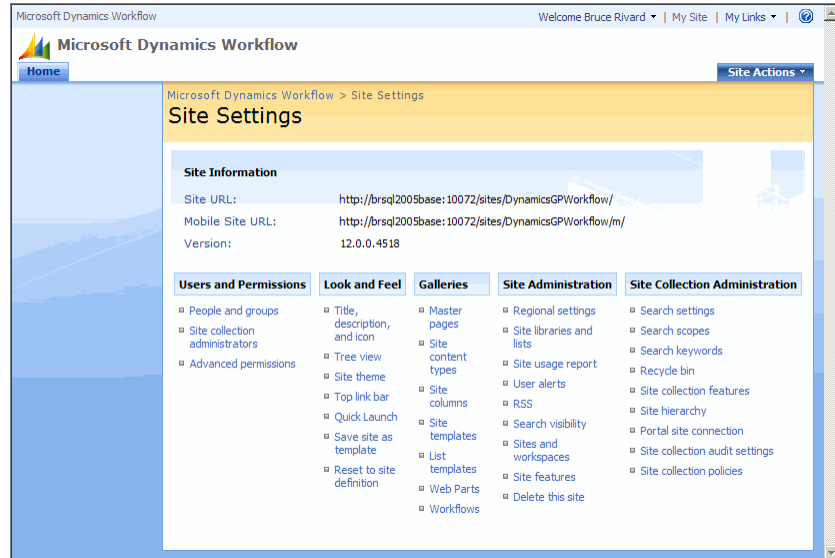
The “iisreset” command flushes and rebuilds the caches used by MOSS and the Dynamics Workflow web service. Rebuilding these caches can eliminate the slow response times and errors.

Verifying authorization

If you encounter errors that indicate you are not authorized to view a document or perform an action, ensure the login you are using has sufficient privileges. To view workflow permissions, you need to review settings in Microsoft Dynamic GP Workflow, SharePoint server, and Dynamics Security Service.

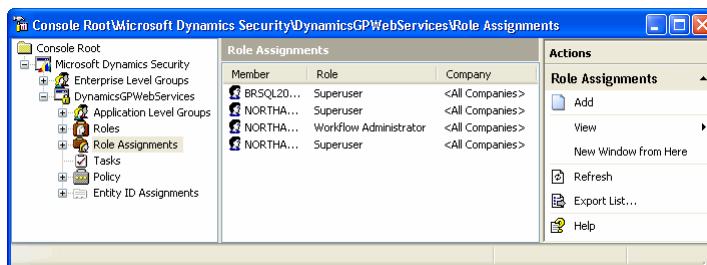
- To view Microsoft Dynamics GP privileges, log in with the credentials of the Workflow administrator. Use Microsoft Internet Explorer to open the Microsoft Dynamics Workflow site. Click Administration to view your active workflows. To view the configuration of a specific workflow, click its name in the Workflow Name column. Review the Configure Workflow settings and the Workflow Step configuration to ensure the your login is identified as an originator, an approver, or has been give read-only access to workflow information. If you need to add your login, update the Configuration Settings and click Save.

- To view SharePoint settings, log in with the credentials of the Workflow administrator. Use Microsoft Internet Explorer to open the Microsoft Dynamics Workflow site. Click Site Actions >> Site Settings. In Users and Permissions, click People and Groups. Check that People and Groups includes your login identity. If it does not, click New and add it to the list.



- To view Dynamics Security Service permissions, open the Dynamics Security Console. Expand the DynamicsGPWebService node and select Role Assignments. Be sure your login is listed as a Member. If your login is not shown, click Add. Also be sure the role assigned to your login provides web service access to your workflow document.

For additional information about managing access to workflow, refer to the **Microsoft Dynamics GP Workflow Administrator’s Guide**.



Appendix B: Debugging

As you develop workflow integrations it is often useful to use Visual Studio to debug your assemblies. The following sections describe how to use the Visual Studio debugger to debug your client or server workflow assembly.

- [Debugging the client workflow assembly](#)
- [Debugging the server workflow assembly](#)

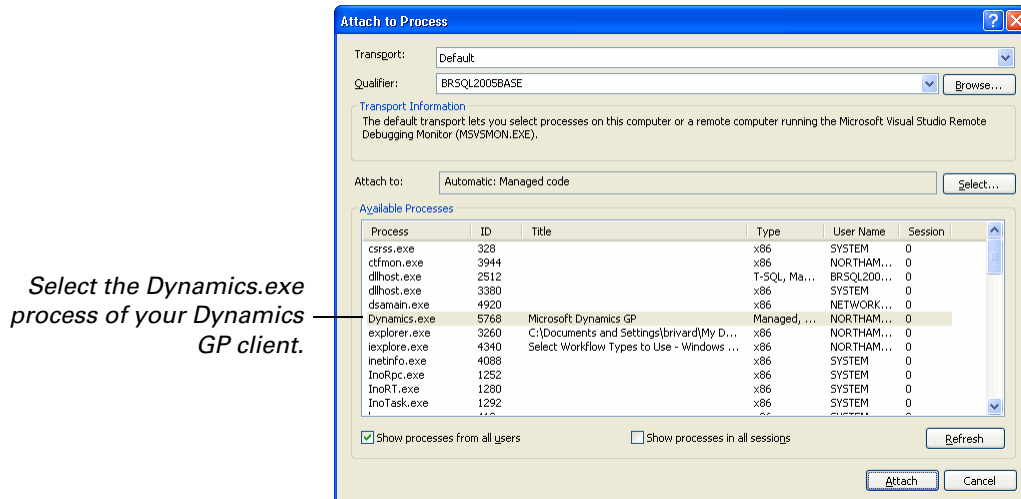
Debugging the client workflow assembly

While developing the client workflow assembly or to determine the cause of unexpected results, it is often useful to see your client assembly running in the Visual Studio debugger.

The following procedure assumes you have loaded the Microsoft Dynamics GP client on the same machine where you are running Visual Studio. To debug the client workflow assembly with Visual Studio, complete the following steps:

- 1. Open your client workflow assembly project with Visual Studio.**
To debug, you need to work with a debug build of your client workflow assembly. The debug build includes a .pdb file for your assembly. The .pdb file contains the debug symbols that Visual Studio requires.
- 2. Set a breakpoint where you would like to begin viewing code execution**
In Visual Studio, open the file you wish to debug. Place the cursor on a line where you would like to stop the code execution. Right-click and choose Breakpoint >> Insert Breakpoint from the menu.
- 3. Start the Microsoft Dynamics GP client.**
Start the Microsoft Dynamics GP client and login to the company you would like to test.
- 4. Attach the Visual Studio debugger to the Microsoft Dynamics GP client process.**
From the Visual Studio Debug menu, choose Attach to Process. In the Attach to Process window, set Transport to Default. In Qualifier, enter the name of the machine running the Microsoft Dynamics GP client.

In the list of Available Processes, select Dynamics.exe. Click Attach. The debugger will attach to the process and load the symbol file. If the symbol file cannot be found, the debugger will prompt you to locate the .pdb file.



5. Use the Microsoft Dynamics GP client to initiate a workflow action.

Perform an action with the Dynamics GP client that initiates workflow. When the assembly reaches the point where you have set the breakpoint, the Visual Studio debugger will stop execution.

6. Step through your code.

Use the F10 key to manually control the execution of your code. Use the F11 key to step into method calls that are within your client workflow assembly.

Use the debugger's Autos, Locals, and Watch windows to monitor the value of individual objects and data members. This information can help you to pinpoint the cause of unexpected results.

Debugging the server workflow assembly

While developing the server workflow assembly or to determine the cause of unexpected results, it is often useful to see your server assembly running in the Visual Studio debugger.

The following procedure assumes you have loaded Visual Studio on your Workflow server. To use the Visual Studio debugger with the server workflow assembly complete the following steps:

1. Open your server workflow assembly project with Visual Studio.

To debug, you need to work with a debug build of your server workflow assembly. The debug build includes a .pdb file for your assembly. The .pdb file contains the debug symbols that Visual Studio requires.

2. Set a breakpoint where you would like to begin viewing code execution.

In Visual Studio, open the file you wish to debug. Place the cursor on a line where you would like to stop the assembly. Right click and choose Breakpoint >> Insert Breakpoint from the menu.

3. Be sure the workflow process is running.

Use the Microsoft Dynamics GP client to initiate a workflow action. This will ensure the workflow service is running on the server.

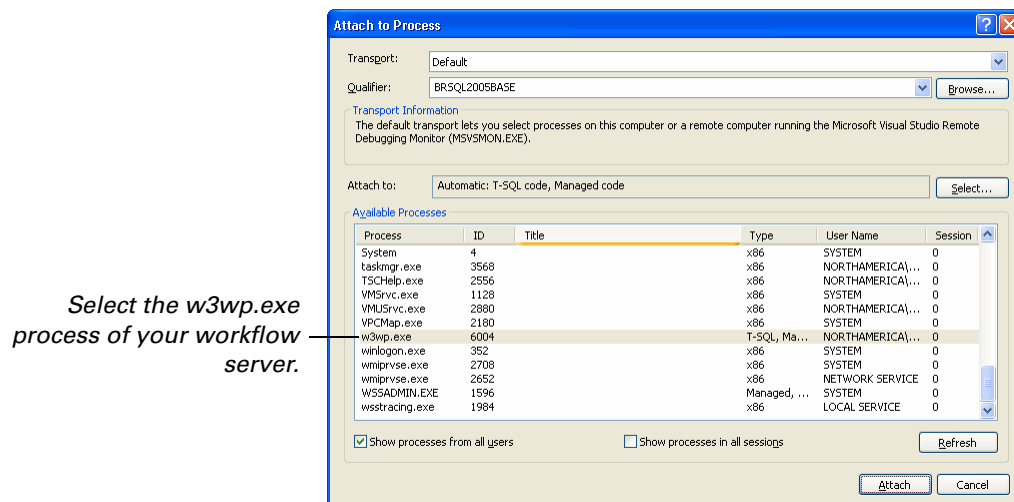
4. Attach the Visual Studio debugger to the workflow server.

From the Visual Studio Debug menu, choose Attach to Process. In the Attach to Process window, set Transport to Default. In Qualifier, select the name of the machine where you are running the workflow server.



The Visual Studio debugger also allows you to do debugging on a remote server. However, you must enable remote debugging on the server before attempting to attach to the workflow server process. You also need to ensure the server is running a debug assembly and you have access to the .pdb file for that assembly.

In the list of Available Processes, select w3wp.exe. Click Attach. The debugger will attach to the process and load the symbol file. If the symbol file cannot be found, the debugger will prompt you to locate the appropriate .pdb file.

**5. Use the Microsoft Dynamics GP client to initiate a workflow action.**

Perform a Dynamics GP client action that uses your server workflow assembly. When the assembly reaches the point where you have set the breakpoint, the Visual Studio debugger will stop execution.

6. Step through your code.

Use the F10 key to manually control the execution of your server code. Use the F11 key to step into method calls that are within your server assembly.

Use the debugger's Autos, Locals, and Watch windows to monitor the value of individual objects and data members. This information can help you to pinpoint the cause of unexpected results.

Appendix C: Changing Passwords

Microsoft Office SharePoint Server installs many services on the server. The installer uses the same login credentials for each service.

You might encounter a situation that requires you to change the password of the login credentials that all the SharePoint services use as a login identity. For example, many companies have policies that expire login passwords after a specified period of time. When the login expires, the user is prompted to create a new password.

It can be difficult to manually locate and update each SharePoint service that requires the new password. To simplify the process, create a batch file that contains the following script:

```
@echo off
rem other app pools
echo *** Updating app pool passwords
"%commonprogramfiles%\Microsoft Shared\Web server
extensions\12\BIN\Stsadm.exe" -o updateaccountpassword -userlogin %1 -
password %2 -noadmin
rem central admin
echo *** Updating Central Admin password
"%commonprogramfiles%\Microsoft Shared\Web server
extensions\12\BIN\Stsadm.exe" -o updatefarmcredentials -userlogin %1 -
password %2
rem ssp - new
echo *** Updating ssp password for new installs
"%commonprogramfiles%\Microsoft Shared\Web server
extensions\12\BIN\Stsadm.exe" -o editssp -title "SharedServices1" -ssplogin
%1 -ssppassword %2
rem ssp - upgrade
echo *** Updating ssp password for upgraded installs
"%commonprogramfiles%\Microsoft Shared\Web server
extensions\12\BIN\Stsadm.exe" -o editssp -title "Default Web Site" -ssplogin
%1 -ssppassword %2
rem osearch
echo *** Updating osearch password
"%commonprogramfiles%\Microsoft Shared\Web server
extensions\12\BIN\Stsadm.exe" -o osearch -farmserviceaccount %1 -
farmservicepassword %2
echo *** MANUAL UPDATE NEEDED. To update the password, visit the SSP Web
application page, click Search Settings, and then click Default Content
Access Account. rem spsearch
echo *** Updating spsearch password
"%commonprogramfiles%\Microsoft Shared\Web server
extensions\12\BIN\Stsadm.exe" -o spsearch -farmserviceaccount %1 -
farmservicepassword %2
echo *** Updating spsearch content access account
"%commonprogramfiles%\Microsoft Shared\web server
extensions\12\BIN\stsadm.exe" -o spsearch -farmcontentaccessaccount %1 -
farmcontentaccesspassword %2
rem restarting IIS
echo *** Doing soft restart of IIS
iisreset /noforce
echo on
```



You may need to modify the script to include the correct names of each SSP in your server farm.

To run the batch file, open a command prompt and set the working directory to the folder where you placed the batch file. Enter the following command. Substitute

FileName, *DomainName\UserName* and *NewPassword* with the batch filename, identity, and password you want to use with your SharePoint services. Press Enter.

```
FileName.bat DomainName\UserName NewPassword
```

For additional information on changing passwords for service accounts in SharePoint Server 2007 and in Windows SharePoint Services 3.0, search the Knowledge Base articles available at the Microsoft Help and Support site.

Glossary

Abstract base class

A class that cannot be instantiated directly and is solely for the purpose of inheritance. The class has abstract members that have no implementation. The derived class supplies the implementation of the base class members. The abstract base class defines the features of the derived class.

Application programming interface (API)

A set of functions or features you access to programmatically use or manipulate a software component or application.

BusinessObjectKey

A class that Microsoft Dynamics GP Workflow uses to identify a business document.

Business Logic

A collection of rules that constrain and guide the handling of business data.

Client workflow assembly

A Visual Studio Tools for Microsoft Dynamics GP that adds workflow functionality to a Dynamics GP form.

Criteria object

An object that contains the restrictions that define what is to be returned from a GetList method in the Dynamics GP web service.

eConnect

A collection of tools, components, and APIs that provide programmatic integration with Microsoft Dynamics GP.

Extension assembly

An assembly that contains the processing code for a web service extension that is extending one or more business objects for the Dynamics GP web service.

Form factory

A Microsoft .NET managed code component that runs immediately before a Dexterity window is created. A form factory can be used for many purposes, such as adding managed code controls to the window.

Global Assembly Cache

A machine-wide cache that stores assemblies that are shared by several applications on the computer.

Interface

In C#, interface describes a group of related behaviors that belong to a class

Microsoft Office SharePoint Server

Provides a host process for workflows. Provides a data store for workflow definition and for each active workflow instance.

Proxy

A special set of classes that will act as a wrapper for the operations, objects, and enumerations defined by the web service.

Server workflow assembly

A Microsoft .NET assembly that defines a workflow type for a Microsoft Dynamics GP document.

SOAP

Simple Object Access Protocol. The XML-based protocol used to communicate with a web service.

Summary object

An object that contains only the most important details of the main object. For example, the customer summary object contains only the most important details of the customer object. Summary objects are returned by the GetList methods in the Dynamics GP web service.

Transaction Requester

An eConnect service that retrieves XML that represents Microsoft Dynamics GP documents.

Transform

The process of converting an input XML document in an output XML document with a different structure.

Web reference

A URL that points to the .asmx file that defines the web service.

Web service

A software system that provides data and services to other applications. Web services use standard Internet transport protocols such as Hypertext Transfer Protocol (HTTP) and standard XML-based document formats such as Simple Object Access Protocol (SOAP) to exchange information.

Windows Workflow Foundation

A Microsoft technology for defining, executing, and managing workflows.

Workflow

A package of functionality used to enable a process that can be applied to documents and list items. Workflow automates the routing of documents to the person responsible for working on the document.

Workflow definition

Contains the activities, schedule, and configuration information that control the operation of a workflow.

Workflow metadata

Configuration information that describes the approval routing hierarchy for an individual workflow.

Workflow schedule

Controls what types of tasks a workflow can perform and how each task is scheduled.

XML

A text-based format that uses markup tags (words surrounded by '<' and '>') to describe how a document is structured and the data it contains.

XSLT

Extensible Stylesheet Language Transformation is a language that transforms an XML document into another document that is different in form or structure.

Index

A

- abstract base class defined 217
- action pane
 - commands 55
 - constants 56
- Appendix 208-216
 - Appendix A, Troubleshooting 209-210
 - Appendix B, Debugging 211-213
 - Appendix C, Changing Passwords 215-216
- application assembly, creating 67
- application programming interface, defined 217
- ApplicationServer.exe, example 170
- approval steps, adding properties with workflow extension 31, 177
- approval workflow, described 39
- ApprovalWorkflow 138
- ApprovalWorkflowController, required properties 81
- Architecture, chapter 7-13
- architecture, diagram 7
- archive, workflow history 41
- assembly configuration, web reference URL 136

B

- business document summary, adding data to with workflow extensions 31, 177
- business logic
 - defined 217
 - described 40
 - server workflow assembly 148
 - window integration 46
- BusinessObjectFile.config, described 183
- BusinessObjectKey
 - client workflow assembly 70
 - defined 217
 - described 204
 - reference 205
- BusinessService, described 124
- BusinessSummaryObjectInformation 146

C

- Changing Passwords, Appendix C 215-216
- class, for web service extension 180
- Client Workflow Assembly, chapter 69-95
- client workflow assembly
 - create project 69
 - debugging 211
 - define key 70
 - defined 217
 - described 69
 - form controller class 72
 - form factory 91

- client workflow assembly (*continued*)
 - installing 171
 - list controller class 83
 - workflow type sample 27
- command form
 - described 44
 - workflow enabled checkbox 45
 - WorkflowStatus 44
- Connecting to the Web Service, chapter 199-201
- conventions, in documentation 3
- CreateDexDialogForm, form factory example 93
- CreateDexForm, form factory example 92
- Creating a New Workflow, part 38-173
- credentials, for web service instance 201
- criteria class
 - creating 109-114
 - described 109
- criteria object, defined 217

D

- data, for web service extension 180
- data access
 - for workflow extensions 178
 - requirements 40
- Debugging
 - Appendix B 211-213
 - client workflow assembly 211
 - server workflow assembly 212
- Deploying the New Workflow, chapter 167-173
- Deploying the Workflow Extension, chapter 193-195
- Designing a New Workflow, chapter 39-42
- Designing a Workflow Extension, chapter 177-178
- Dictionary Assembly Generator 67
- Dictionary Changes, chapter 43-67
- document summary, adding data to with workflow extensions 31, 177
- document summary class
 - creating 105-109
 - described 105
- document type assembly
 - components 97
 - create a summary class 105
 - creating 97-114
 - criteria class 109
 - described 97
 - document type class 98
 - enumerations 105
 - installing 167
 - key class 102
 - list class 109
- document type class, creating 98
- document viewer
 - described 10, 155
 - installing 171
 - links 11

- document viewer (*continued*)
 - workflow type sample 27
- documentation, symbols and conventions 3
- documents, specifying keys for 204
- drop-down list fields, as filterable properties 143
- Dynamics Security Service
 - described 125
 - subscription application updates 162
 - workflow type sample install 23
- Dynamics Workflow web service
 - see also* workflow web service 2
 - documentation for 203
 - example 203
 - instance 200
 - namespace for 200
 - reference documentation 203
 - URL 199
- Dynamics Workflow Web Service Reference, described 203
- DynamicsGPService.asmx 199
- DynamicsWorkflow attribute 138
- DynamicsWorkflowEventManager 161
- DynamicsWorkflowEvents.config, subscribing to workflow events 170
- DYNWORKFLOWGRP
 - security group 9
 - table security 44

E

- eConnect
 - defined 217
 - described 114
 - web service 114
- eConnect Transaction Requester
 - see also* transaction requester
 - columns 114
 - described 114
 - update query example 114
 - updating the eConnect_Out_Setup table 167
 - workflow type sample 22
- enumerations, document type class 105
- event handler, for web service events 181
- EventHandlerImplementation 149
- example, Dynamics Workflow web service 203
- examples, document viewer 156
- existing workflows, listed 16
- Extending an Existing Workflow, part 176-195
- extension assembly, defined 217

F

- filterable properties
 - implementing 143
 - resource entries 140
 - retrieving from a resource file 142
- FindAllTrackingHistory 147
- FindAllWorkflowHistory 147

form controller
 described 72
 event handlers 77
 fields 74
 methods 74
 properties 80

form factory
 CreateDexDialogForm example 93
 CreateDexForm example 92
 creating 91-95
 defined 217
 described 91
 parameters 172
 registering in Dynamics.exe.config 171
 tasks 91

form-level procedures
 list integration 65
 window integration 47

G
 General Ledger Batch, BusinessObjectKey for 205
 GetAvailableFilterableProperties 143
 GetAvailableProperties method, in workflow extension assembly 189
 GetBusinessObject 144
 GetBusinessObject method, in workflow extension assembly 190
 GetOrganizationName 145
 GetSummaryInformation 146
 GetSummaryInformation method, in workflow extension assembly 191
 GetViewerName 146
 global assembly cache
 adding a server workflow assembly 169
 adding web service extension assembly 193
 adding workflow extension assembly 193
 defined 217
 global fields, required 43
 Glossary 217
 GUIDs
 creating a GUIDAttribute 98
 generating for server workflow assembly 138
 generating for workflow extension assembly 189
 security application 125
 server workflow assembly resources file 139
 subscription application 163

H
 home page integration, configuring 51

I
 IDynamicsWorkflow
 described 142
 GetAvailableFilterableProperties 143

IDynamicsWorkflow (*continued*)
 GetBusinessObject 144
 GetOrganizationName 145
 GetSummaryInformation 146
 GetViewerName 146
 implementing 142-147

IDynamicsWorkflowHistory
 described. 147
 FindAllTrackingHistory 147
 FindAllWorkflowHistory 147
 implementing 147-148
 SaveWorkflowHistory 148
 SaveWorkflowTrackingHistory 148

iisreset command 209
 installation helper, Visual Studio project for 194
 integrating applications
 installing a new application 167
 using with existing workflows 177

Integration Types, chapter 15-17
 integrations
 create a workflow 15
 extend an existing workflow 16

interface defined 217

K
 KeyPart
 described 204
 value of 204

keys, for business objects 204

L
 light bulb symbol 3
 list class
 creating 109
 described 109

list controller
 described 83
 fields 85
 methods 85
 properties 88
 VS Tools event handlers 87
 WFActionForList 87

list fields, as filterable properties 143
 list integration
 action pane 55
 columns 64
 constants 53
 described 53
 form-level procedures 65
 hidden fields 54
 tables 55

ListWorkflowController, required properties 89
 login credentials, for web service instance 201

M
 margin notes 3
 Microsoft Dynamics GP client, workflow integration 11

Microsoft Dynamics GP workflows, listed 16
 Microsoft Office SharePoint client, *see* SharePoint client
 Microsoft Office SharePoint Server
 defined 217
 described 8
 Microsoft Outlook client, *see* Outlook
 Microsoft.Dynamics.Workflow.Proxy.dll, described 200

N
 namespace, for Dynamics Workflow web service proxy 200
 notifications, described 48

O
 Outlook, client 13

P
 Payables Batch, BusinessObjectKey for 206
 priority, updating 47
 product support, for Workflow for Microsoft Dynamics GP 3
 proxy, defined 217
 proxy assembly, for workflow web service 200
 Purchase Order, BusinessObjectKey for 206

R
 Receivables Batch, BusinessObjectKey for 206
 Registration, form factory 171
 RegistrationSchedule.exe, example 170
 resources file
 for workflow extension assembly 188
 required entries for server workflow assembly 139
 Retrieved event handler, for web service extension 181

S
 Sales Lead web service, *see* web service
 Sales Order, BusinessObjectKey for 206
 Sample Workflow Extension, chapter 31-36
 Sample Workflow Type, chapter 19-29
 SaveWorkflowHistory 148
 SaveWorkflowTrackingHistory 148
 security
 application pool identity 9
 described 9
 DYNWORKFLOWGRP 9
 SharePoint groups 9
 table security 44
 web services 9
 security application
 configuration file 129
 creating 124-133
 described 125

- security application (*continued*)
 - GUIDs 125
 - uninstall 130
 - Server Workflow Assembly, chapter 135-165
 - server workflow assembly
 - business logic 148
 - create workflow type 137-154
 - creating 135-165
 - debugging 212
 - defined 217
 - described 135
 - document viewer 155
 - events 141
 - IDynamicsWorkflow interface 142
 - installing 168
 - registering 170
 - resources file 139
 - signing 154
 - web reference 136
 - workflow type sample 25
 - server workflow assembly configuration, installing 168
 - SharePoint, client 12
 - .snk file, described 182, 192
 - SOAP, defined 217
 - strong name key
 - for web service extension assembly 182
 - for workflow extension assembly 192
 - subscription application
 - configuration file 164
 - creating 160-165
 - Dynamics Security Service GUIDs 163
 - DynamicsWorkflowEventManager 161
 - example 163
 - execute 170
 - include installation tasks 162
 - installing 170
 - summary object, defined 217
 - support, for Workflow for Microsoft Dynamics GP 3
 - symbols in documentation 3
- T**
- tables, adding fields 43
 - technical support, for Workflow for Microsoft Dynamics GP 3
 - testing
 - sales lead web service 132
 - web service extensions 184
 - transaction requester
 - see also* eConnect Transaction Requester
 - defined 217
 - transform
 - defined 217
 - eConnect XML 115
 - Troubleshooting
 - Appendix A 209-210
 - reset the server 209
 - verify authorization 209
- U**
- URL, for workflow web service 199, 201
 - Using the Web Service, chapter 203-206
- V**
- Visual Studio project
 - for client workflow assembly 69
 - for document type 97
 - for server workflow assembly 135
 - for subscription application 160
 - for web service 121
 - for web service extension 179
 - for web service extension installation helper 194
 - for web service security application 125
 - for workflow extension 187
- W**
- warning symbol 3
 - web reference, defined 217
 - Web Service, chapter 97-133
 - web service
 - create asmx file 121
 - creating 114-124
 - creating an XSLT file 115-121
 - defined 217
 - eConnect changes 114
 - installing 167
 - installing new web service files 167
 - new web method 123
 - securing a new web service 168
 - security application 124
 - testing 132
 - XSLT map 115
 - web service event handler, for web service extension 181
 - web service extension assembly
 - adding to global assembly cache 193
 - building 182
 - defining data for 180
 - event handler 181
 - location of 182
 - web service extensions
 - chapter 179-185
 - defining data for 180
 - registering 183
 - testing 184
 - using with workflow extensions 178, 179
 - Visual Studio project for 179
 - web service event handler 181
 - window integration
 - business logic 46
 - Dexterity 45
 - form-level procedures 47
 - hidden fields 46
 - window integration (*continued*)
 - workflow wrapper 47
 - Windows Workflow Foundation, defined 217
 - workflow
 - defined 217
 - support 3
 - Workflow Basics, part 6-29
 - workflow client
 - Microsoft Dynamics GP 11
 - Outlook 13
 - SharePoint 12
 - workflow clients, described 11-13
 - workflow definition, defined 217
 - workflow events, described 78
 - workflow extension assembly
 - adding to global assembly cache 193
 - building 192
 - chapter 187-192
 - code for 189
 - described 178, 187
 - resources file for 188
 - workflow extensions
 - accessing data 178
 - capabilities of 177
 - deploying 193
 - designing 177
 - installation helper 194
 - registering 195
 - Visual Studio project for 187
 - workflow approval steps 31, 177
 - workflow extension assembly 178
 - workflow history, archive options 41
 - workflow metadata
 - defined 217
 - described 8
 - workflow schedule
 - defined 217
 - described 8
 - workflow server, described 8-9
 - workflow subscription
 - described 160
 - DynamicsWorkflowEvents.config 160
 - workflow type
 - business logic 148
 - described 137
 - required base class 138
 - required interfaces 138
 - resource handler 142
 - workflow type sample
 - client workflow assembly 27
 - eConnect Transaction Requester 22
 - installing 20-28
 - installing the application 20
 - installing the document type assembly 21
 - installing the document viewer 27
 - installing the web service 22
 - overview 19
 - sample files 20
 - server workflow assembly 25

INDEX

- workflow type sample (*continued*)
 - updating the Dynamics Security Service 23
 - viewing 28
- workflow types
 - data management 10
 - described 9-11
 - document viewer 10
 - events 10
 - names of 190, 194
 - requirements 10
- workflow web service
 - see also* Dynamics Workflow web service
 - described 8
 - proxy assembly 200
 - URL 199, 201
- workflow wrapper, described 47
- WorkflowEventManager 160
- WorkflowStatus window 44
- workfow type, event handler 141

X

- XML
 - defined 217
 - eConnect documents 115
- XSLT
 - creating 115
 - defined 217
 - example 119
 - installing 167
 - transform 115