

Developing SharePoint Applications



patterns & practices
proven practices for predictable results

August 2009

Summary

The *Developing SharePoint Applications* guidance helps architects and developers design and build applications that are both flexible and scaleable. It shows developers how to provide IT professionals with the information they need to maintain those applications and diagnose problems when they arise. The two reference implementations illustrate how to solve many of the common challenges developers encounter. One reference implementation addresses basic issues such as creating lists and content types. The other addresses more advanced problems such as how to integrate line of business services, how to create collaboration sites programmatically, and how to customize aspects of publishing and navigation. A library of reusable components helps you adopt techniques used in the reference implementations. The guidance discusses approaches for testing SharePoint applications, such as how to create unit tests, and documents experiences with stress and scale testing one of the reference implementations.

Downloads	Developing SharePoint Applications – August 2009 (includes the SharePoint Guidance Library, Contoso Partner Portal Reference Implementation, Contoso Training Management Reference Implementation, and QuickStarts)
Community	CodePlex Community Site
Learn More	Video Overview Setting up the Contoso RI Walkthrough of the Contoso Reference Implementation How to use the configuration component? How to use the logging components? How to use the SharePoint Service Locator?

Overview

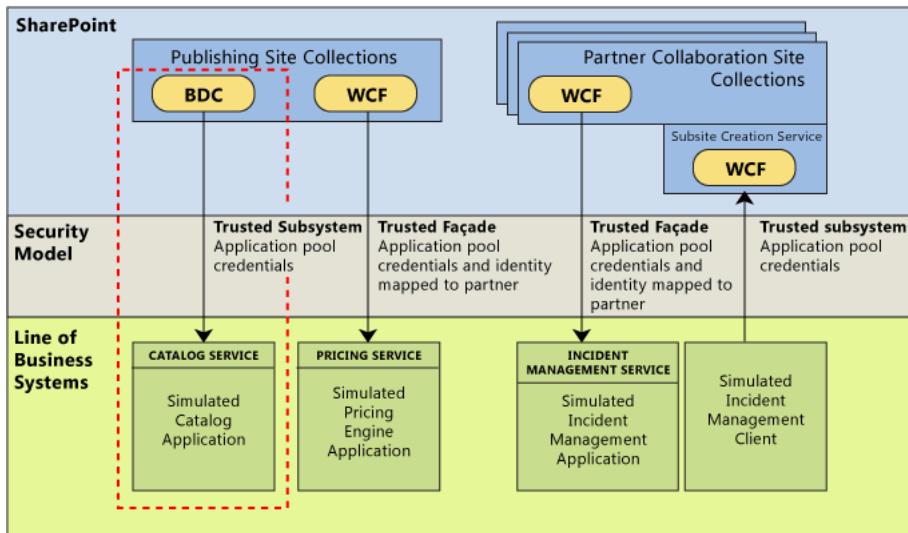
The guidance can help you to address these common SharePoint development scenarios:

- How to use application and design patterns to address common development challenges.
- The design and use of the SharePoint Guidance Library components.
- Architectural decisions that affect site topology and security.
- How to design and implement SharePoint applications that are scalable, manageable, and configurable.
- How to integrate SharePoint applications with Web services. This includes discussions about design tradeoffs and decisions about security.
- How to incorporate SharePoint's publishing and content deployment capabilities into your applications.
- Flexible approaches to navigation and branding, such as how to implement custom, cross-site-collection global navigation and custom site navigation
- How to decide between design options, such as how to choose between a list and database.
- How to design a SharePoint application for testability, how to create unit tests, and how to run continuous integration tests.
- A discussion of how scale and stress tests were performed on the Partner Portal application.
- How to set up different team build and testing environments.
- How to manage the application life cycle through the development, test, deployment, and upgrade stages.
- How to implement a team-based development environment.

The guidance also includes implementations that exemplify many of these scenarios. The implementations include the Partner Portal and Training Management applications, and the QuickStarts. The following illustration shows how the Partner Portal application integrates SharePoint capabilities with line-of-business systems.

Partner Portal application

Catalog Integration (Product and Category)



Note:

This guidance incorporates the v1 content, which was written documentation and the Training Management application.

Audience Requirements

Developing SharePoint Applications is intended for software architects and experienced developers. To get the most benefit from this guidance, you should have an understanding of the following technologies:

- Microsoft Visual C#
- Microsoft .NET Framework
- ASP.NET

The guidance includes the Training Management application, which is for experienced developers who are new to SharePoint. These developers may also be interested in [Developing SharePoint Applications](#), in the written guidance. This topic introduces some basic SharePoint concepts to ASP.NET developers.

The Partner Portal application is more advanced and assumes that you have some experience with SharePoint. It demonstrates how to build an enterprise-scale application.

System Requirements

To develop applications with the SharePoint reference implementation, the minimum requirements are the following:

- Windows Server 2003 or later
- [Microsoft .NET Framework 3.5](#)
- Microsoft Visual Studio 2008 development system (any of the following editions):
 - Professional Edition
 - Team Edition for Software Developers
 - Team Edition for Software Testers
 - Team Edition for Software Architects
 - Team System

Note:

To run the test projects, you must have Team Edition for Software Testers or Team Systems.

- [Visual Studio 2008 extensions for Windows SharePoint Services 3.0, v. 1.3 – March 2009 CTP](#)
- [Microsoft Silverlight 2 Software Development Kit](#)
- (Optional) [Typemock Isolator 5.2.1](#) or later (this is required to run the unit tests)
- Windows SharePoint Services 3.0 for the Training Management application or Microsoft Office SharePoint Server 2007 with Service Pack 1 or Service Pack 2 for both the Training Management application and the Partner Portal application

Getting Started

If you are new to SharePoint development, the first step is to study the Training Management application, which is based on Windows SharePoint Services. The documentation and the application can help developers understand the fundamentals of SharePoint development, and compliments other training resources and publications.

If you are an experienced ASP.NET developer but new to SharePoint, read [An Overview of the SharePoint Platform for ASP.NET Developers](#). If you are interested in the Partner Portal application, see [A Quick Tour of the Partner Portal Application](#).

Architects may want to read [Architectural Decisions](#). This topic explains the rationale for the design of the Partner Portal application. It includes discussions on the design of the site topology, security decisions and how the application was integrated with external LOB systems.

If you are interested in the reusable components, see [The SharePoint Guidance Library](#). These components can help you to create your own applications more quickly and to ensure that they follow best practices.

Future Plans

The patterns & practices team is currently investigating additional areas where it can provide SharePoint guidance. Download early drops from the [CodePlex community site](#). To provide feedback, please create and vote on work items in the CodePlex [issue tracker](#).

Feedback and Support

Questions? Comments? Suggestions? To provide feedback about this guidance, or to get help with any problems, please visit the [SharePoint Guidance community site](#). The message board on the community site is the preferred feedback and support channel because it allows you to share your ideas, questions, and solutions with the entire community. *Developing SharePoint Applications* is a guidance offering, designed to be reused, customized, and extended. It is not a Microsoft product. Code-based guidance is shipped "as is" and without warranties. Customers can obtain support through Microsoft Support Services for a fee, but the code is considered user-written by Microsoft support staff.

Introduction

Developing SharePoint Applications provides guidance for architects and developers who want to create enterprise-scale applications. This introductory topic includes the following:

- [Overview](#). This is a brief description of the guidance.
- [SharePoint Guidance Assets](#). This describes what is contained in the guidance.
- [Intended Audience](#). This describes the prerequisites for understanding the guidance.
- [When to Use This Guidance](#). This describes situations that are appropriate for the guidance.
- [Community and Knowledge Base](#). This provides a link to a site for users of this guidance.
- [Evaluating the SharePoint Guidance](#). This describes how to decide whether this guidance is appropriate for you.
- [Useful Development Tools](#). This describes various tools that are helpful for developing SharePoint applications.

Overview

By using SharePoint, businesses can create the following types of applications:

- Applications with comprehensive content management capabilities.
- Applications with collaborative spaces to share information.
- Applications that can integrate with external line-of-business (LOB) services.

Additionally, it provides IT professionals and developers with the platform and tools they need for server administration, application extensibility, and interoperability.

Developing SharePoint Applications helps architects and developers design, build, test, and deploy enterprise-scale SharePoint applications. A reference implementation named the Partner Portal application is based on Microsoft Office SharePoint Server 2007. It shows how to develop a content-oriented, collaborative application that integrates LOB data for a partner extranet using several cooperating sites.

Another reference implementation, the Training Management application, is based on Windows SharePoint Services (WSS) 3.0. It shows how to build a single site, intranet SharePoint application.

The SharePoint Guidance Library includes a set of reusable components that is helpful when you develop your own SharePoint applications. They were created in response to the challenges of building the Partner Portal application.

This guidance covers the following subjects in the context of SharePoint applications:

- It describes how to use application and design patterns to help solve common development challenges.
- It describes the design and use of the SharePoint Guidance Library, which is a set of reusable components.
- It describes how to design and secure a site topology.
- It describes how to design and implement an application that is scalable, manageable, and configurable.
- It describes how to integrate a SharePoint application with Web services. This includes security options and design tradeoffs.
- It describes how to develop a SharePoint application that can publish and deploy content.
- It describes flexible approaches to navigation and branding, including how to implement custom cross-site collection global navigation and custom site navigation.
- It includes guidance on many of the common decisions that developers encounter, such as choosing between a list and database.
- It includes implementation examples that are demonstrated in the Partner Portal application, the Training Management application, and in the QuickStarts.
- It includes guidance on testing. For example, there are discussions on unit tests, build verification tests, and continuous integration tests.
- It includes a discussion of scale and stress testing that uses the Partner Portal application as an example.
- It describes how to set up different team build and testing environments.
- It describes how to manage the application life cycle through development, test, deployment, and upgrading.
- It describes a team-based application development.

For more complete information about the topics in the guidance, see [Topics at a Glance](#). The guidance incorporates the content and Training Management reference implementation provided in the previous release of SharePoint Guidance. The guidance does not address how to build Internet-scale SharePoint applications or multilingual SharePoint applications.

SharePoint Guidance Assets

The patterns & practices Developing SharePoint Applications guidance includes the following assets:

- **Documentation.** There are two CHM files. One contains the written guidance for the Partner Portal application, the SharePoint Guidance Library, and the Training Management application. The other is the API reference for the SharePoint Guidance Library.
- **SharePoint Guidance Library.** This library contains reusable components that you can include in your own SharePoint applications. The library components can help you to build SharePoint applications that incorporate many of the best practices that are discussed in the documentation.
- **Contoso Partner Portal Reference Implementation (Partner Portal application).** (This is referred to as the Partner Portal application in the written guidance). This SharePoint application demonstrates how Contoso Pharmaceuticals uses SharePoint to provide an extranet to its external partners. This application demonstrates how to design and implement an enterprise-scale SharePoint application.
- **Contoso Training Management Reference Implementation (Training Management application).** (This is referred to as the Training Management application in the written guidance). This is a much simpler SharePoint application than the Partner Portal application. It demonstrates how to perform basic SharePoint tasks.
- **QuickStarts.** These include the source code for two small, focused applications that illustrate how to debug an **SPItemEventReceiver** and how to access items in a SharePoint list.

The following are the remaining topics in this section:

- [Intended Audience](#)
- [When to Use This Guidance](#)
- [Community and Knowledge Base](#)
- [Evaluating the SharePoint Guidance](#)
- [Useful Development Tools](#)
- [More Information](#)
- [Copyright and Terms of Use](#)

Intended Audience

Developing SharePoint Applications is intended for software architects and experienced developers. To get the most benefit from this guidance, you should have an understanding of the following technologies:

- Microsoft Visual C#
- Microsoft .NET Framework
- ASP.NET

The guidance includes the Training Management application, which is for experienced developers who are new to SharePoint. This application shows how to create a basic intranet SharePoint application. For example, it shows how to use fundamental SharePoint components, such as lists and content types. These developers may also be interested in [Developing SharePoint Applications](#), which introduces some basic SharePoint concepts to ASP.NET developers.

The Partner Portal application is more advanced and assumes that you have some experience with SharePoint. It demonstrates how to build an enterprise-scale application.

When to Use This Guidance

Some aspects of SharePoint development, such as adding Web Parts, are relatively straightforward, but as organizations grow, they require more complex applications. To build those applications demands a deeper understanding of SharePoint's capabilities. Organizations can gain that understanding from this guidance.

If you are new to SharePoint development, the first step is to study the Training Management application, which is based on Windows SharePoint Services. The documentation and the application can help organizations understand the fundamentals of SharePoint development. Other areas of interest in the documentation include how to set up a team development environment, and how to test a SharePoint application.

The guidance also covers more advanced areas of SharePoint development. These areas include the following:

- Integrating SharePoint applications with line-of- business (LOB) systems
- Customizing published content
- Adding custom navigation to an application
- Composing Web Parts that use the Business Data Catalog (BDC) to access publishing data
- Adding enterprise-scale capabilities to your application, such as caching, exception management, and configuration management

The Partner Portal application, which is the second reference implementation that is included in the guidance, shows how to implement these capabilities. For more information about the topics that are covered in the guidance, see [Topics at a Glance](#).

This guidance is not a replacement for product documentation. Instead, it enhances the documentation by applying that information to a realistic business situation or to one of the reference implementations. In many cases, the guidance refers to the product documentation. You can use the guidance to gain initial understanding. You can then use the product documentation for deeper understanding.

The guidance includes several reusable components that are grouped together as the SharePoint Guidance Library. These components can help you with your own development projects. They implement many of the principles that were used to develop the reference implementations. However, you do not need to use these components to get value from the guidance.

The guidance does not discuss how to customize many areas of SharePoint functionality. In other areas, such as search, this guidance provides a starting point, but it does not provide in-depth information about the subject. The following are several areas that are not covered in this guidance:

- Business intelligence
- Excel and InfoPath services
- Globalization and localization
- SharePoint My sites and social networking

Community and Knowledge Base

The Developing SharePoint Applications guidance, like other Microsoft patterns & practices deliverables, is associated with a [community site](#). On this community site, you can post questions, provide feedback, or connect with other users for sharing ideas. Community members can also help Microsoft plan and test future offerings and download additional content, such as extensions and training material.

Evaluating the SharePoint Guidance

Developing SharePoint Applications provides useful information in specific areas of SharePoint development without requiring you to first understand the more complex and general topics, such as the patterns that are used in the reference implementations, or how to develop an application with the reusable library components. Architects and developers will need to devote some time and effort to fully understand and evaluate the reference implementations and reusable components.

The guidance is divided into the following four areas:

- Documentation
- Library of reusable components
- Training Management application
- Partner Portal application

Some of the documentation and the Training Management application are from the previous release. Some of this earlier documentation has been integrated with guidance from the current release.

The documentation provides practical engineering guidance to various aspects of building SharePoint applications. It is organized to provide useful information that is independent of the reference implementations and reusable components. Many of the topics refer to the reference implementations so that you can see how the general principles that are discussed in the documentation are applied. In particular, understanding the Partner Portal application, which is a relatively complex SharePoint application, is easier if you read the general guidance first and then refer back to the application.

The library of reusable components demonstrates many of the principles that are discussed in the documentation. For example, the guidance includes discussions of the patterns that the components implement. You should study the SharePoint Guidance Library documentation before you incorporate the components into your own applications. The documentation explains how they are designed and how to use them in common situations. It also points out implementation details such as that there are sometimes dependencies between the components. For example, the SharePoint Logger depends on the SharePoint service locator.

The Training Management application demonstrates many principles of SharePoint development that apply to an application with a single site. The following are some of the scenarios where the Training Management application can provide guidance:

- You are using Windows SharePoint Services 3.0 or Microsoft Office SharePoint Server 2007.
- You are relatively new to SharePoint or would like to learn how to apply basic best practices to your application.
- You want to learn how to upgrade a SharePoint application
- You want to learn how to build a relatively complex application for a single site.
- You want to learn how to develop workflows for SharePoint.
- You want to learn how to use lists and how to use event receivers with lists.
- You want to learn how to package features and solutions.

The Partner Portal application shows how to design and implement enterprise-scale SharePoint applications. The following are some of the scenarios where the Partner Portal application can provide guidance:

- You are using Microsoft Office SharePoint Server 2007.
- You want to provide an extranet for external organizations. The extranet includes collaboration sites.
- You want to publish and deploy content.
- You want to integrate line-of-business (LOB) information with the application and create published content that includes LOB information.
- You want to aggregate and cross-link information.
- You want to be able to navigate across multiple site collections.
- You want to incorporate best practices for performance and manageability issues.

Useful Development Tools

The Partner Portal application was developed with the Visual Studio extensions for Windows SharePoint Services. The extensions include many useful features and project templates. To learn more, see [Visual Studio extensions for Windows SharePoint Services](#).

Two common development activities are writing Collaborative Application Markup Language (CAML) queries and creating custom list definitions. There are various tools available that simplify these tasks. For information about tools that can help you write CAML queries, see [CAML Query Tools](#). For information about tools that can help you create custom list definitions, see [List Definition Authoring Tools](#).

The Partner Portal application uses the Business Data Catalog (BDC) to access an external system that contains the Contoso product catalog. There are several editors available to help you write the BDC application definition file. For more information, see [Business Data Catalog Editors](#).

If you do not want to programmatically change a SharePoint property bag, you can use one of the property bag editors that are available. For more information, see [Property Bag Editors](#).

This release of the Partner Portal application and Training Management application includes unit tests that use mock objects. The unit tests require a third-party software tool named Typemock Isolator. For more information, see [Typemock](#).

SharePoint Designer allows you to create and deploy interactive solutions on the SharePoint platform, without having to write code. For more information, see [SharePoint Designer](#).

Visual Studio extensions for Windows SharePoint Services

[Visual Studio extensions for Windows SharePoint Services](#) is a tool for developing custom SharePoint applications. It provides developers with a familiar development environment within Visual Studio. The extensions include a collection of useful project and item templates for many of the most common development activities in SharePoint.

The following Visual Studio 2008 project templates are included with Visual Studio extensions for SharePoint Services:

- Web Part
- Team Site Definition
- Blank Site Definition
- List Definition
- Empty SharePoint Project

The following Visual Studio 2008 item templates are included with Visual Studio extensions for SharePoint Services:

- Web Part
- Custom Field
- List Definition (with optional Event Receiver)
- Content Type (with optional Event Receiver)
- Module
- List Instance
- List Event Handler
- Template

The extensions also provide one-click deployment and F5 debugging. Many tools such as [WSP Builder](#) and [STSDEV](#) require developers to maintain Feature.xml and Manifest.xml files for SharePoint solution packaging into a Web solution package (WSP).

The extensions provide a graphical user interface for organizing and packaging SharePoint solutions and do not require users to maintain their own Feature.xml and Manifest.xml files. Another difference between the extensions and other SharePoint tools such as WSP Builder is that they do not require developers to use the typical 12 hive folder structure in their Visual Studio project structures.

SharePoint Solution Generator

The SharePoint Solution Generator is a part of Visual Studio extensions for Windows SharePoint Services. Developers can use it to generate a site definition from an existing SharePoint site. Developers can use a Web browser to first customize a list on a new or existing SharePoint site and then use the SharePoint Solution Generator to generate list definition files. For more information about using the SharePoint Solution Generator, see the [Visual Studio extensions Windows SharePoint Services User Guide](#).

CAML Query Tools

Collaborative Application Markup Language (CAML) is an XML-based language that developers can use to create custom views and query SharePoint lists. Because it can be difficult to write anything other than basic queries, many tools have been developed to simplify this task. Several of these tools allow developers to write CAML queries within a designer. This is similar to the way developers write SQL queries for many commonly used databases. The following are some of the available tools:

- [U2U CAML Query Builder](#). This Windows-based application includes a user interface for writing CAML queries.
- [U2U CAML Query Builder Feature](#). This is a SharePoint version of the U2U CAML Query Builder tool. It is installed as a feature; developers can use it to build queries directly inside of SharePoint.
- [Stramit SharePoint CAML Viewer](#). This is a Windows-based application that is similar to U2U CAML Query Builder.
- [CAML.NET](#). This is a .NET Framework assembly that provides an abstraction to CAML. Developers can use it to dynamically create queries with a set of .NET Framework classes.

For more information about CAML, see [Collaborative Application Markup Language Core Schemas](#) on MSDN.

List Definition Authoring Tools

A custom list definition requires a Schema.xml file. This file defines the list schema, which includes content type associations, field, and view definitions and form and toolbar references. A list template element file references the list definition that is contained in the schema file. You can use the list template element file to create instances of new lists.

The following three types of tools can help you to create custom list definitions:

- A CAML query authoring tool
- The [Windows SharePoint Services 3.0 Tools: Visual Studio 2008 Extensions, Version 1.3](#)
- The SharePoint Solution Generator

The next sections describe these tools.

CAML Query Authoring Tools

A CAML query authoring tool can help you to create custom list views that contain CAML query definitions. For more information about CAML query authoring tools, see [CAML Query Tools](#).

Windows SharePoint Services 3.0 Tools: Visual Studio 2008 Extensions

The Visual Studio extensions for Windows SharePoint Services tool provides templates for authoring list definitions. These include a list definition project template and a list definition item template. Either can be based on a standard content type or a custom content type. For more information about Visual Studio extensions for Windows SharePoint Services, see [Visual Studio extensions for Windows SharePoint Services](#). For more information about creating custom list definitions using the extensions, see the [Visual Studio extensions for Windows SharePoint Services User Guide](#).

SharePoint Solution Generator

The SharePoint Solution Generator is a part of Visual Studio extensions for Windows SharePoint Services. Developers can use it to generate a site definition from an existing SharePoint site. For more information, see "SharePoint Solution Generator" in [Visual Studio extensions for Windows SharePoint Services](#).

More Information

For more information, see [Schema.xml](#), [List Schema](#), and [View Schema](#) on MSDN.

For more information about creating custom list definitions, see [How to: Create a Custom List Definition](#) on MSDN.

Business Data Catalog Editors

Defining the metadata model in the Business Data Catalog (BDC) application definition file can be time consuming and error prone. The Business Data Catalog Definition Editor, which is provided in the [SharePoint Server 2007 SDK](#), helps you to write this file. It uses a graphical user interface and automatically creates entities and methods from the database metadata and the Web Services Description Language (WSDL). For a demonstration of how to use the editor, see [Creating a Web Service Connection by Using the Business Data Catalog Definition Editor](#) on MSDN.

Another option is to use a commercial editor. An example is [BDC Meta Man](#). This has a number of features that are not available in the BDC Definition Editor.

Property Bag Editors

There are several editors available on CodePlex that allow you to manage property bag settings through a user interface instead of programmatically. These include the following:

- [SharePoint Property Bag Settings](#). This editor helps you manage configuration information at all SharePoint levels.
- [SharePoint 2007 WCM Utilities](#). This includes a number of Web Content Management utilities, including a property bag feature.

Typemock

The Training Management application includes unit testing with mock objects. There are a variety of tools available that support this approach. The Training Management application uses a commercially available tool named [Typemock Isolator](#). This is a valuable tool because you can use it to create mock instances of classes that have internal constructors. You can also use it to specify the behavior of class methods even though the class is sealed. For information about unit testing the Training Management application, see [Unit Testing with Mock Objects](#).

SharePoint Designer

By using SharePoint Designer, you can create and deploy interactive solutions on the SharePoint platform without having to write code. The following are some of the features that SharePoint Designer provides:

- The Workflow Designer, which allows you to automate business processes such as document approval, custom event notification, and other collaboration tasks.
- Pre-built Microsoft Windows SharePoint Services application templates, which are fully customizable and extensible using Office SharePoint Designer 2007.
- A what-you-see-is-what-you-get (WYSIWYG) editor, cascading style sheet (CSS) tools, and full support of ASP.NET master pages.

For more information, see [Microsoft Office SharePoint Designer](#).

More Information

The following links are for other SharePoint sites that you may find useful:

- [Automating Solution Package Creation for Windows SharePoint Services by Using MSBuild](#)
- [Microsoft Office SharePoint Developer Center](#)
- [Microsoft SharePoint Team Blog](#)
- [SharePoint Solution Installer](#)
- [Team-Based Development in Microsoft Office SharePoint Server 2007](#)
- [Using Visual Studio 2005, MakeCab.exe and MSBuild to Create Windows SharePoint Services v3 Solution Files \(*.WSPs\)](#)
- [Web Application Automation Tool \(WatIN\)](#)
- [Windows SharePoint Services](#)
- [Windows SharePoint Services Developer Center](#)
- [Windows SharePoint Services 3.0 SP1 Developer Evaluation VPC](#)

Copyright and Terms of Use

Information in this guidance, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples 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. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

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

© 2009 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows Server, Windows Vista, Active Directory, IntelliSense, MSDN, SharePoint, Silverlight, TechNet, Visual C#, Visual Basic, and Visual Studio are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Installing the SharePoint Guidance

This section explains how to install and uninstall the SharePoint Guidance and how to set up a development environment. Both the Contoso Training Management application and the Contoso Partner Portal application are installed by the installation script.

The section includes the following topics:

- [Contents of This Release](#)
- [System Requirements](#)
- [Selecting a Development Environment](#)
- [Setting Up a Development Environment](#)
- [Setting Up the Partner Portal Application](#)

Contents of This Release

The current release of this guidance contains the following folders:

- **Build.** This folder contains the MS build project for the Partner Portal build verification tests (BVTs).
- **Docs.** This folder contains the Partner Portal Help (.chm) files. One file contains the written guidance. The other file contains the API reference for the SharePoint Guidance Library.
- **Lib.** This folder contains the **Microsoft.Practices.ServiceLocation.dll** assembly.
- **Setup.** This folder contains the setup scripts for the Partner Portal application.
- **Source.** This folder contains the source files for the Partner Portal application, the Training Management application, the SharePoint Guidance Library, and the QuickStarts.
- **Test.** This folder contains the BVTs for the Partner Portal application and the integration tests for the SharePoint Guidance Library.

There is also a readme file named Readme.txt.

System Requirements

To develop applications with the SharePoint reference implementation, the minimum requirements are the following:

- Windows Server 2003 or later
- [Microsoft .NET Framework 3.5](#)
- Microsoft Visual Studio 2008 Service Pack 1 (SP1) development system (any of the following editions):
 - Professional Edition
 - Team Edition for Software Developers
 - Team Edition for Software Testers
 - Team Edition for Software Architects
 - Team System

Note:

To run the test projects, you must have Team Edition for Software Testers or Team System.

- [Visual Studio 2008 extensions for Windows SharePoint Services 3.0, v. 1.3 – March 2009 CTP](#)
- [Microsoft Silverlight 2 Software Development Kit](#)
- (Optional) [Typemock Isolator 5.2.1](#) or later (this is required to run the unit tests)
- Windows SharePoint Services 3.0 for the Training Management application or Microsoft Office SharePoint Server 2007 with Service Pack 1 or Service Pack 2 for both the Training Management application and the Partner Portal application

Selecting a Development Environment

A virtual machine or virtual PC (VPC) is the recommended development environment. You can use either [Virtual PC 2007](#) or [Virtual Server 2005 R2 SP1](#).

By using a virtual PC, developers can quickly begin their SharePoint development. It also provides a uniform environment for development teams.

The Partner Portal application's installation script is intended for systems that have a standalone Microsoft Office SharePoint Server configuration.

This guidance uses Windows Server 2003 for its configuration examples, but similar steps work for Windows Server 2008.

Setting Up a Development Environment

Before you install the SharePoint guidance, you should set up the development environment. The following series of procedures explain how to do this. Use the first version of the procedures if you are using Windows Server 2003. Use the second set if you are using Windows Server 2008.

Setting Up a Development Environment on Windows Server 2003

Use the following procedure to check the Visual Studio extensions for Windows SharePoint Services application identity pool to confirm that it belongs to the correct Windows groups.

To find the application identity pool name

1. Click **Start**, point to **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
2. Expand the **Web Sites** node.
3. Right-click **VSeWSS**, and then click **Properties**.
4. Click the **Home Directory** tab. In the **Application settings** section, find the application pool name and make a note of it.
5. In IIS Manager, expand the **Applications Pool** node.
6. Right-click the node that has the name you found in Step 4. Click **Properties**.
7. Click the **Identity** tab. Make a note of the user name. You will need this in step 3 of the next procedure.

After you have the identity name, check to see that the identity belongs to the appropriate Windows groups. The following procedure explains how to do this.

To check the Windows groups

1. Click **Start**, point to **Administrative Tools**, and then click **Computer Management**.
2. Expand the **Local Users and Groups** node, and then click the **Users** node.
3. In the right pane, right-click the user name that you found in step 7 of the previous procedure, click **Properties**, and then click the **Member of** tab.
4. Check whether the user name is a member of the following groups:
 - Administrators
 - IIS_WPG
 - Users
 - WSS_ADMIN_WPG
 - WSS_RESTRICTED_WPG
 - WSS_WPG

The next procedure explains how to confirm that Visual Studio extensions for Windows SharePoint Services is running correctly.

To check whether Visual Studio extensions for Windows SharePoint Services is running correctly

1. Click **Start**, point to **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
2. Expand the **Web Sites** node.
3. Click the **VSeWSS** node.
4. In the right pane, right-click **SPService.svc**. Click **Browse**.
5. You will see either a logon dialog box or the **SPService** service page. If you see a log-on dialog box, enter your Windows credentials. You must log on with administrator privileges to gain access to the **SPService** page.

Next, confirm that the SharePoint search service is running. The following procedure explains how to do this.

To check whether the search service is running

1. Click **Start**, point to **Administrative Tools**, and then click **SharePoint Central Administration 3.0**.
2. In the **Quick Launch**, click **Operations**.
3. In the **Topology and Services** section, click **Services on server**.
4. The status of the Office SharePoint Search service should be **Started**. If it is not, click **Start** in the Action column.

Windows Server 2008 Procedures

The following procedure allows you to check the Visual Studio extensions for Windows SharePoint Services application identity pool name to confirm that it belongs to the correct Windows groups.

To find the application identity pool name

1. Click **Start**, point to **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
2. Expand the **Sites** node in the **Connection** pane tree.
3. Right-click **VSeWSS**, point to **Manage Web Site**, and then click **Advanced Settings**. Note the name of the application pool.
4. Click **Application Pools** in the **Connection** pane.
5. In the **Application Pools** pane, right-click the pool with the name that you noted in step 3. Click **Advanced Settings**.

6. Under **Process Model**, note the name of the **Identity** setting.

After you have the identity name, confirm that the identity belongs to the appropriate Windows groups. The following procedure explains how to do this.

To check whether the identity belongs to the correct Windows groups

1. Click **Start**, point to **Administrative Tools**, and then click **Computer Management**.
2. Expand the **Local Users and Groups** node, and then click the **Groups** node.
3. Click each of the following groups and check that the identity name is a member:
 - Administrators
 - Users
 - WSS_ADMIN_WPG
 - WSS_RESTRICTED_WPG
 - WSS_WPG

The next procedure explains how to check whether Visual Studio extensions for Windows SharePoint Services is running correctly.

To check whether Visual Studio extensions for Windows SharePoint Services is running correctly

1. Click **Start**, point to **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
2. Expand the **Sites** node in the **Connection** pane.
3. Click the **VSeWSS** node, and then click the **Content View** tab.
4. In the right pane, right-click **SPService.svc**, and then click **Browse**.
5. You will see either a log-on dialog box or the **SPService** service page. If you see a log-on dialog box, enter your Windows credentials. The **SPService** service page appears. (You must log on with administrative privileges.)

Next, confirm that the SharePoint Search service is running. The following procedure explains how to do this.

To check whether the search service is running

1. Click **Start**, point to **Administrative Tools**, and then click **SharePoint Central Administration 3.0**.
2. In the **Quick Launch**, click **Operations**.
3. In the **Topology and Services** section, click **Services on server**.
4. The status of the Office SharePoint Search service should be **Started**. If it is not, click **Start** in the Action column.

Use the next procedure to confirm that the Windows Communication Foundation (WCF) Activation feature is installed on your server. If it is not installed, perform the steps in the subsequent procedure to install it.

To check whether the WCF Activation feature is installed

1. Click **Start**, point to **Administrative Tools**, and click **Server Manager**.
2. In the left pane, click **Features**.
3. In the **Features Summary** pane, under **Features**, look for **WCF Activation**. If you see it, the feature is installed and you are done. If it is not listed, you have to install it. Follow the remaining steps in this procedure.
4. In the toolbar, click **Action**, and then click **Add Features**.
5. If necessary, expand the **.NET Framework 3.0 Features** node.
6. Select **WCF Activation**, and then click **Next**.
7. Click **Install**.

Setting Up the Partner Portal Application

This section includes the following topics:

- [Installing the Partner Portal Application](#)
- [Uninstalling the Partner Portal Application](#)
- [Setting Up the Extranet Host Header](#)
- [Installation Scripts](#)

Installing the Partner Portal Application

To install the SharePoint guidance, you have to complete the following steps:

1. Download the [Developing SharePoint Guidance](#) and extract the .zip file to a directory.
2. If you have an earlier version of the guidance installed on your system, use the uninstall script to remove it. See [Removing Previous Versions of the Guidance](#).
3. If your SharePoint installation does not use the default SQL instance, modify the database name settings. See [Using a Different SQL Server Instance to Install the Guidance](#).
4. If you are installing the guidance on an Active Directory domain controller, modify the AppPoolId location.
5. In the <your installation directory>\Setup\PartnerPortal directory, right-click **ContosoSetup.bat**, and then click **Run as administrator**.
6. Configure SSL for the ContosoServices8585 Web site.
7. Verify that the guidance installed correctly.

The following topics describe each of these steps in detail.

Note:

The following procedures assume that you have downloaded the guidance.

Removing Previous Versions of the Guidance

The following procedure explains how to use the uninstall script to remove a previous version of the guidance. You must run the script as an administrator.

To use the uninstall script

1. Go to the directory in which you extracted the guidance.
2. In the Setup\PartnerPortal directory, right-click **ClearContosoSetup.bat**, and then click **Run as administrator**.
3. Select the administrator account that you want to use, and then type the password.

If an error occurs during the uninstall process, see [Uninstalling the Partner Portal Application](#), which explains how to uninstall the guidance manually.

Note:

You can also use the uninstall script if an installation fails.

Using a Different SQL Server Instance to Install the Guidance

This topic explains how to install the SharePoint guidance if your SharePoint installation does not use the default SQL Server instance, **OFFICESERVERS**. You first must modify some of the settings in the 00_Parameters.bat file and then install the Partner Portal application by running ContosoSetup.bat.

To modify the database name in the 00_Parameters.bat file

1. Go to the \your directory\Setup\PartnerPortal\SupportingFiles\ directory, where *your directory* is the location in which you extracted the SharePoint guidance.
2. If your SharePoint installation does not use the default Instance name **OFFICESERVERS**, you must modify the file as follows:
 - a. Open the 00_Parameters.bat file.
 - b. Modify one of the following lines of code to contain the name of the existing instance.

```
REMset SQLserverName=%computename%\OFFICESERVERS
REMset SQLserverName=%computename%\SQLEXPRESS
REMset SQLserverName=%computename%
```

- c. Delete the **REM** keyword from that line. This sets the server instance to the file you specified.

Installing the Guidance on an Active Directory Domain Controller

This topic explains how to install the SharePoint guidance if your system is an Active Directory domain controller. First, you must modify some of the settings in the 00_Parameters.bat file and then install the Partner Portal application by running ContosoSetup.bat.

To modify the AppPoolId Location in the 00_Parameters.bat file

1. Go to the \your directory\Setup\PartnerPortal\SupportingFiles\ directory, where *your directory* is the location in which you extracted the SharePoint guidance.
2. If your server is an Active Directory domain controller, remove the **REM** keyword from the following line of code.

```
REM set AppPoolIdLocation=%USERDOMAIN%
```

3. Add the **REM** keyword to the following line of code.

```
set AppPoolIdLocation=%computename%
```

After you install the guidance, you must modify the web.config files for the Contoso.LOB.Services project and the

Contoso.PartnerPortal.Services project by supplying the domain name for the ContosoTrustedAccounts group and the ContosoSPTrustedAccounts group.

To supply the domain name in the web.config file

1. Go to the \your directory\Source\PartnerPortal\Contoso.LOB.Services directory, where *your directory* is the location of the SharePoint guidance.
2. Edit the web.config file. Look for the following XML under the **appSettings** node.

XML

```
<add key="TrustedAccountGroup" value="LocalMachineName\ContosoTrustedAccounts"/>
```

3. Replace **LocalMachineName** with the domain name of the ContosoTrustedAccounts group.
4. Save the web.config file.
5. Go to the \your directory\Source\PartnerPortal\Contoso.PartnerPortal.Services directory, where *your directory* is the location of the SharePoint guidance.
6. Edit the web.config file. Look for the following XML under the **appSettings** node.

XML

```
<add key="TrustedAccountGroup" value="LocalMachineName\ContosoSPTrustedAccounts"/>
```

7. Replace **LocalMachineName** with the domain name of the ContosoSPTrustedAccounts group.
8. Save the web.config file.
9. Restart IIS.

Configuring SSL for the ContosoServices8585 Web Site

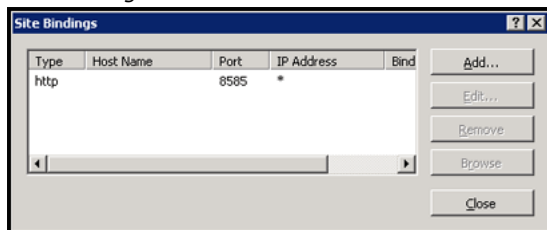
The following procedures explain how to configure the ContosoServices8585 Web site in IIS to use the temporary certificate created in the ContosoSetup batch script for Secure Sockets Layer (SSL) communication. This will enable SSL for the transport communication. There are different procedures for IIS version 7 and IIS version 6:

- If you are using IIS version 7, click [here](#) for procedures.
- If you are using IIS version 6, click [here](#) for procedures.

To install and configure SSL for IIS version 7

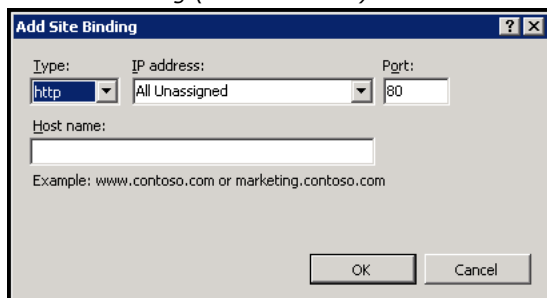
1. Click **Start**, and then click **Run**.
2. In the **Run** dialog box, type **inetmgr**, and then click **OK**.
3. In the **IIS Manager** dialog box, expand the *(local computer)* node, and then expand the **Sites** node.
4. Click **ContosoServices8585**.
5. In the **Actions** pane, click **Bindings**. This opens the Site Bindings editor as shown in the following illustration. Use the Site Bindings editor to create, edit, and delete bindings for your Web site.

Site Bindings editor



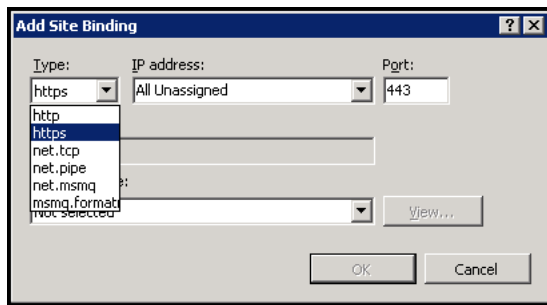
6. Click the **Add** button to add the SSL binding to the site. The default settings for new bindings are **Type http** on **Port 80**, as shown in the following illustration.

Add Site Binding (default values)



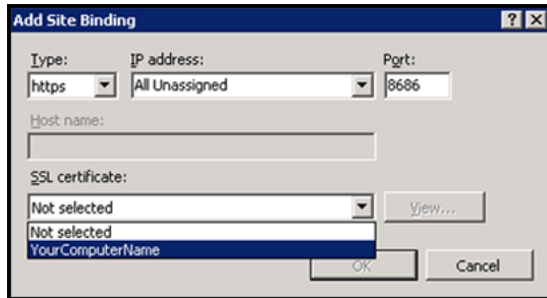
7. In the **Type** drop-down list box, select **https**. The default port for new https type bindings is **443**.

Add Site Binding (https default values)



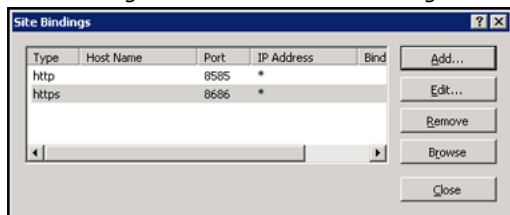
8. Change the **Port** value to **8686**.
9. In the **SSL certificate** drop-down list box, select the certificate that has your computer name, as shown in the following illustration. (The certificate is automatically created by the ContosoSetup.bat file.)

SSL certificate selection



10. Click **OK** to return to the Site Bindings window, as shown in the next illustration.

Site Bindings window with new binding



11. Click **Close** and then restart IIS.

To install and configure SSL for IIS version 6

1. Click **Start**, and then click **Run**.
2. In the **Run** dialog box, type **inetmgr**, and then click **OK**.
3. In the **IIS Manager** dialog box, expand the *(local computer)* node, and then expand the **Web Sites** node.
4. Right-click **ContosoServices8585 Web site** and then click **Properties**.
5. In the **ContosoServices8585 Web Site Properties** dialog box, click the **Directory Security** tab, and then in the **Secure Communications** section, click **Server Certificate**.
6. When the Web Server Certificate Wizard welcome screen appears, click **Next**.
7. On the Server Certificate screen, select **Assign an existing certificate**, and then click **Next**.
8. On the Available Certificates screen, select the certificate that has your computer name. (The certificate is created automatically during setup.) The **Intended Purpose** should be **Server Authentic**. Click **Next**.
9. Change the port number from **443** to **8686**. Click **Next**.
10. Verify the information on the Certificate Summary screen, and then click **Next**.
11. Click **Finish** to complete the certificate installation.
12. In the **ContosoServices8585 Web Site Properties** dialog box, click **OK**.
13. Restart IIS.

Uninstalling the Partner Portal Application

If the Partner Portal application installation is unsuccessful, use the following procedures to remove all the SPG components. (Some of these steps might not apply to your situation.) Another approach is to run the uninstall batch files that are described in [Details of the Uninstall Scripts](#).

Note:

Depending on how the SharePoint Guidance was installed, you might have to use the Contoso Admin account to delete the service. If your default account fails to delete the service when you use the following steps, log on as the Contoso Admin and try again.

Use the following credentials to log on as the Contoso Admin:

User Name: **ContosoWinAdmin**

Password: **P2ssw0rd\$**

Removing the SPG Components

Remove the components as follows (click the link to go to the complete procedure):

1. [Delete the ContosoSSP shared service.](#)
2. [Delete the Contoso-related Web applications.](#)
3. [Delete the Contoso-related WSP solutions.](#)
4. [Check whether the components were removed. If they were not, remove the components manually.](#)
5. [Delete the Contoso-related application pools.](#)
6. [Delete the Contoso-related certificate.](#)

To delete the ContosoSSP shared service

1. Open SharePoint Central Administration.
2. Click **Shared Services Administration**.
3. On the **Manage this Farm's Shared Services** list, select **ContosoSSP**.
4. On the drop-down menu, select **Delete**.
5. Delete all databases associated with the ContosoSSP shared service.

To delete the Contoso-related Web applications

1. Open SharePoint Central Administration.
2. Click **Application Management**.
3. Under **SharePoint Web Application Management**, click **Delete Web Application**. The Delete Web Application page appears.
4. On the Delete Web Application page, select **Change Application** on the **Web Application** drop-down menu. The Select Web Application page appears.
5. Select the **ContosoSSPWeb** application. The Delete Web Application page appears.
6. Select **Yes** to delete the content databases, and select **Yes** to delete the IIS Web sites. Click **Delete**.
7. Return to the Select Web Application page, and then complete steps 4 – 6 for the ContosoWeb application.

To delete and remove Contoso-related WSP solutions

1. Open SharePoint Central Administration.
2. Click **Operations**.
3. Under **Global Configuration**, click **Solution Management**. The Solution Management page appears, displaying a list of solutions.
4. Remove the following Contoso-related Web solution package (WSP) solutions:

- ContosoCommon.wsp
- ContosoPartnerPortal.wsp
- ContosoPartnerPortalProductCatalog.wsp
- ContosoPartnerPortalPromotions.wsp
- ContosoPartnerPortalCollaboration.wsp
- ContosoPartnerPortalCollaborationOrderException.wsp
- MicrosoftPracticesSPGSubSiteCreation.wsp
- ContosoPartnerPortalCollaborationIncident.wsp
- MicrosoftPracticesSPGAJAXSupport.wsp
- ContosoPartnerPortalPartnerCentral.wsp
- ContosoPartnerPortalPartnerDirectory.wsp
- ContosoLOBServicesClient.wsp

Repeat the following steps for each WSP file:

- Click the WSP name. The Solution Properties page appears.
- Click **Retract Solution**. The Retract Solution page appears.
- Click **OK** to remove the WSP. When the deletion is completed, the Solution Management page appears.
- Confirm that the file is no longer deployed, and then click the WSP name again. The Solutions Properties page appears.

- Click **Remove** to remove the solution from the list. The Solution Management page will appear.
- Wait several seconds to allow the WSP to be removed, and then refresh the page. The WSP will no longer appear on the list.

To confirm that the components were deleted

1. Confirm that the Contoso virtual directories were deleted. Using Windows Explorer, go to C:\Inetpub\wwwroot\wss\VirtualDirectories. Check whether the following Contoso virtual directories are listed:
 - 9001
 - 9002
 - 9004
 - 9005

If any of the directories are listed, delete them.

2. Confirm that the Contoso Web sites were deleted. In IIS Manager, expand the **Web Sites** node. Check whether the following Contoso Web sites are listed:
 - ConsosoService8585
 - ContosoSSPWeb
 - ContosoWeb
 - SharePoint 9002
 - SharePoint 9005

If any of the Web sites are listed, delete them.

To delete the Contoso related application pools

- Open IIS Manager, and then delete the following application pools:
 - ContosoApplicationPool
 - ContosoSSPAppPool
 - ContosoWEBAppPool
 - ContosoServiceAppPool

To delete the Contoso-related certificate

1. Click **Start**, click **All Programs**, click **Accessories**, and then click **Run**.
2. In the **Open** box, type **mmc**, and then click **OK**. A Microsoft Management Console (MMC) window appears.
3. On the **File** menu, select **Add/Remove Snap-in**. The Add/Remove Snap-ins window appears.
4. Click **Certificates**, and then click **Add**. The Certificates snap-in window appears.
5. Select **Computer account**, and click **Next**. The Select Computer window appears.
6. Select **Local computer**, and then click **Finish**.
7. Click **OK** to close the Add/Remove Snap-ins window and return to the Console Root window.
8. In the tree control, expand **Trusted Root Certification Authorities**, and then click **Certificates**.
9. Delete the **ComputerName** row.
10. Restart IIS.

Removing Packages from SharePoint

In some situations, a user might receive an error when he or she uses Central Administration to deploy or remove a WSP solution package. If this occurs, you have to remove the package that caused the error.

To remove the package

1. Open a Command Prompt window, and then change the directory to c:\Programs Files\Common Files\Microsoft Shared\Web Service Extensions\12\Bin.
2. At the command prompt, type the following:
stsadm -o displaysolution -name solution_name
 where **solution_name** is the name of the solution that you want to delete.
3. Type **Get SolutionDeploymentJobId**, and then press ENTER.
4. To cancel the solution, type the following, and then press ENTER:
**C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN>stsa
 dm -o canceldeployment -id << solutionDeploymentJobId>>**
5. To delete the solution, type the following, and then press ENTER:
**C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN>stsadm -o
 deletesolution -namesolution_name**
 where **solution_name** is the name of the solution that you want to delete.

Setting Up the Extranet Host Header

Instead of accessing the extranet by using the address `http://servername:9002`, you can set up a host header on your local development server, `http://extranet.contoso.com`.

Note:

If you are using a production server, the first two steps are unnecessary.

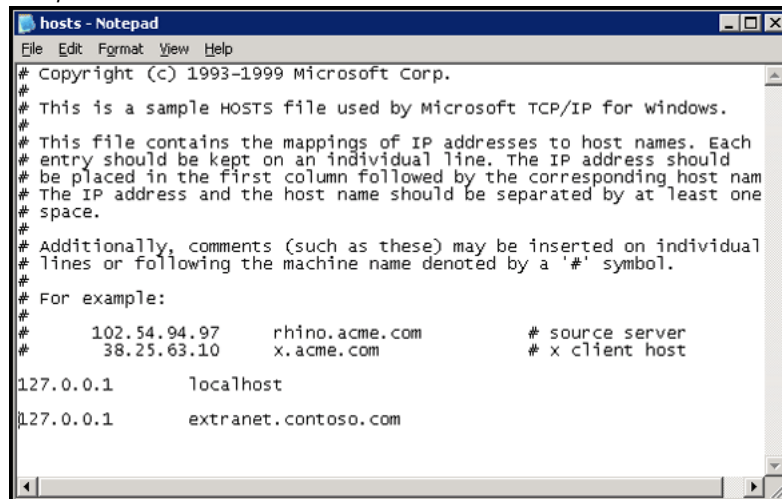
To set up an extranet host header

1. Add the host header to the host file as follows:
 - a. In a text editor such as Notepad, open the following file: `C:\Windows\System32\drivers\etc\hosts`
 - b. Add the following at the end of the file:

127.0.0.1 extranet.contoso.com

The host file contents should be similar to the following illustration.

Sample hosts file



2. Associate the host header name with the extranet version of ContosoWeb as follows:
 - a. Start IIS Manager.
 - b. Under **Sites**, select **SharePoint - 9002**.
 - c. In the **Actions** pane on the right, select **Bindings**. The Site Bindings window appears.
 - d. Click **Add**, and then type **extranet.contoso.com** in the host name box.
3. Disable the loopback check.

Because you are using a fully qualified domain name to access a site on your local development server, you might encounter authentication issues. If this occurs, perform Method 2 described in the following Microsoft Help and Support article:

[Error message when you try to access a server locally by using its FQDN or its CNAME alias after you install Windows Server 2003 Service Pack 1: "Access denied" or "No network provider accepted the given network path"](#)

4. If your Web browser is configured to use a proxy server, add **extranet.contoso.com** to the list of exceptions. Follow these steps:
 - a. Start Internet Explorer.
 - b. On the **Tools** menu, click **Internet Options**, and then click the **Connections** tab.
 - c. Click **LAN settings**.
 - d. In the **Proxy server** section, click **Advanced**.
 - e. In the Exceptions text box, add **extranet.contoso.com**, and then click **OK**.

You should now be able to access the extranet portal by using the following URL:
`http://extranet.contoso.com/sites/ProductCatalog/default.aspx`.

Installation Scripts

The Setup\PartnerPortal directory contains following scripts, which orchestrate the setup and removal of the SharePoint Guidance:

- **ContosoSetup.bat.** This script installs the guidance by sequentially calling the installation scripts.
- **ClearContosoSetup.bat.** This script uninstalls the guidance by sequentially calling the uninstall scripts.

The install and uninstall scripts are located in the following directories:

- **SetupSource.** This script contains a solution that holds the C# code that is invoked by the BAT files.
- **SupportingFiles.** This script contains the .bat files, Visual Basic Scripting Edition (VBScript), JavaScript, and SQL files.

Details of the Installation Scripts

The following table describes the installation scripts called by ContosoSetup.bat. (The scripts are listed in the order in which they are called.)

No.	Name of script	What it does
1	00_Parameters.bat	Called first in every .bat script. This defines the global variables.
2	01a_CreateTestCertificate.bat	Used to uninstall and install a certificate.
3	01b_CreateWindowsUsers.bat	Uses the Visual Basic CreateLocalAccounts.vbs script to complete the following tasks: - Creates the following Windows user group accounts: - ContosoWinPartner1 - ContosoWinPartner2 - ContosoSPTrustedAccounts - ContosoTrustedAccounts - Creates the following Windows local users (the password for the users is P2ssw0rd\$): - ContosoWinAdmin - ContosoAppPoolUser - ContosoServiceUser - ContosoPartner1User6 - ContosoPartner1User7 - ContosoPartner1User8 - ContosoPartner1User9 - ContosoPartner2User6 - ContosoPartner2User7 - ContosoPartner2User8 - ContosoPartner2User9 - Adds the Windows local users listed in step 2 to the Windows groups, as follows: - Adds ContosoPartner1User6, 7, 8, and 9 to ContosoWinPartner1. - Adds ContosoPartner2User6, 7, 8, and 9 to ContosoWinPartner2. - Adds ContosoWinAdmin to ContosoWinPartner1, ContosoWinPartner2, WSS_WPG, WSS_ADMIN_WPG, WSS_RESTRICTED_WPG and Users. - Adds ContosoAppPoolUser to ContosoWinPartner1, ContosoWinPartner2, WSS_WPG and Users. - Adds ContosoServiceUser to ContosoSPTrustedAccounts, ContosoTrustedAccounts, WSS_WPG and Users.
4	02_CreateContosoServices.bat	- Creates ContosoServiceAppPool with Windows user ContosoServiceUser as the identity. - Creates ContosoSPServiceAppPool with Windows user ContosoAppPoolUser as the identity. - Creates the ContosoServices8585 Web site with port 8585, and sets the application pool to ContosoServiceUser. - Creates the ContosoSPServices8787 Web site with port 8787, and sets the application pool to ContosoAppPoolUser. - Creates the following virtual directories in the ContosoSPServices8787 Web site:

		• Contoso.PartnerPortal.Services - Creates the following virtual directories in the ContosoServices8585 Web site: • Contoso.LOB.Services • Contoso.LOB.Web - Enables the ServiceModel.
5	03a_BuildSolutions.bat	Uses the MsBuild command to build all projects.
6	03b_PackageSolutions.bat	Creates a WSP package that contains the SharePoint project and Visual Studio extensions for Windows SharePoint Services.
7	04a_CreateContosoWebandSsp.bat	- Creates the ContosoWeb SharePoint Web application, the ContosoWebAppPool IIS application pool and the http://localhost:9001/ site collection. - Creates the ConsotoSSPWeb SharePoint Web application and the ContosoSSPAppPool application pool. - Creates the ContosoSSP shared services provider in the ContosoSSPWeb application. - Associates ContosoWeb with ContosoSSP. - Adds ContosoWinpartner1, ContosoWinPartner2, and ContosoWinAdmin to ContosoWeb. - Gives the installer user full control of ContosoSSPWeb.
8	04b1_AddSolutions.bat	Adds the WSP solutions to the SharePoint solution store.
9	04b2_DeploySolutions.bat	Deploys solutions globally or deploys them on the SharePoint Web applications.
10	04c_ActivateAppFeatures.bat	Activates Ajax and the ContosoMasterPage features in http://localhost:9001.
11	05A_CreatePublishingPortal.bat	- Creates the ProductCatalog site collection and activate its features. - Creates the Promotions site collection and activates its features.
12	06_CreatePartnerSites.bat	- Creates site collection http://servername:9001/sites/Partner1. - Creates the site collection http://servername:9001/sites/Partner2. - Creates the /Incident/ subsite for the site collections created in steps 1 and 2. - Activates the features of the site collections created in steps 1 and 2. - Adds the subsite creation template incidentsubsite.stp to the global list.
13	06_CreatePartnerSites_AddUsers.bat	- Adds ContosoWinpartner1 and ContosoWinAdmin to http://localhost:9001/sites/Partner1. - Adds ContosoWinPartner2 and ContosoWinAdmin to http://localhost:9001/sites/Partner2.
14	07_CreatePartnerCentral.bat	- Creates http://localhost:9001/sites/partnerCentral site collection. - Creates the http://localhost:9001/sites/partnerCentral/SPGsubsite subsite. - Activates the features on the site collection and subsite created in steps 1 and 2. - Populates the test data in http://localhost:9001/sites/partnercentral/SPGsubsite.
15	07_CreatePartnerCentral_AddUsers.bat	Adds ContosoWinAdmin to the http://localhost:9001/sites/partnercentral site collection.
16	08_ModifyWebConfig.bat	Modifies C:\Inetpub\wwwroot\wss\VirtualDirectories\9001\web.config to support forms-based authentication.

17	09_ImportBDC.bat	Imports the Business Data Catalog (BDC) application definition \spgdropLocation\Source\PartnerPortal\ProductCatalogDefinition.xml into the shared services provider ContosoSSP.
18	10_EnableFBA.bat	Creates the ContosoFBADB database in <SQLServerName>\OFFICESERVERS as follows: - Creates the following two roles: - ContosoFbaPartner1 - ContosoFbaPartner2 - Creates the following 10 users (the password for the users is P2ssw0rd\$): - ContosoPartner1User1 - ContosoPartner1User2 - ContosoPartner1User3 - ContosoPartner1User4 - ContosoPartner1User5 - ContosoPartner2User1 - ContosoPartner2User2 - ContosoPartner2User3 - ContosoPartner2User4 - ContosoPartner2User5 - Adds users to roles as follows: - Adds ContosoPartner1User1, 2, 3, 4, and 5 to ContosoFbaPartner1. - Adds ContosoPartner2User1, 2, 3, 4, and 5 to ContosoFbaPartner2. - Extends Web application http://servername:9001/ to http://servername:9002/. - Extends Web application http://servername:9004/ to http://servername:9005/. - Gives BDC access to ContosoSSP http://servername:9004/. - Adds people picker, partners, partnergroups, and fbaconnectionstrings to the following Web.config files: - C:\Inetpub\wwwroot\wss\VirtualDirectories\9001\web.config - C:\Inetpub\wwwroot\wss\VirtualDirectories\9002\web.config - C:\Inetpub\wwwroot\wss\VirtualDirectories\centraladmpart\web.config - Adds ContosoFbaPartner1 and ContosoFbaPartner2 to the appropriate site collections. - Gives ContosoAppPoolUser access to ContosoFBADB Note: 10_EnableFBA.bat dynamically generates an SQL script (ServiceLoginPermissions.sql) that grants database rights to ContosoAppPoolUser. However, this is not a recommended production approach for configuring permissions.
19	11_RecycleAppPools.bat	Recycles the following: - AppPool Central Administration - AppPool ContosoWEBAppPool - AppPool ContosoSSPAppPool - AppPool ContosoServiceAppPool

Details of the Uninstall Scripts

The following table describes the uninstall scripts called by CleanContosoSetup.bat. (The scripts are listed in the order in which they are called.)

No.	Name of script	What it does
1	00_Parameters.bat	Called first in each .bat script. This script defines the global variables.
2	901a_CleanTestCertificate.bat	Uninstalls the certificate.
3	901b_CleanWindowsUsers.bat	Removes all Windows users that were created by the setup script.
4	902_CleanContosoServices.bat	Removes all Contoso services and their corresponding virtual directories and application pools.
5	904a_CleanContosoWebandSsp.bat	Removes all Contoso Web and SSP files from SharePoint.
6	904b1_DeleteSolutions.bat	Deletes all solutions from SharePoint.
7	904b2_RetractSolutions.bat	Retracts all solutions from SharePoint.
8	904c_DeActivateAppFeatures.bat	Deactivates features inside SharePoint.
9	904c_UninstallFeatures.bat	Uninstalls features from SharePoint.
10	905_CleanPublishingPortal.bat	Deletes the promotions site.
11	906_CleanPartnerSites.bat	Deletes Partners 1 and 2 and related incidents.
12	907_CleanPartnerCentral.bat	Deletes SPGSubSite, PartnerDirectory, and PartnerCentral.
13	910_CleanFBA.bat	Removes forms-based authentication.

Validating the Partner Portal Installation

After you install the SharePoint guidance, you can validate it in one of two ways. You can either browse to the relevant sites or you can use Build Verification Tests (BVT):

- [Validating the Sites](#). Use the browser to manually verify the installation.
- [Validating the Installation with Build Verification Tests](#). Use Visual Studio Team Edition for Software Testers or Visual Studio Team System to run the automated tests included in the distribution.

Validating the Sites

The rest of this section explains how to validate an installation by using the browser and performing some tasks. You will need to validate the following sites and services (use the procedures that follow this list to validate each service or site):

Note:

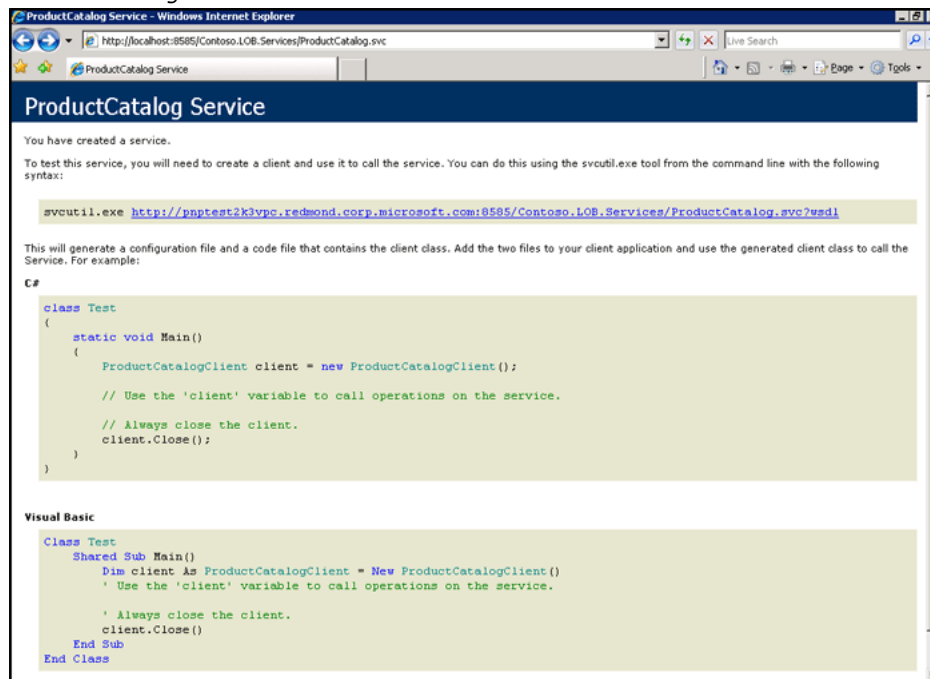
The password for all Contoso user accounts is **P2ssw0rd\$**.

- <http://localhost:8585/Contoso.LOB.Services/ProductCatalog.svc>
- <http://localhost:8585/Contoso.LOB.Services/Pricing.svc>
- <http://localhost:8585/Contoso.LOB.Services/IncidentManagement.svc>
- <http://localhost:8787/Contoso.PartnerPortal.Services/IncidentSite.svc>
- <http://localhost:8787/Contoso.PartnerPortal.Services/SubSiteCreation.svc>
- <http://localhost:9001/sites/ProductCatalog/default.aspx>
- <http://localhost:9001/sites/Promotions/default.aspx>
- <http://localhost:9001/sites/partnercentral/Default.aspx>
- <http://localhost:9001/sites/PartnerCentral/SpgSubsite/default.aspx>
- <http://localhost:9001/sites/partner1>
- <http://localhost:9001/sites/partner2>
- <http://computername:9002/sites/partner1> (please replace *computername* with the name of your computer)
- <http://computername:9002/sites/partner2> (please replace *computername* with the name of your computer)
- <http://localhost:8585/Contoso.LOB.Web/NewIncidentSite.aspx>
- <http://localhost:8585/Contoso.LOB.Web/CreateOrderExceptionSite.aspx>

To validate the Partner Portal Services

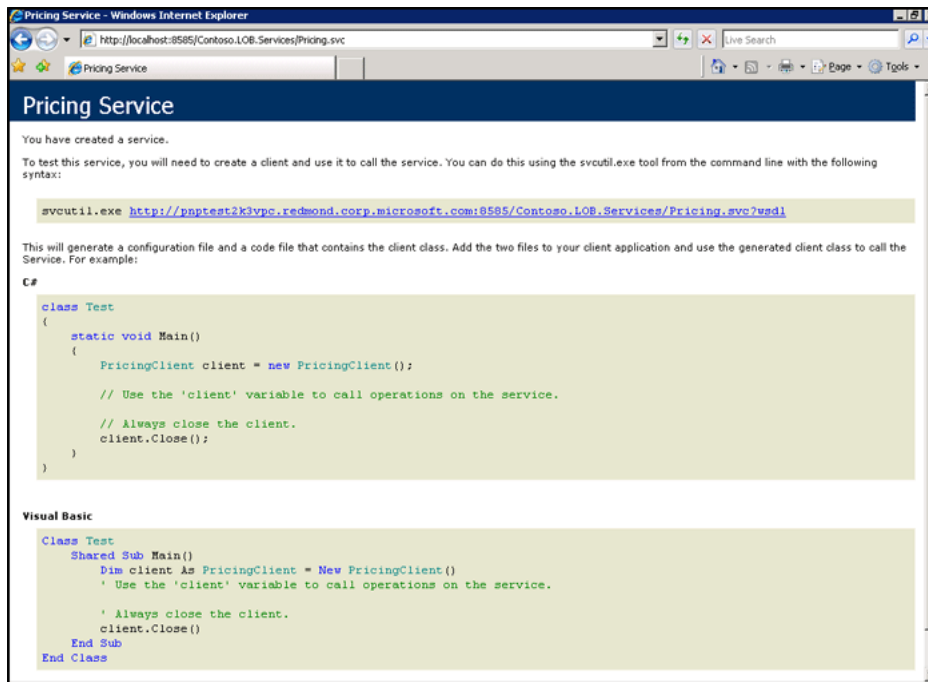
1. Make sure that the Product Catalog Service is correctly installed. Use the browser to navigate to <http://localhost:8585/Contoso.LOB.Services/ProductCatalog.svc>. You should see the following Web page.

Product Catalog Service



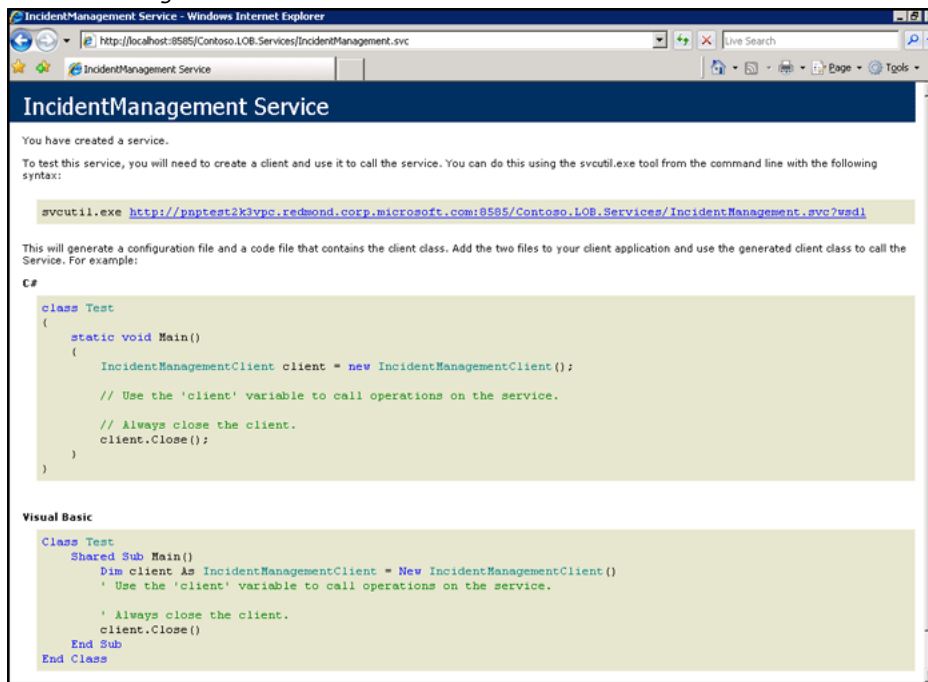
2. Make sure that the Pricing Service is correctly installed. Use the browser to navigate to <http://localhost:8585/Contoso.LOB.Services/Pricing.svc>. You should see the following Web page.

Pricing Service



3. Make sure that the Incident Management Service is correctly installed. Use the browser to navigate to <http://localhost:8585/Contoso.LOB.Services/IncidentManagement.svc>. You should see the following Web page.

Incident Management Service



4. Make sure that the Incident Site Service is correctly installed. Use the browser to navigate to <http://localhost:8787/Contoso.PartnerPortal.Services/IncidentSite.svc>. You should see the following Web page.

Incident Site Service

IncidentSite Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://pptest1k8vps.redmond.corp.microsoft.com:8787/Contoso.PartnerPortal.Services/IncidentSite.svc?wsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

```
C#
class Test
{
    static void Main()
    {
        IncidentSiteClient client = new IncidentSiteClient();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
        client.Close();
    }
}
```

Visual Basic

```
Class Test
Shared Sub Main()
    Dim client As IncidentSiteClient = New IncidentSiteClient()
    ' Use the 'client' variable to call operations on the service.

    ' Always close the client.
    client.Close()
End Sub
End Class
```

5. Make sure that the Subsite Creation Service is correctly installed. Use the browser to navigate to <http://localhost:8787/Contoso.PartnerPortal.Services/SubSiteCreation.svc>. You should see the following Web page.

Subsite Creation Service

SubSiteCreation Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://pptest1k8vps.redmond.corp.microsoft.com:8787/Contoso.PartnerPortal.Services/SubSiteCreation.svc?wsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

```
C#
class Test
{
    static void Main()
    {
        SubSiteCreationClient client = new SubSiteCreationClient();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
        client.Close();
    }
}
```

Visual Basic

```
Class Test
Shared Sub Main()
    Dim client As SubSiteCreationClient = New SubSiteCreationClient()
    ' Use the 'client' variable to call operations on the service.

    ' Always close the client.
    client.Close()
End Sub
End Class
```

Use the next procedure to validate the Partner Portal Web sites.

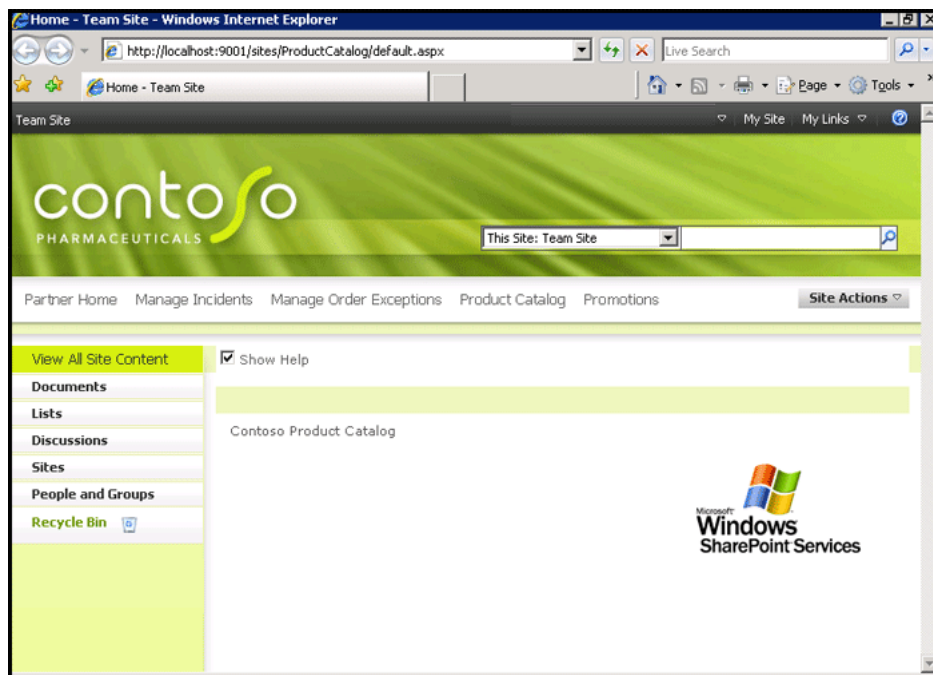
To validate the Partner Portal Web sites

1. Make sure that the Product Catalog site is correctly installed. Use the browser to navigate to <http://localhost:9001/sites/productcatalog/default.aspx>. You should see the following Web page.

Note:

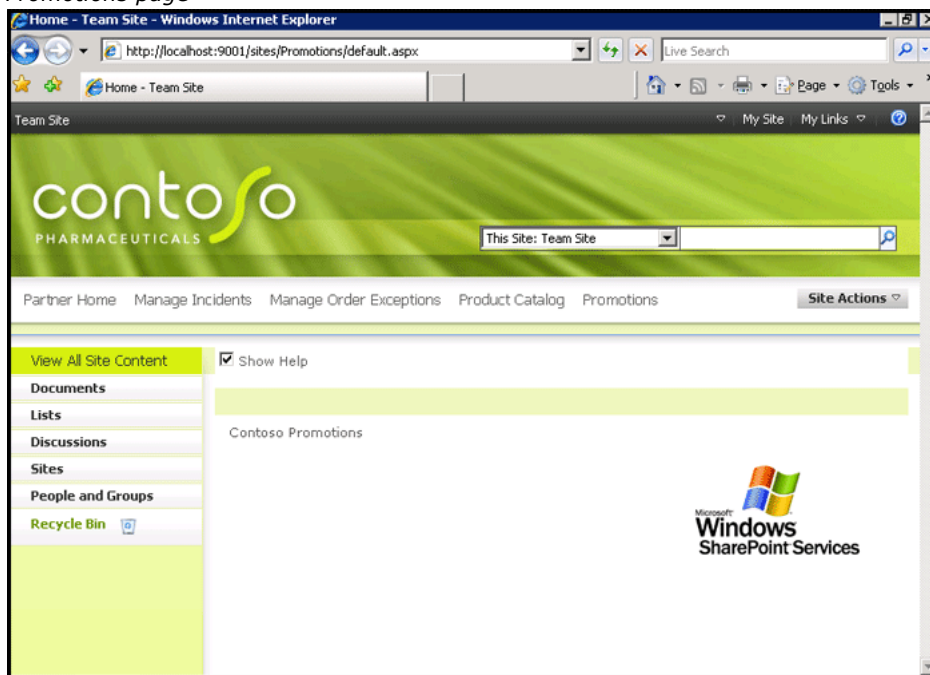
Note: You can also view this page by logging on as a partner (for example, ContosoPartner1User6 or ContosoPartner2User6).

2. *Product Catalog page*



- 3.
4. Make sure that the Promotions site is correctly installed. Use the browser to navigate to <http://localhost:9001/sites/Promotions/default.aspx>. You should see the following Web page.

Promotions page

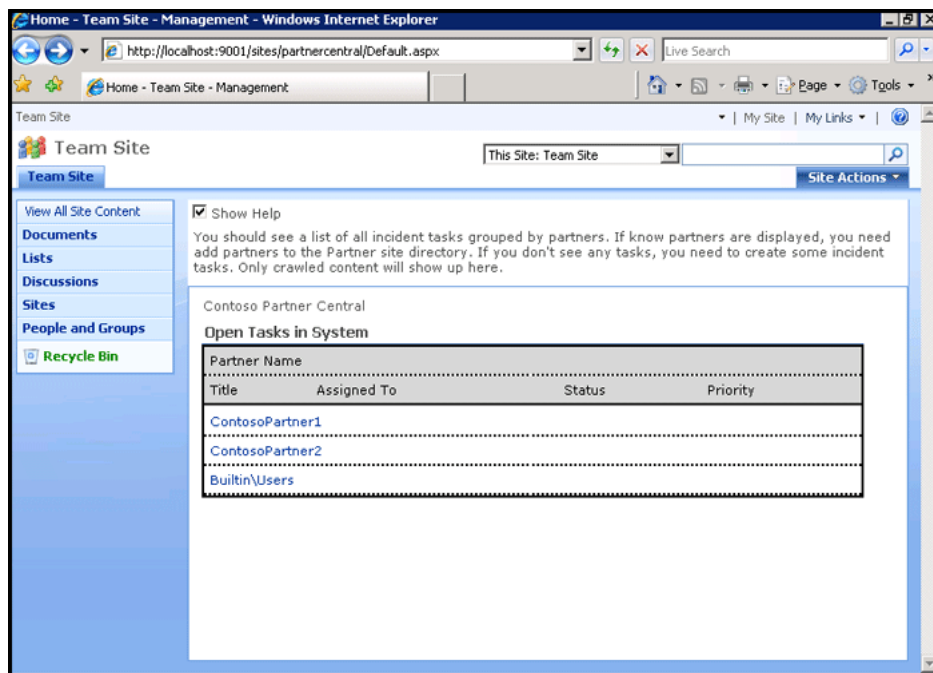


5. Make sure that the Partner Central site is correctly installed. Use the browser to navigate to <http://localhost:9001/sites/partnercentral/Default.aspx>. You should see the following Web page.

Note:

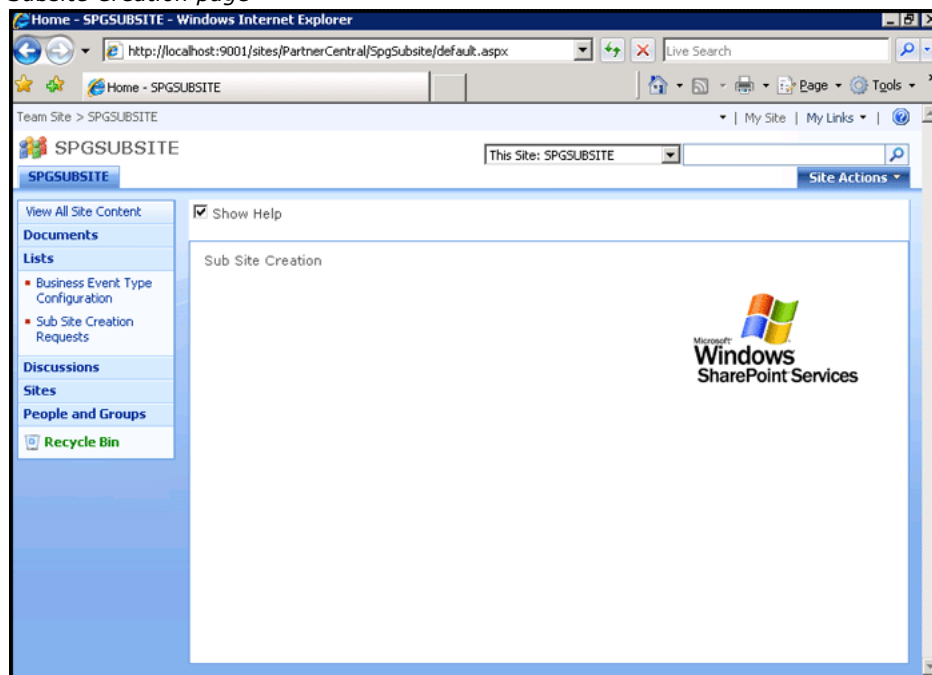
Note: You can also view this page by logging on with the ContosoWinAdmin account.

6. *Contoso Partner Central site*



- 7.
8. Make sure that the Subsite Creation site is correctly installed. Use the browser to navigate to <http://localhost:9001/sites/PartnerCentral/SpGSubsite/default.aspx>. You should see the following Web page.

Subsite Creation page



Use the next procedure to validate the Internal Partners Web sites.

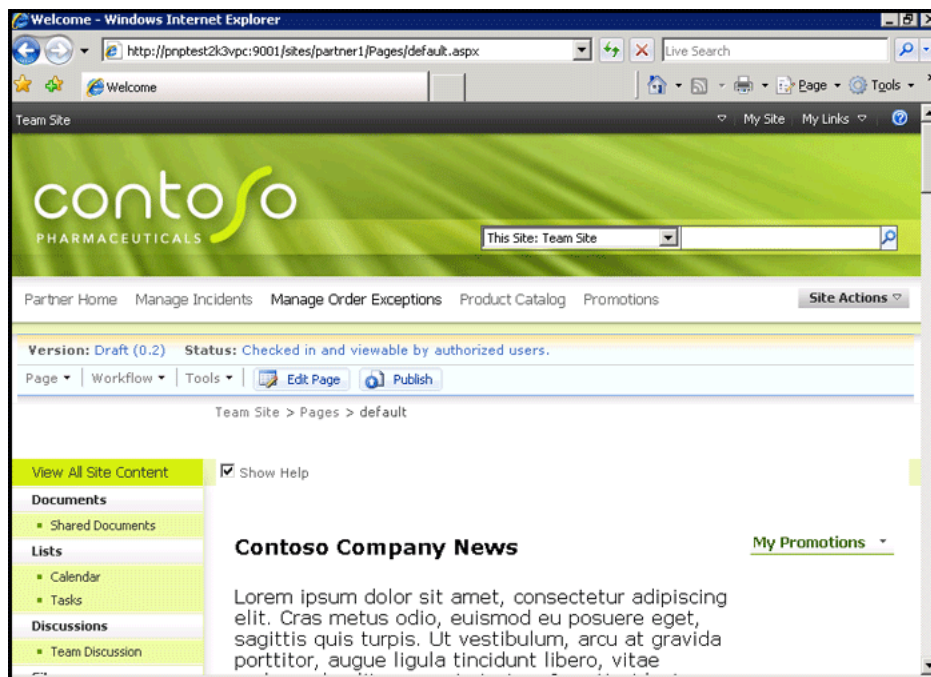
To validate the Internal Partners Web sites

1. Make sure that the Partner 1 Web site is installed correctly. Use the browser to navigate to <http://localhost:9001/sites/partner1>. You should see the following Web page.

Note:

Note: You can also view this page by logging on with a Partner1 account (for example, ContosoPartner1User6).

2. *Internal Partner1 Web page*

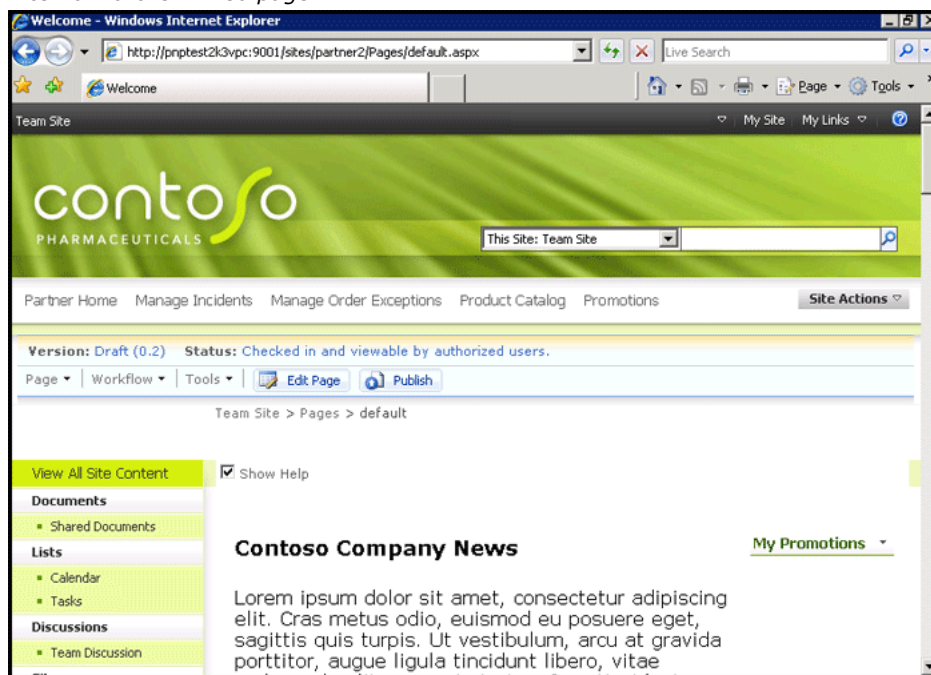


- 3.
4. Make sure that the Partner 2 Web site is correctly installed. Use the browser to navigate to <http://localhost:9001/sites/partner2>. You should see the following Web page.

Note:

You can also view this page by logging on with a Partner2 account (for example, ContosoPartner2User6).

5. *Internal Partner2 Web page*



- 6.

Use the next procedure to validate the External Partners Web sites.

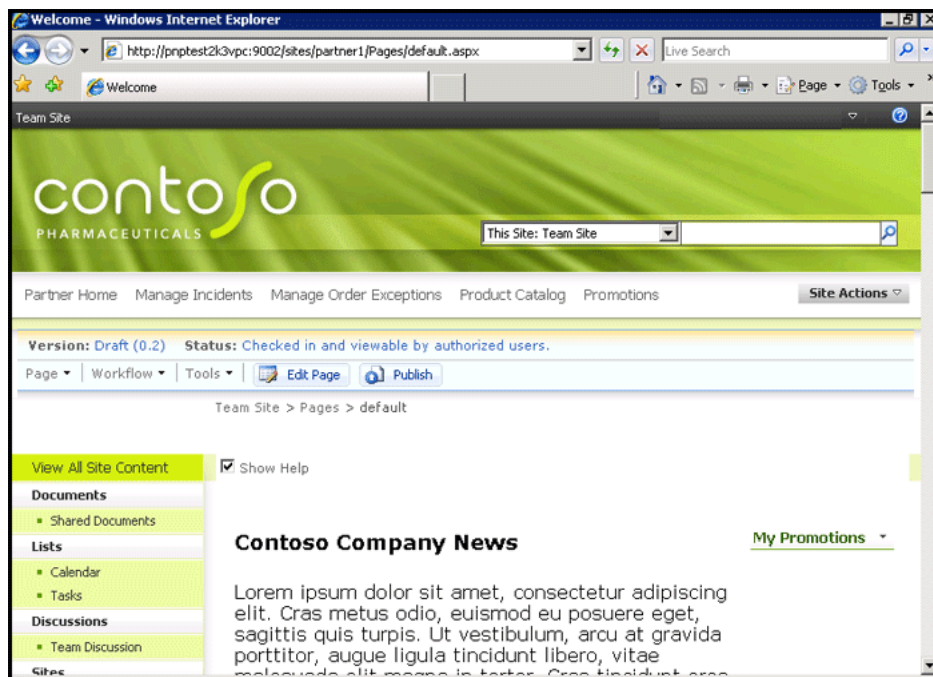
To validate the Internal Partners Web sites

1. Make sure that the Partner 1 Web site is correctly installed. Use the browser to navigate to <http://computername:9002/sites/partner1>, but replace *computername* with the name of the targeted computer. You should see the following Web page.

Note:

You can also view this page by logging on with a Partner1 FBA account (for example, ContosoPartner1User1).

2. *External Partner1 Web page*

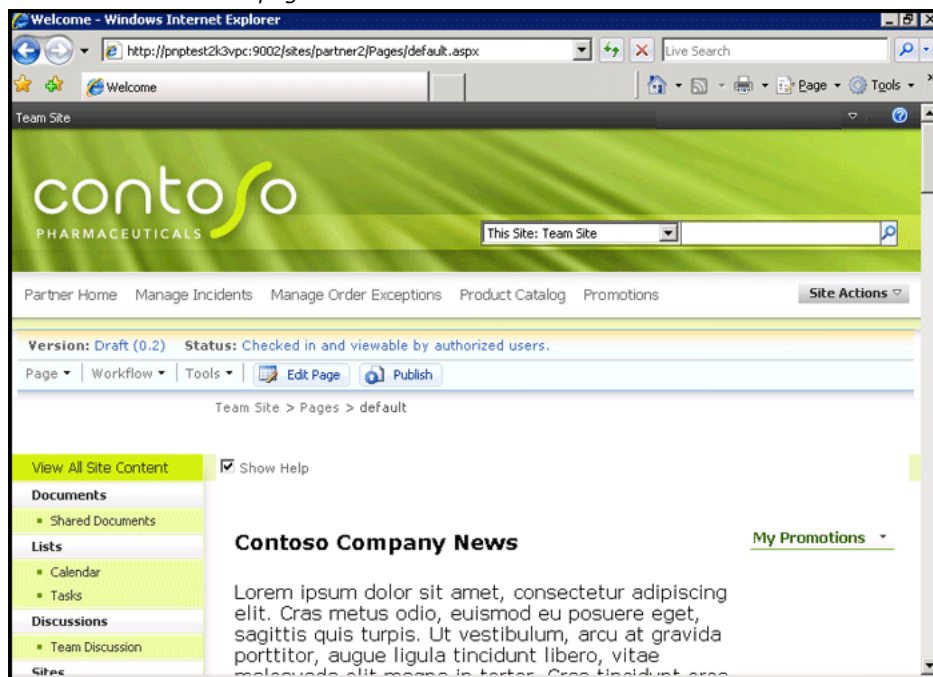


- 3.
4. Make sure that the Partner 2 Web site is correctly installed. Use the browser to navigate to <http://computername:9002/sites/partner2>, but replace *computername* with the name of the targeted computer. You should see the following Web page.

Note:

You can also view this page by logging on with a Partner2 FBA account (for example, ContosoPartner2User1).

5. *External Partner2 Web page*

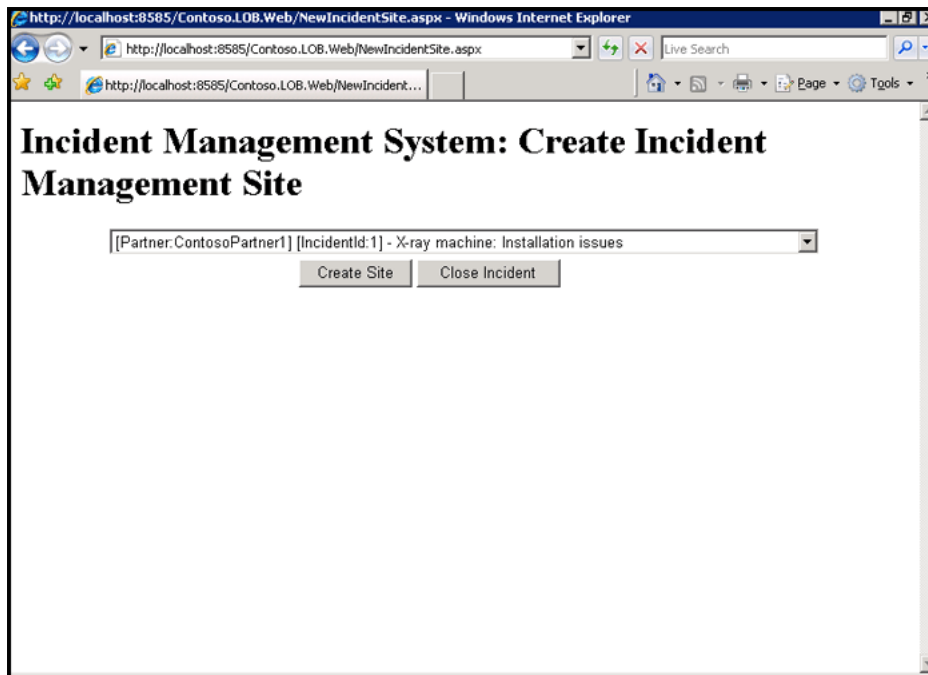


- 6.

Use the next procedure to validate the Services Web sites.

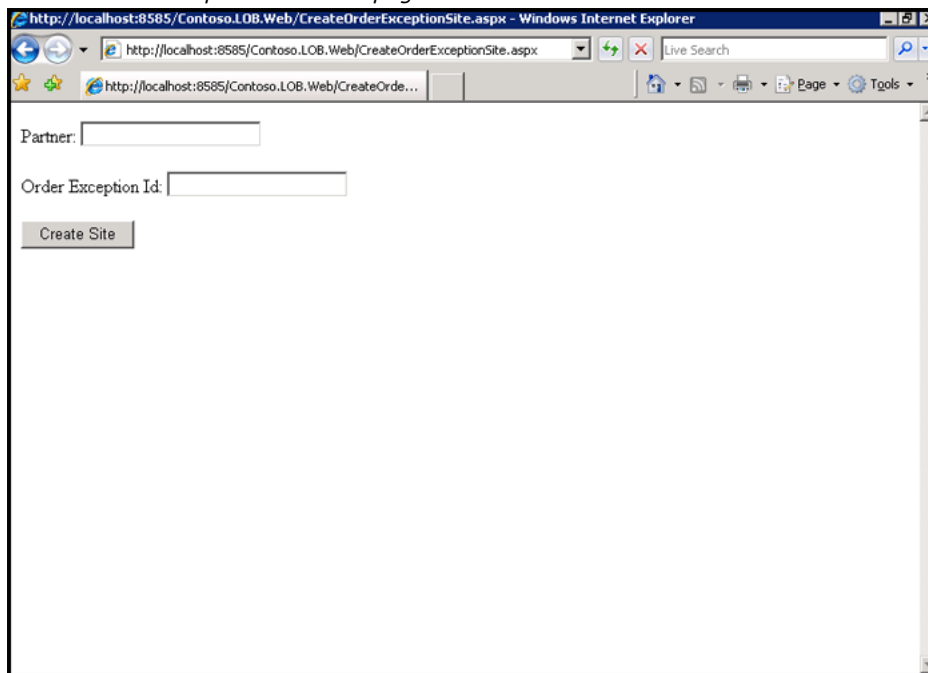
To validate the Services Web sites

1. Make sure that the Create Incident Management site is correctly installed. Use the browser to navigate to <http://localhost:8585/Contoso.LOB.Web/NewIncidentSite.aspx>. You should see the following Web page.
Create Incident Management page



2. Make sure that the Create Order Exception site is correctly installed. Use the browser to navigate to <http://localhost:8585/Contoso.LOB.Web/CreateOrderExceptionSite.aspx>. You should see the following Web page.

Create Order Exception Site Web page



Validating the Installation with Build Verification Tests

The Build Verification Tests (BVTs) offer a simple way to validate the SharePoint Guidance 2 installation and operations. The BVTs validate the following functions:

- Contoso LOB Services
- Contoso Partner Portal Services
- Partner 1 Web site
- Partner 2 Web site
- Portal Web site
- Catalog pages
- Product pages

Running the BVTs

To run the BVTs, you need to have the Visual Studio version that has the Test SKU, including the Visual Studio Team Server (VSTS) suite.

To run the BVTs

1. Start Visual Studio, and then browse to the solution (*drive* : \spgdroplocation\Test\PartnerPortal\Contoso.PartnerPortal.BVT.sln).
2. Select the file, and then click **Open**.
3. In the Test Manager window, click **By Test List** to display the test list pane.
4. Select the entire test list.
5. Run the tests.

The test will take several minutes to run. All tests should pass.

For more information, see [Visual Studio Team System Team Edition for Testers](#) on MSDN.

Installing the Training Management Application

This topic includes procedures that explain how to install the Contoso Training Management application. The first procedure describes the steps you must complete in Visual Studio. The second procedure describes the steps you must complete in SharePoint and in the operating system to configure the SharePoint Web site and create the local users. The last procedure describes how to use SharePoint to associate the workflow with the registration list.

You must have administrative privileges for both the SharePoint server and the SharePoint site on which you deploy the application.

To build and deploy the Training Management application in Visual Studio

1. Unzip the SharePoint Guidance self-extracting zip file.
2. In Visual Studio 2008, open <<Extracted folder>>\Source\TM\Contoso.TrainingManagement.RI\Contoso.TrainingManagement.RI.sln.
3. Right-click the **Contoso.TrainingManagement** project, and then click **Properties**. Click **Debug**.
4. Specify the URL of the site collection on which the project will be deployed. The default location is <http://localhost>.
5. Right-click the **Contoso.TrainingManagement.RI** solution, and then click **Build Solution**.
6. Right-click the **Contoso.TrainingManagement** project, and then click **Deploy**. Make sure that the Contoso.TrainingManagement/pkg folder and the Contoso.TrainingManagement /bin/Debug folder are not read-only.

Note:

If you are using User Account Control, you might need to use the **Run as Administrator** option when you open Visual Studio. Also, you might need to configure the Visual Studio extensions for Windows SharePoint Services Application Pool account to be a member of the Site Collection Administrators group. To configure the account, browse to http://localhost/_layouts/mngsiteadmin.aspx, and then specify the Application Pool account.

The next procedure describes the steps that you must complete in the operating system and in SharePoint.

To configure the Training Management application and create the users

1. Open a command prompt window, and then run the following Internet Information Services (IIS) command-line utility:
Isreset.exe
2. In Internet Explorer Address bar, enter the URL for your SharePoint server.
3. Click the **Site Actions** drop-down box. Click **CreateSite**. The New SharePoint Site page opens.
4. In the **Title** box, type **Training Management**. In the **URL name** box, type **training**. Use the Contoso Training Management site template that is located on the **Contoso** tab. Click **Create** to create the site.
5. In Windows, create two local Windows users named spgmanager and spgemployee.
6. On the SharePoint site, click the **Site Actions** tab. Click **SiteSettings**.
7. In the Users and Permissions column, click **People and groups**.
8. On the People and Groups page, click **Groups**.
9. Click the **Contoso Employees** group, click **New**, and then type **spgemployee**.
10. In the **Give Permission** section, click **Contoso Employees [Read, Limited Access]**.
11. Click **OK**.
12. Repeat steps 9 through 12 for the **Contoso Managers** group, typing **spgmanager** at the end of step 10.

To install the registration approval workflow

1. In Visual Studio 2008, right-click the **Contoso.TrainingManagement.Workflows.RegistrationApproval** project, and then click **Properties**. Click **Debug**.
2. Specify the URL of the Training Management site that you created in step 4 of the first procedure.
3. Right-click the **Contoso.TrainingManage.Workflows.RegistrationApproval** project, and then click **Deploy**. Make sure that the Contoso.TrainingManagement.Workflows.RegistrationApproval/pkg folder and Contoso.TrainingManagement.Workflows.RegistrationApproval/bin/Debug folder are not read-only.

Developing SharePoint Applications

The SharePoint platform enables you to develop applications that have the following features:

- **Scalable.** SharePoint provides an extensible framework that allows you to add applications and features as needed to meet growing demands.
- **Collaborative.** SharePoint includes features that allow you to create workspaces where people share ideas, documents, and other assets.
- **Content-driven.** SharePoint allows you to create and publish content that is tailored to your users' needs.

This guidance describes many of SharePoint's features. In addition, it demonstrates how to apply design and application patterns to SharePoint development. The guidance includes a *reference implementation* that is named the Partner Portal application, which shows how SharePoint capabilities are used in enterprise-scale applications. The reference implementation shows aspects of a SharePoint application that cannot be captured in smaller code samples. The guidance often refers to specific areas in the Partner Portal application to show how the guidance is implemented in an actual solution.

This topic contains the following two subtopics.

- [An Overview of the SharePoint Platform for ASP.NET Developers](#)
- [A Quick Tour of the Partner Portal Application](#)

To help developers who are new to SharePoint development, [An Overview of the SharePoint Platform for ASP.NET Developers](#) provides an overview of some key capabilities and concepts. It assumes that you have a working knowledge of ASP.NET application development. This topic does not provide a complete description of SharePoint. Instead, it highlights some of the most important SharePoint capabilities that extend features from ASP.NET. After you read it, you will be able to think of your development project in terms of what SharePoint offers you.

[A Quick Tour of the Partner Portal Application](#) provides an overview of the business scenario that motivated the development of the Partner Portal application. It points out aspects of the application that show why SharePoint was the best development environment to address those business challenges. Understanding the various elements of the Partner Portal application and how they map to specific SharePoint capabilities will help you to apply the guidance to your own applications.

Although both subtopics are far from exhaustive, they do illustrate why it makes sense to build many applications using SharePoint. Additional learning resources include the following:

- [Introduction to SharePoint Products and Technologies for the Professional .NET Developer](#)
- [SharePoint Architecture](#)
- [SharePoint Developer](#)

Note:

In addition to the Partner Portal application, the guide includes a supplemental reference implementation that is named the Training Management application. The Training Management application is a more introductory example of how to use basic SharePoint capabilities, such as lists and content types. For more information, see [Training Management Reference Implementation](#).

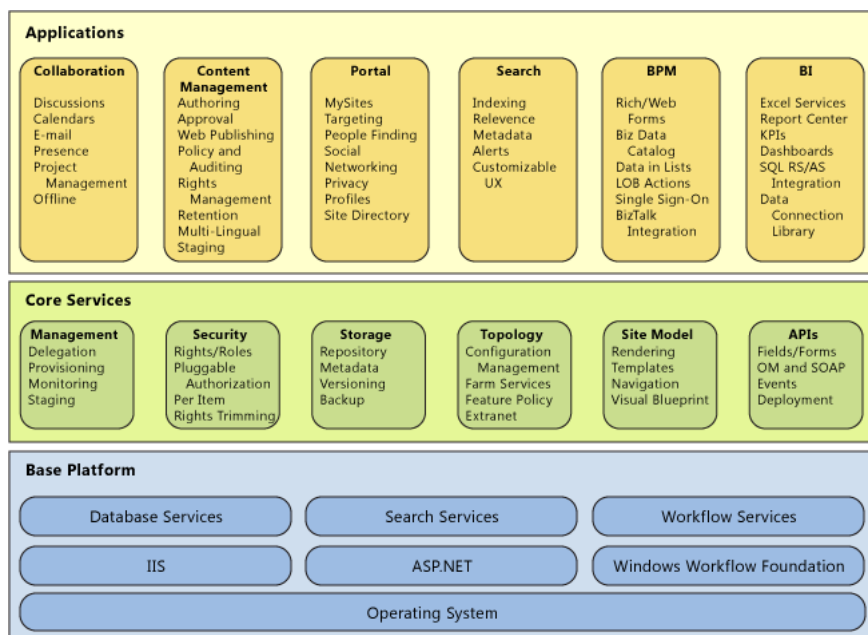
An Overview of the SharePoint Platform for ASP.NET Developers

When ASP.NET developers begin to learn about SharePoint development, they find many familiar concepts. SharePoint is built on top of the ASP.NET framework and uses many ASP.NET features including Web Parts, Web Part zones and master pages, security providers, session state managers, and user controls on ASPX pages. ASP.NET developers can view SharePoint as an extension of a framework that they already know.

SharePoint has two distinct layers. The base layer, which is named Windows SharePoint Services (WSS), is built on top of the ASP.NET framework. It provides the foundation for the ready-to-use collaborative application capabilities that are standard in SharePoint. It is also the foundation for custom Web-based portal and collaborative business applications. Windows SharePoint Services 3.0 is licensed as a component of Windows Server 2003 and Windows Server 2008. Microsoft Office SharePoint Server 2007 builds on top of ASP.NET and Windows SharePoint Services to provide enhanced core services and functioning, extensible applications that are meant for information workers.

The following illustration shows the SharePoint development platform.

The SharePoint development platform



There are many types of functionality available in SharePoint, particularly if you combine what is available in Windows SharePoint Services with what is available in Microsoft Office SharePoint Services. This topic focuses on the concepts that explain the core services. These services form the foundation for both standard and custom SharePoint applications. For more information, see [Introduction to SharePoint Products and Technologies for the Professional .NET Developer](#) on MSDN.

Information Hierarchies and Content Types

Two key concepts in SharePoint are containment hierarchies and content types. Containment hierarchies store and manage information and applications. The containment hierarchy for SharePoint is, from the top to the bottom, the farm, the Web application, the site collection, the site, the list and libraries, and the list items and folders. A containment hierarchy relates to a logical hierarchy that overlays a physical set of resources. For example, a farm is a grouping of physical resources that includes Web front-end servers, databases that provide logical application services (in SharePoint, these are termed shared services) along with configuration data, and a set of farm level management capabilities. The entire set of applications in the farm use these resources. For information about how the logical hierarchy relates to the physical topology, see [Server and Site Architecture](#) on MSDN.

Web applications are contained in a farm and correspond to the Internet Information Services (IIS) application pools that run on each Web front-end server in the farm. Web applications contain configuration data and provide a database boundary. A Web application also contains one or more site collections.

A site collection provides a data boundary, a scale boundary, an administrative boundary, and a permission boundary in the Web application. The site collection is a scale boundary as well because it is the smallest unit that is managed across databases. A site collection must be in a single database, but a Web application can have multiple databases for all the site collections it contains. A site collection contains a group of sites that are maintained in a hierarchy. The root site is termed the top-level site (it is also often termed the root Web). Sites are similar in concept to virtual directories in ASP.NET, although they are not equivalent. A site contains more functionality and is not actually a virtual directory. For example, in SharePoint, pages for a site are often rendered by merging file-based definitions, page templates, and content from the content database. This type of page is termed a site page or content page. The template can either be defined in a file or stored on the content database. SharePoint has application pages in addition to site pages. Application pages (these are also known as

_layouts pages) are conventional ASP.NET pages that are available to all SharePoint applications, and are less typical in a SharePoint application. For more information, see the related discussion in [SharePoint Architecture](#) and [SharePoint Page Types](#) on MSDN.

The logical hierarchies establish default behaviors for areas in SharePoint such as navigation and information presentation. A newly created SharePoint site is a reflection of a set of data that represents the site instance in the content database. Sites can contain subsites and sites also typically contain lists. Lists are a basic construct in SharePoint for containing, managing, and displaying information. Document libraries are considered a specialized type of list. Lists contain list items and folders.

Generally, with ASP.NET applications, a URL is a reflection of the files in your solution, and the URL typically refers to physical files in the Web site. It is not as straightforward in SharePoint. The URL is based on configured relationships that combine Web applications, site collections, and a hierarchy of contained sites as they are defined in the content database, although it is possible to override the default URL structure with managed paths. For information about the URL hierarchy and the relationship between the SharePoint logical and physical architectures, see [Site Collections and Web Applications](#), [SharePoint Architecture](#) and [Server and Site Architecture](#) on MSDN.

The site hierarchy is also an important element in a SharePoint application's information architecture. The site hierarchy establishes a site's navigational structure and URL structure. A logical information architecture leads to sites that are more intuitive to navigate and are easier to partition and scale as the sites grow. Although many aspects of the information hierarchy are determined by IT professionals instead of developers, the decisions that developers make often will direct (or limit) the decisions made by IT when they deploy the application or capability. Therefore, understanding information architecture is an important aspect of developing well-designed SharePoint applications. For a scenario-driven overview of how to develop an information architecture, see [Information architecture in Office SharePoint Server](#) on TechNet; this article is intended for both IT professionals and developers.

A concept that is related to the information architecture is *content types*. Content types are metadata that describe the attributes and actions associated with an entity that is managed within SharePoint. For example, a legal firm may create a content type named *Contract*. The attributes of the Contract content type define the metadata you want to associate with a document that is an actual contract. You might also want to associate workflows with the content type. These workflows run when a Contract document is created within a document library. Content types are powerful because they can be centrally managed, reused, and applied across different types of artifacts within SharePoint. For example, a content type can be associated with a list or attached to a document library. You must plan your content types in order to develop a logical information architecture. For more information about content types, see [Plan content types](#) on TechNet.

Deploying Applications to a Farm

SharePoint provides a scalable, shared infrastructure for deploying applications to a farm. You set up the farm once, add capacity as needed, and continue to add applications to the shared infrastructure. You only need to maintain one physical infrastructure for all SharePoint applications. However, using a distributed, shared infrastructure has its consequences. For example, it is more difficult to use Web.config files to manage configuration in SharePoint applications than in other types of applications. It is generally preferable to use the configuration mechanisms that are provided by SharePoint.

Although there are many concepts related to managing and deploying applications, this topic focuses on two of them: solutions and features. Both solutions and features help you to manage and deploy multiple applications that are within the SharePoint infrastructure.

SharePoint solutions are often referred to by their .wsp file name extension. Web solution packages (WSPs) package solution artifacts for an application or capability (in many cases, the artifacts do not comprise an entire application). WSPs are CAB files that contain a directory structure and definition files that are understood by the SharePoint deployment infrastructure. SharePoint imports the solution into the SharePoint store and then deploys it to the farm. In addition to the directory and definition files, a WSP primarily contains a set of features. A SharePoint feature is a logical unit of functionality and is self-contained.

SharePoint uses the definition files that are part of the WSP to determine artifacts should be deployed, where they should be deployed within the infrastructure, what configuration actions need to be taken, and what configuration information needs to be created.

For example, when a solution is deployed, SharePoint puts the files, such as pages, user controls, style sheets, and master pages, into the appropriate locations for each Web front-end server. It also deploys assemblies to the global assembly cache on each Web front-end server, installs SharePoint features, and makes configuration updates that indicate the availability of the newly deployed features and site definitions to SharePoint administrators. For more information about SharePoint site definitions and features, see [Using Site Definitions](#).

Solutions also can be retracted. In this case, all artifacts are removed from the file system and all assemblies are removed from the global assembly cache. However, the data that is related to the retracted solution, such as instanced data from an activated feature, remains in the content database even after the related solution is retracted. If the solution is reinstalled and the SharePoint features are reactivated, the site instance becomes available again. For more information, see [Deployment Scenarios](#).

Understanding how SharePoint solutions and features build a SharePoint application is important. You should take the time to understand them before you begin your first SharePoint application.

Site Definitions and Features

Defining an application template instead of an instance of an application itself is another important SharePoint

concept. When you define an application template, you are creating a standardized form that dictates how an instance of the application is provisioned. This differs from traditional ASP.NET development where you create a unique instance of an application that you expect to provision and operate in isolation.

The following illustration shows the concept of provisioning.

Provisioning capabilities



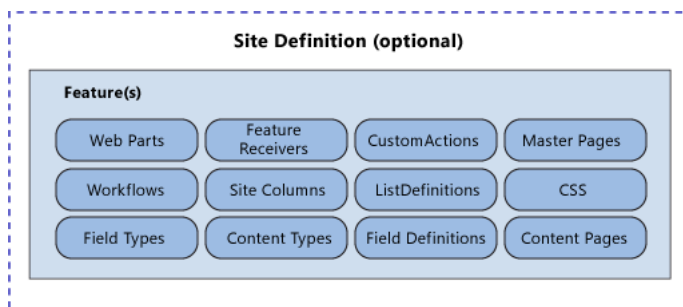
Note that the unit to provision can be smaller than an application. This is why it is represented as a capability instance in the diagram. In SharePoint, a feature is equivalent to a capability. This is shown in the following illustration.

Feature activation



A site definition is a template that contains the instructions for creating a site instance. Use a site definition as a collection of references to features that will be activated on the site when it is created. Use features to package related capabilities instead of defining the capabilities directly within the site definition. For example, if your application needs a list, include the list definition in a feature instead of putting it in the site definition. Features provide discrete modules of functionality that can be upgraded, in contrast to site definitions, which cannot be altered after they are deployed. The following illustration shows some of the types of artifacts that are found within a feature and the recommended relationship between features and site definitions.

Features and site definitions



A site definition only refers to a set of features. Features exist independently of any site definition. An effective way to think about a feature is as a blueprint for a capability, while a site definition is a blueprint for a site. Simpler applications may be defined by a single site definition and have a set of features that are activated on the site when it is created. More complex applications may have multiple cooperating sites and features that are activated outside the scope of a site.

Typically, a feature contains multiple elements that work together to implement it. For example, a feature can define a custom list and an associated Web Part. Features support the reuse of capabilities by different applications, and they make managing and versioning capabilities easier. Features can be activated with the site definition when a site is created and provisioned, or they can be activated later on a site that already exists. New instances of a site are provisioned either through SharePoint Central Administration or with a command line utility that is named STSADM.

Note:

Applications can include features that are not referenced in the site definition. This is because features can be scoped for different levels in the hierarchy. For example, there can be a feature that is scoped for a site collection instead of for a site.

The Training Management application demonstrates how to use features. When a feature is activated on a blank site, information is created in the content database that directs SharePoint to attach the set of lists and capabilities that are included in the Training Management feature to that site.

The Training Management application includes a site definition that contains the Training Site feature. This creates the site and activates the feature, which provisions the site. The Training Management application also has feature dependencies. There are site collection-scoped features that must be activated before activating the Training Management feature.

The following diagram shows what a Training Management site looks like before and after a feature is activated.

Training Management application feature



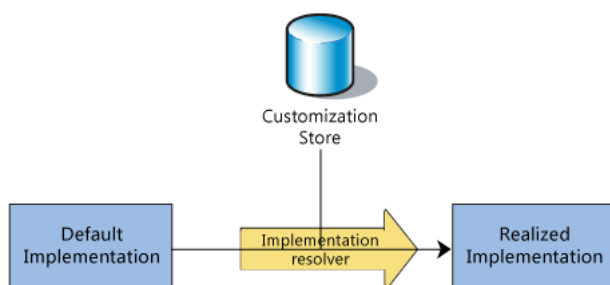
Activating a feature does not deploy any bits or files. Only deploying a solution deploys files to the file system. Activating a feature creates information in the SharePoint database stores, such as the configuration database and the content database, that lets SharePoint know that an instance of the feature is present.

Features should be carefully designed. Managing a SharePoint farm with illogically organized features can be chaotic. Conversely, properly designed features are very helpful to the IT professionals who administer the farm. For more information about features and templates, see [Features and Templates](#).

Post-Development Application Customization

SharePoint allows non-developers to construct an application by composing and customizing solution artifacts that developers provide. (Examples of artifacts that can be customized are master pages, styles, and page layouts.) The ability to customize solution artifacts that a developer has already defined will be unfamiliar to most ASP.NET developers. Developers need to understand what artifacts can be customized and how. Architects need to work with their IT counterparts to establish customization policies in order to control how applications evolve as well as what customizations will be preserved when an application is upgraded. The following illustration shows a generic model for how SharePoint manages customizations:

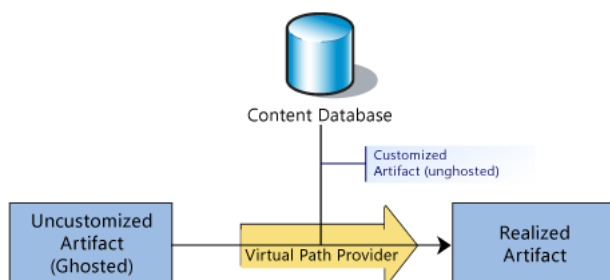
Managing customizations



In this model, the default implementation of an artifact is defined by the application developer. If the artifact is uncustomized, the implementation resolver sends it to the realized implementation. However, if the artifact is customized, the implementation resolver detects this, retrieves it from the customization store, and provides the customized version to the realized implementation at run time. In ASP.NET, this pattern uses a resolver that is named a [virtual path provider](#).

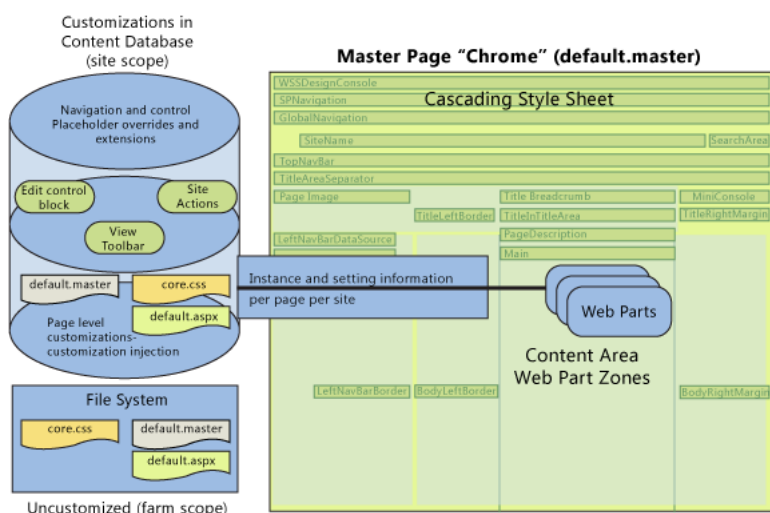
The SharePoint implementation of a virtual path provider determines if a customized version is in the content database. It uses the customized version if there is one. Otherwise, it uses the file-based version. The following illustration shows a SharePoint virtual path provider.

SharePoint virtual path provider



Users and IT professionals use the SharePoint Designer tool to create customizations and save them to a content database. Although SharePoint now uses the terms *uncustomized* and *customized*, you might also hear the older terms of *ghosted* and *unghosted*. For more information, see [Defining Artifact Terminology](#). The following illustration shows how SharePoint uses and extends ASP.NET capabilities to compose pages.

SharePoint page composition



This diagram is a good example of how SharePoint builds on and extends ASP.NET *capabilities*. On the right, the page is built from ASP.NET building blocks, such as master pages, Web Parts, Web Part zones, and cascading style sheets. Navigation providers and security providers also use the ASP.NET building blocks. SharePoint extends these capabilities with the ability to inject additional configuration-defined items, such as menus entries, master page customizations, and style customizations.

This section is a brief description of the many ways that non-developers can customize a SharePoint application. Developers must understand how their components can be customized. They should work with architects and IT professionals to put a customization policy in place. For more information about customization, see [Understanding and Creating Customized and Uncustomized Files in Windows SharePoint Services 3.0](#) on MSDN.

For a detailed discussion of the relationship between developed and customized artifacts, see [Defining Artifact Terminology](#).

A Quick Tour of the Partner Portal Application

The Partner Portal application demonstrates solutions to some of the most common situations that SharePoint developers encounter when they build enterprise-scale applications. It includes a collaborative extranet site that partners of Contoso can use to perform a variety of activities, including the following:

- They can view the Contoso catalog.
- They can view promotions and other news that is meant specifically for them.
- They can collaborate with Contoso employees to resolve problems.

The Partner Portal application is a reference implementation. It demonstrates the guidance that is discussed in the documentation and shows aspects of SharePoint development that cannot be captured in smaller code samples. It is intentionally incomplete. You can use it as the basis for your own solutions.

The following topic, [The Business Scenario](#), describes the business scenario that motivated the development of the Partner Portal application. It explains how the application uses SharePoint and the guidance to implement the solution. It also demonstrates how SharePoint is an effective platform for developing sites that are meant to facilitate collaboration, information sharing, and content management.

The Business Scenario

Contoso manufactures medical equipment. It sells its equipment through an extensive partner network. A team of account managers handles partner relationships. A team of customer service representatives provides support when there are problems. Contoso wants to build stronger ties with its partners by improving communications and by allowing partners to work efficiently with Contoso employees. It also wants to reduce the costs.

As part of this effort, Contoso decides to create an extranet site where partners can easily access information and also collaborate with Contoso employees to resolve technical and sales-related issues. This site is named the Partner Portal.

Contoso would like the Partner Portal to fulfill the following objectives:

- Currently, partners must speak to an account manager to get specific pricing information. Pricing calculations are complex and based on individual contracts that each partner has negotiated with Contoso. Partners use a printed catalog to find components and products, which is often out-of-date. The Partner Portal will have an online catalog. Each partner will see a catalog with pricing information that reflects the terms of that partner's contract.
- Contoso wants to market to their partners more effectively. They will use the Partner Portal to target subsets of partners, and to offer different promotional prices to specific partners. Promotions must be approved before they appear on the Partner Portal and they must be created in a secure environment that is inside the Contoso corporate firewall. Because their medical equipment is complex, Contoso wants to take advantage of features such as multimedia presentations to showcase and explain their products.
- Currently, there is no central location where partners and customer service representatives can work together to resolve problems. The most common issues that Contoso encounters are tier three support incidents. These are issues that require close collaboration between a partner and a Contoso representative. An example is when a customer needs help to diagnose an equipment malfunction. Another common problem that requires collaboration is an order exception such as a manufacturing delay or a logistical issue. Managing these issues is labor-intensive and there is no standard process in place. Contoso wants the Partner Portal to automatically create collaboration spaces for incidents that reach tier three. These spaces will integrate information from both the incident management and the order management systems. They will also host discussion threads, keep track of tasks, and store and version documents.

Partner Stories

The following stories describe the types of activities that partners can perform with the Partner Portal application.

Note that most, but not all, of the activities are implemented in the application. Areas of the application that are implemented show a particular technique or concept. Areas that do not contribute to an understanding of SharePoint have been left unimplemented. However, the guidance should be sufficient for you to implement them if you want. For example, an actual extranet site would probably show a specification sheet from the catalog to a partner. Because this does not contribute additional guidance about developing a SharePoint application, it is unimplemented.

Partners can use the Partner Portal to perform the following activities:

- View information from Contoso such as news and promotions.
- Navigate the product categories and view products by category.
- View product information.
- View product details and related information such as specification sheets and pricing. The pricing includes discounts that reflect contractual agreements. (This is only partially implemented.)
- See special offers and their details.
- Collaborate with Contoso on high-priority incidents.
- See a summary of the incidents that are currently open and the incidents that are closed.
- See an incident's current status, manage outstanding tasks, and exchange information about the incident with Contoso.
- View historical information about closed incidents.
- See a summary of the order exceptions, such as those that are currently open or closed. (This is only partially implemented.)
- Collaborate with Contoso to resolve order exceptions.
- See an order exception's current status, manage outstanding tasks, and exchange information with Contoso.

Contoso Stories

Contoso employees can use the Partner Portal to perform the following activities.

- Account managers can use the partner collaboration portal home page to edit content that is intended for specific partners.
- Account managers can view outstanding tasks for all their partners to determine what needs to be done to resolve problems.
- Account managers can navigate to a partner's portal from a central location.
- Employees can use images and video to create promotions that are intended for a specific set of partners.

Pricing can be different for different partners.

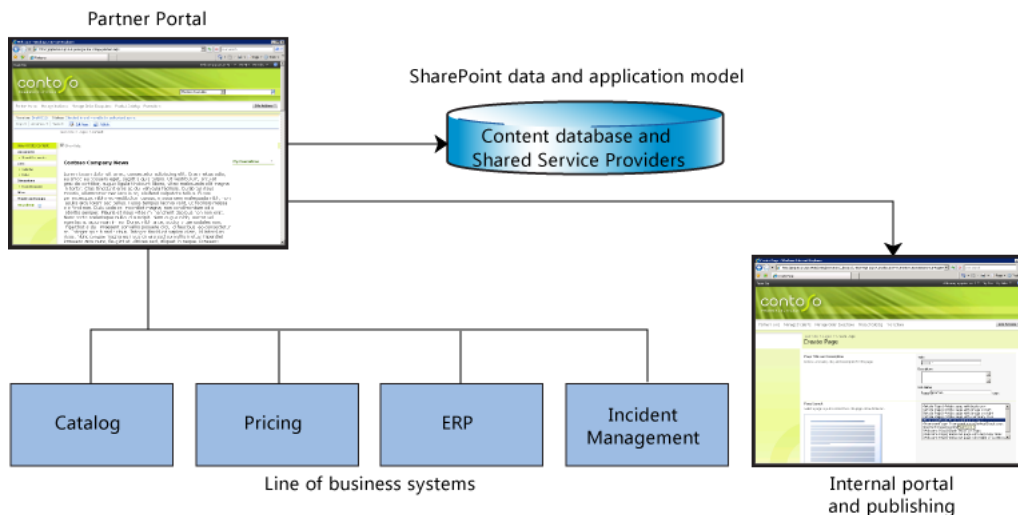
- Sales managers can review and approve promotions before they are released. (This is not implemented.)
- Customer service representatives can use the standard incident management system client to create a collaboration space. (In the Partner Portal application, this is implemented as a stand-in ASPX page.)
- Customer service representatives and service engineers can collaborate with a partner to exchange information and coordinate tasks in order to resolve open incidents.
- Customer service representatives can close incidents in the incident management system. The partner can see historical information about the incident.
- The ERP/logistics system automatically creates order exception collaboration sites when the order is delayed for more than 48 hours for when the order is worth over \$25,000. (This is not implemented.)

The Partner Portal Application

This topic describes the high-level architecture and design of the Partner Portal application and discusses how different aspects of the guidance and various SharePoint capabilities implement the stories that are described in [The Business Scenario](#). For detailed information about the Partner Portal application, see the [Partner Portal Reference Implementation](#).

The following illustration shows the high-level elements of the Partner Portal application, the content database, and the shared services providers (SSPs).

Elements of the Partner Portal application



The Partner Portal site is the entry point that all partners use to access the application. The Partner Portal presents an integrated view of the application's different capabilities and provides all partners with their own site collection. The Contoso IT department provisions a partner's site collection. For detailed information about this process, see [New Partner Provisioning](#).

The Partner Portal site provides a dedicated collaboration space for interacting with Contoso, access to the product catalog, promotional offers, and other line-of-business information. Subsites are automatically created to collaborate on individual incidents. The Partner Portal integrates with multiple line-of-business (LOB) systems to provide business information to partners. This has two benefits. One benefit is that partners can access relevant information themselves, without requiring help from a Contoso employee. The other benefit is that formal LOB processes are introduced into informal collaboration processes.

The Partner Central internal portal site represents the entry point that Contoso employees use to process information from partners and to access a specific partner's collaboration space. Additionally, Contoso employees use the site's authoring capabilities to create promotional offers that are reviewed, approved, and then deployed to the production farm. (The reviewing and approval process is not implemented.)

The Partner Portal application is a combination of standard SharePoint functionality and custom application logic. The following sections describe the Partner Portal application's main features of the reference implementation, reference the relevant sections of the guidance, and show how SharePoint's capabilities were used or extended.

Partner Portal and Incident Subsites

SharePoint automatically creates an incident subsite when a partner has a problem whose severity is termed "tier three." Partners access their subsites through the Partner Portal site.

SharePoint uses both site definitions and features to produce repeatable instances of a site or a capability. For more information, see [An Overview of the SharePoint Platform for ASP.NET Developers](#). The Partner Portal application uses site definitions and features in the following ways:

- **Creating a new partner extranet site.** Contoso frequently adds new partners. SharePoint provides an infrastructure that automatically provisions an instance of a new site when a company becomes a Contoso partner. This process requires no development effort and no need to make any physical changes to the operating environment. To add a partner, administrators create a new site collection, add it to the site directory, apply the appropriate features, and configure the site's security permissions. For more information, see [New Partner Provisioning](#).
- **Adding subsites for incident and order exception management.** Collaboration subsites allow partners and Contoso employees to resolve order exceptions and incidents. The type of subsite and the information that is retrieved from the LOB systems varies, depending on the type of incident. Templates allow instances of each subsite type to be created automatically when a particular business event occurs. The guidance shows how application patterns describe a general solution to the challenge of automatically creating subsites. For more information, see [Application Patterns](#). The guidance also includes a library of reusable components that make the implementation of the patterns simpler. For more information, see [Workflow-Driven Site Creation](#).

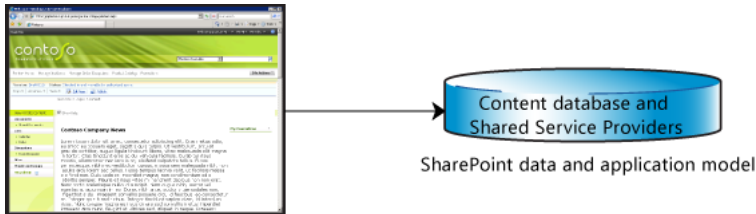
In addition to its site provisioning capabilities, the Partner Portal site and subsites take advantage of many of the features that SharePoint provides for building scalable, enterprise-quality applications. For more information, see

[Building for Scale.](#)

Data Isolation and Security

The following illustration shows the relationship between data on a partner's site, the content database, and the SSPs.

Data isolation and security



SharePoint is designed to simultaneously support multiple applications and users on a single infrastructure. To provide a secure environment for each application and user, SharePoint has different levels of data and security isolation. The Partner Portal application demonstrates some advanced techniques for implementing them. The following are some of the highlights:

- **Site collection security and data isolation boundary.** The Partner Portal application uses separate site collections for each partner's collaboration space. Roles that are specific to the partner are assigned to this collaboration site collection. Within the database, data for a site collection is isolated from other site collections. As a result, there is no possibility of data from one site collection appearing on another site collection. For more information, see [Considerations for Extranet Development](#).

Note that although the Partner Portal application uses the site collection as the boundary between partners, SharePoint includes resources and configuration capabilities that provide even greater separation. For example, all partners can have their own Web applications. This provides a process boundary and allows you to assign a dedicated database to each partner. The increased security must be weighed against the increased operational complexity and additional hardware costs, but this may be a feasible solution in high-security environments. For more information, see "Subsite scripting" in [Plan site security](#) and [Logical architecture components](#) on TechNet.

- **Security zones.** Although security zones can be a complex topic, the basic concept is straightforward. Users often belong to different security groups that are authenticated in different ways with different credential stores. In the case of the Partner Portal application, Contoso employees are authenticated within the corporate network with standard Windows domain security mechanisms.

Contoso does not want their partners' credentials to be included in the Active Directory domain. Generally, organizations want to use a dedicated credential store for external users and to authenticate the users within a perimeter network (also known as DMZ, demilitarized zone, and screened subnet). SharePoint provides zones to address this situation. Zones allow different groups of users with different security permissions to access the same application. Each group can be authenticated with a different security mechanism and assigned different rights. A commonly used authentication mechanism is forms-based authentication. For more information, see [Considerations for Extranet Development](#).

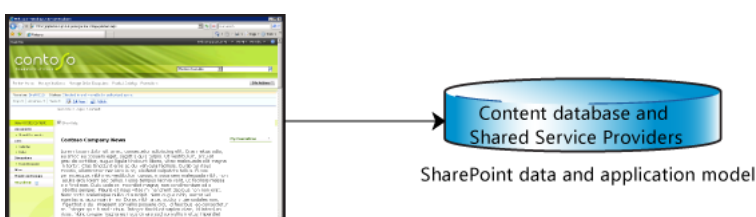
- **Access control lists.** In the Partner Portal application, Contoso employees create and approve partner promotions. (The approval process is not implemented.) It is possible for the same promotions to apply to several partners. In situations where information is centralized and is accessed by multiple users, SharePoint uses an access control list to determine who can view and modify information. This list assigns permissions to items in a SharePoint list or library. The Partner Portal application uses a central site collection for all published promotions, and then it uses the access control list to assign rights to each promotion. Partners can only access the promotions that apply to them. SharePoint has standard features that support this security model, such as the ability to assign permissions to users in different security zones to view the same content.

There are several other ways to control access to information. One alternative is to publish the promotions to the individual partner site collection. This approach is more complex than the one that is used by the Partner Portal application and requires custom deployment logic for the promotions. Another approach is to divide the partners into a few groups, such as a Gold group, Silver group, and Bronze group, and have a publishing site for each group. This simplifies the access control because partners are assigned to a site and not to an item, but the flexibility of the application is reduced. For more information, see [Understanding Publishing and Content Deployment](#).

Collaboration

The following illustration shows the relationship between the collaboration site, the content database and the SSPs.

Collaborating in SharePoint



Contoso account managers must be able to publish information on the partners' sites, and partners must be able to exchange information with Contoso. The Partner Portal application takes advantage of SharePoint's standard collaboration features to enable these scenarios in the following ways:

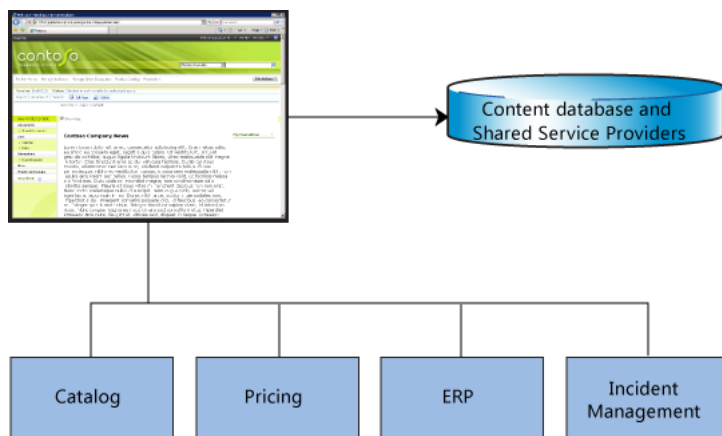
- **Creating collaboration spaces.** The Partner Portal application uses standard SharePoint collaboration tools, such as document libraries and task lists. As a result, the collaboration spaces provide more benefits and better communications than e-mail or telephone calls. Documents are tracked and versioned, tasks are recorded and their status monitored, and pertinent information is posted in a timely manner.
- **Extending collaboration.** SharePoint makes it easy to add additional collaboration features, such as blogs and wikis, by changing the partner site's configuration. In addition, each partner site can be branded and customized. This is not shown in the Partner Portal application.
- **Aggregating promotional information.** The Partner Portal application uses Web Parts to query for promotional information on the central publishing site. The information is then displayed on the partner's home page.
- **Extending navigation.** The Partner Portal application extends the standard SharePoint navigational features to provide cross-site navigation. This allows the partner to navigate between the Partner Portal site, the promotional information site, and the catalog site.

For more information, see *Considerations for Content-Driven Applications*. This topic also provides guidance on other issues such as custom navigation, site branding, content deployment, and the central template gallery.

Accessing LOB Information

The following illustration shows the relationship between the Partner Portal, the LOB systems, the content database and the SSPs.

Accessing LOB information



The Partner Portal application displays information from the LOB systems on each partner's site. This means that the partners do not need to rely on Contoso employees to access this information. The SharePoint infrastructure includes security mechanisms that protect each partner's privacy. It also provides the functionality for integrating information from LOB systems into SharePoint applications. The Partner Portal application takes advantage of these capabilities in the following ways:

- **Promoting self-service to improve efficiency and lower cost.** The Partner Portal provides a catalog that partners use to see product information, which is in the LOB systems. The LOB systems expose this data through Web services. The Business Data Catalog (BDC), which is provided by SharePoint, determines how SharePoint accesses these services and provides the Web parts that bind to and display this data.
- Another way to display LOB information is with data field types, which allow you to include the information in SharePoint lists. This technique is not implemented in the Partner Portal application.
- **Searching LOB data.** Partners can search for product information. The Partner Portal application uses Web services and the BDC to crawl the catalog and create a search index. The detailed product information is gathered from the LOB systems and includes sensitive data such as pricing.
 - **Protecting confidential information.** The Partner Portal application displays partner-specific pricing information and incident information on each partner's site. This requires that the application securely transmit the partner ID to the appropriate LOB system to extract the correct information. The application combines SharePoint security capabilities with Windows Communication Foundation (WCF) capabilities to accomplish this.

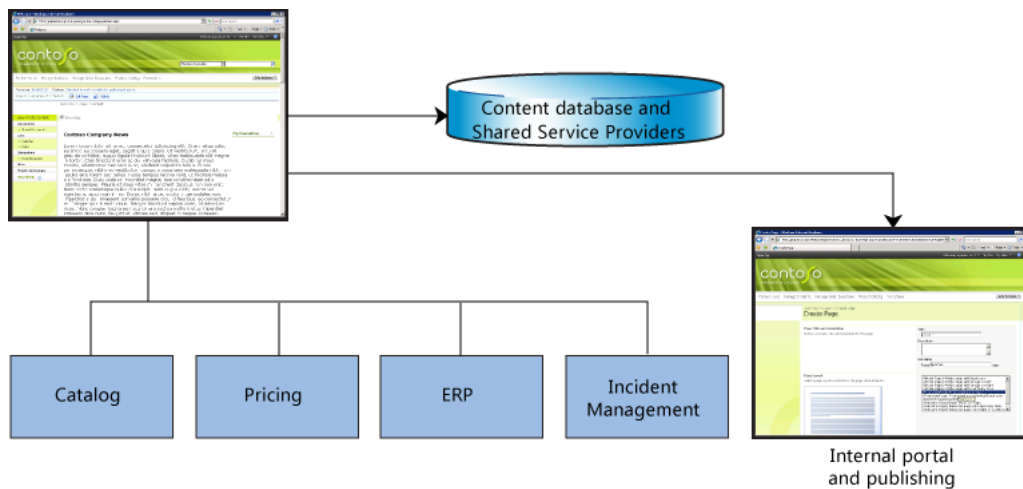
SharePoint also supports *security trimming*. Security trimming restricts access to pages or information to particular users. It can be applied to search results, such as the pricing information that the BDC gathers. This technique is not implemented in the Partner Portal application.

For more information about how to access LOB systems from a SharePoint application, see [Integrating Line-of-Business Systems](#). This topic includes guidance on how to use the BDC and Windows Communication Framework to integrate line-of-business systems that are exposed through Web services into a SharePoint application. This chapter also provides guidance on creating Web services within a SharePoint solution and on security considerations.

Managing and Approving Content

The following illustration shows the relationship between the Partner Portal, the LOB systems, the internal publishing site, the content database, and the SSPs.

The Partner Portal application relationships



Contoso managers, business analysts, and other non-technical employees create new promotions and associate them with product data and one or more partners. Promotions include product and pricing information that is stored in the LOB systems. After they are complete, the promotions go through an approval process, which is controlled by a workflow. Finally, they are deployed from the corporate authoring site to the production Partner Portal site. (The approval process is not implemented.)

The Partner Portal application demonstrates the following SharePoint capabilities:

- **Creating content.** SharePoint allows non-technical information workers to create content that is based on a predefined layout. They can either use a browser or a Word document.
- **Approving content.** SharePoint provides an approval workflow, which routes a document or item to a group of people for approval. The Partner Portal application routes promotions through this workflow before they are published.
- **Assigning permissions to content.** SharePoint allows you to assign access permissions to content. The Partner Portal application uses this capability to determine which partners can see which promotions.
- **Deploying content.** SharePoint allows you to author content in a site that is within the corporate network and deploy it to the perimeter network. The Partner Portal uses this capability to manage how promotions are deployed to the central publishing site.
- **Adding LOB information to published content.** The Partner Portal application uses Web Part connections to display LOB information on promotions. A filter Web Part extracts information that is entered in the publishing page by the author of the promotion. This information is the product identifier, which is the stock-keeping unit (SKU). The filter Web Part transfers the information to another Web Part that displays the product information. The product information Web Part retrieves product information from the LOB system and displays it.
- For more information, see [Considerations for Content-Driven Applications](#). This topic also provides guidance on custom navigation, site branding, content deployment, and the central template gallery.

Branding SharePoint Sites

Contoso wants all the partner sites to be consistently branded. It also wants the ability to change branding from a single location and have that change reflected across all the partner sites. To accomplish this, the Partner Portal application uses SharePoint themes and delegate controls to provide consistent branding and navigation to all partners.

SharePoint also includes capabilities to create custom branding. For example, you can include a partner logo on a site or site collection in different application instances. This is not implemented in the Partner Portal application but would be a desirable feature in a fully realized application.

For guidance on content and layout customization, site branding, content deployment and the use of the central template gallery, see [Considerations for Content-Driven Applications](#).

Viewing and Aggregating Information

The Partner Portal can use SharePoint's search capabilities to aggregate information in different ways and display it in multiple views. The Partner Portal application uses these views in the following ways:

- **To aggregate data across many sites.** The Partner Portal application uses the search capability programmatically to aggregate large quantities of information that are distributed across all the partner collaboration sites. For example, Contoso employees can see a consolidated view of all the outstanding tasks for all partners for open incidents.
- **To aggregate information within a site hierarchy.** The Partner Portal application aggregates information within a site hierarchy to provide consolidated views to partners and account managers. SharePoint can

perform searches and display information in real time. These capabilities can be expanded by business analysts and developers. The Partner Portal application aggregates information about incident status and outstanding tasks for each partner and displays it on that partner's collaboration site.

- **To view information from related sites.** You can use the SharePoint object model or Web services to query other sites for information. SharePoint ensures that authorized users view the information. SharePoint only returns information that users are authorized to see when that information is accessed through the object model for data that is contained within SharePoint. The Partner Portal application aggregates promotional information for all partners from the central publishing site. However, partners only see the specific promotions that are intended for them.
- For more information, see [Techniques for Aggregating List and Site Information](#).

Design and Development Guidelines

This section includes the following topics:

- [SharePoint Design Basics](#). This topic discusses fundamental challenges that architects and developers face when they create a SharePoint application.
- [SharePoint Implementation Basics](#). This topic describes how to perform some of the basic implementation tasks for creating a SharePoint application.
- [Managing the Application Life Cycle](#). This topic discusses ways to deploy and upgrade SharePoint applications. The guidance uses a topology that is recommended for small- and medium-sized companies.

SharePoint Design Basics

This topic includes guidance about some of the fundamental challenges that architects and developers face when they create a SharePoint application. The examples that provide guidance on how to address these challenges come from the Training Management reference implementation. This application demonstrates how to develop a basic SharePoint application. For a detailed discussion about the Training Management application, see [Training Management Reference Implementation](#).

The following subtopics describe the design decisions that were made for the Training Management application:

- [Using Site Definitions](#). This topic explains the reasons for using a site definition to package the Training Management application.
- [Content Types](#). This topic is a general discussion of content types and their relationships to list.
- [Using SharePoint Lists vs. Database Tables](#). This topic explains the reasons why lists, instead of databases, are used for data storage in the Training Management application.
- [Using List Definitions](#). This topic explains the reasons for using list definitions in the Training Management application.
- [Extending SharePoint List UIs](#). This topic explains the options that were evaluated when extending the SharePoint list user interfaces.
- [Enforcing Custom List Item Data Validation](#). This topic explains the reasons for using custom event receivers to perform data validation.
- [Using Standard and Custom Web Parts](#). This topic explains when to use standard SharePoint Web Parts and when to use custom Web Parts.
- [Using Workflows and Event Receivers](#). This topic explains when to use workflows and when to use event receivers.
- [Using SharePoint Permissions](#). This topic explains how the Training Management application uses custom groups and permission levels to implement security.
- [Organizing Features](#). This topic explains how to organize SharePoint features in a solution. It uses the Training Management application as an example.

Using Site Definitions

A site definition defines a unique type of SharePoint site. It is a collection of files that are located in the Office 12 hive folder on a Web front-end server. Typically, the files are located in the directory \\program files\\Common Files\\Microsoft Shared\\web server extensions\\12\\TEMPLATE. In Windows SharePoint Services 2.0, site definitions were the primary methods for defining lists, provisioning files, and performing other customizations to a SharePoint site.

In Windows SharePoint Services 3.0, the introduction of SharePoint features provides an alternative to site definitions. Updating sites based on custom site definitions can be difficult. Features provide a more flexible and modular way of customizing sites. For example, a drawback to using site definitions instead of SharePoint features is that any changes you make in a site definition file may not affect existing sites.

Site definitions do have benefits. A site definition is a straightforward way to create a site in SharePoint. It is also a convenient way to include all the functionality of a custom site in one place. For example, there is a great deal of management overhead for sites that require many SharePoint features to be manually activated. It is possible to write custom SharePoint feature activation code that automatically activates related SharePoint features, but the custom code can add to the development costs. Instead, you can include the SharePoint features in a site definition. When a site is created based on the custom site definition, all the SharePoint features are automatically activated.

Another scenario that benefits from site definitions is when there will be many instances of a particular type of site. By using site definitions, you can repeatedly create those instances with consistent results.

Another reason for site definitions is *feature stapling*. Developers can use feature stapling to add functionality to site definitions to upgrade the site. For example, one way to apply a custom theme to a site is for a developer to write a custom SharePoint feature that applies the theme. An administrator deploys this SharePoint feature and then manually activates it whenever a site is created. However, with feature stapling, if a SharePoint feature is stapled to a site definition, the new SharePoint feature is automatically activated when a new site is created. For more information, see [Stapling Features to Site Definitions](#).

The Training Management application uses a collection of SharePoint features to provide the application's functionality, but it wraps the collection in a simple custom site definition. All the functionality to run the Training Management site is encapsulated in the SharePoint features. The site definition merely activates the SharePoint features.

Implementing the Site Definition

The site definition determines the basic look of the SharePoint site and also contains much of the site's functionality. Site definitions can include items such as content types, lists, navigation bars, and workflows. The site definition is largely determined by the Onet.xml file. The site definition for the Training Management application defines only the features to be activated.

The Onet.xml File

The Training Management custom site definition development begins with a copy of the Blank configuration of the SharePoint Team Services (STS) site definition (STS#1). If you are using Visual Studio extensions for Windows SharePoint Services, you can use the Blank Site Definition project template. The following procedure demonstrates how to manually modify the Onet.xml file so that it duplicates the one that is used by the Training Management application. It is important to note that the Training Management site was developed with Visual Studio extensions for Windows SharePoint Services.

When using Visual Studio extensions for Windows SharePoint Services, features that are part of the project that contains the site definition are automatically added to the Onet.xml file during deployment. Therefore, it is unnecessary for developers to include the SharePoint feature IDs in the Onet.xml file. However, you must manually add **Feature ID** attributes to the **SiteFeatures** element.

To change the Onet.xml file

1. Provide a new project title. This is shown in the following XML.

XML

```
<Project Title="Training Management" Revision="1" ListDir="$Resources:core,lists_Folder;"
xmlns:ows="Microsoft SharePoint">
```

2. Add the Contoso Training Management site-collection-scoped SharePoint feature. This is shown in the following XML.

XML

```
<SiteFeatures>
...
<!-- ContosoTrainingManageSiteCollection Feature -->
<Feature ID="0cb4445d-ab1a-4053-9301-50c3b2666f31" />
</SiteFeatures>
```

3. Add the Contoso Training Management Web-scoped SharePoint features. This is shown in the following XML.

Note:

If you use Visual Studio extensions for Windows SharePoint Services, the XML samples that are included in this step are only seen in the final Onet.xml file after the solution is deployed. You do not need to manually add the SharePoint feature IDs.

XML

```
<WebFeatures>
...
<Feature ID="00BFEA71-4EA5-48D4-A4AD-7EA5C011ABE5"/>
  <!-- TeamCollab Feature -->
  <Feature ID="F41CC668-37E5-4743-B4A8-74D1DB3FD8A4"/>
  <!-- MobilityRedirect -->
  <!-- VSeWSS will automatically feature id's for ContosoTrainingManagementWeb and
auto generated hidden features here! -->
</WebFeatures>
```

Note:

Customizations to site definitions that are installed with any of the SharePoint default site definitions such as STS, MPS, WIKI, and BLOG should not be modified. Modifications to these site definitions are lost when upgrades or service packs are applied to the SharePoint installation.

For more information, see [Onet.xml](#) on MSDN.

The webTempContosoTrainingManagement.xml File

The webTemp.xml file defines custom site definition configurations that are used when instantiating sites. Site definition configurations are also known as site templates. The Training Management application's site template has an accompanying webTemp.xml file. The following XML shows the webTemp.xml file for the Training Management site definition.

XML

```
<Templates>
  <Template Name="CONTOSOTRAINING" ID="10001">
    <Configuration ID="0" Title="Contoso Training Management Site" Description="Contoso
Training Management Site" Hidden="FALSE" ImageUrl="/_layouts/images/stsprev.png"
DisplayCategory="Contoso" />
  </Template>
</Templates>
```

The **Name** attribute matches the folder that contains the Onet.xml file in the SiteTemplates folder in the Office 12 hive. The **ID** attribute is any unique number. The **Configuration** element determines which configuration to use in the Onet.xml file. The Training Management site definition contains only one configuration. The **Title** and **Description** attributes define what the user sees when creating the site with the SharePoint browser. For more information, see [WebTemp.xml](#) on MSDN.

Content Types

A content type is a reusable collection of metadata, information policies, form URLs, document templates, event receivers, and workflows that is applied to a particular category of information. All content that uses a particular content type contains the same metadata fields and behaviors. For example, the Training Management application has a content type that applies to all training courses. The training course content type includes information such as the start date of the course, the course code, and its cost.

Metadata is literally "data about data" or, in the case of a content type, information about the document or item that is being described. A content type's metadata can include the columns in a list. If you look at the Training Management application's Web pages, the list of training courses includes columns that represent the data that is contained in the content type. For example, there is a Cost column and a Code column.

Content types help you to organize SharePoint content in meaningful ways and enable you to centralize the metadata and behaviors that apply to different types of SharePoint content. By using content types, you can also store different types of information in a single list or document library, while keeping the functionality specific to each type of information. The following is some of the information that can be stored in a content type:

- Metadata or properties of the information that is being captured.
- Custom New, Edit, and Display forms to be used with this content type.
- Workflows that are associated with items of this content type.
- Document templates for document-related content types.
- Any information that is required for custom solutions that are associated with this content type. An example is a list item event receiver. You can store this information in the content type as XML documents.

They can be used in multiple instances of lists or libraries in the same site. Content types are centrally available from the site collection because the SharePoint features that deploy them are site-collection scoped. For more information, see [Content Types](#) on MSDN.

Using SharePoint Lists vs. Database Tables

An important decision in the design of the Training Management application was deciding whether to store information in lists or in a database. SharePoint lists consist of rows and columns that store data in a similar fashion to a traditional relational database management system such as SQL Server. However, a benefit of lists is that SharePoint includes Web Parts that provide simple methods for managing the data. If the data was stored in a database, it would require custom user interface components to access it and manipulate it. Also, specialized skills are required to design, implement, and maintain a custom database. Another advantage of using lists is that custom workflow and event handlers can easily be registered to them.

There are also advantages to storing data inside of a database. One is the availability of all the ACID (Atomic, Consistent, Isolated and Durable) properties of transactions. If your business logic requires transactions, storing data in a database is preferable to using lists. Also, SharePoint lists are meant to store simple data structures. If you require a complex data model with intricate relationships, a database is more appropriate. The Training Management application has three data storage requirements. It stores data related to training courses, registrations, and registration approval tasks. All the data is relatively simple and does not use transactions. The registration data also requires that there be a workflow. All these reasons make SharePoint lists the appropriate choice.

In general, reading and writing to a custom database provides an overall performance advantage. Although SharePoint has made significant strides in its performance, there is still a certain amount of overhead involved in processing lists. The SharePoint product team recommends limiting the number of items in a list to 2,000 for each list container. (A container is the root of the list and any folders in the list.) You may be able to manage larger lists by using views that are filtered so that no single view contains more than 2,000 items. Another way to circumvent this performance degradation is to write custom interfaces to manage and retrieve the data. (It is important to note that the 2,000 item threshold has more to do with HTML rendering performance than with the underlying performance of lists as a storage mechanism.) For more information about the performance of SharePoint lists, see the SharePoint product group's white paper, [Scaling to Extremely Large Lists and Performance Access Methods](#).

The following table summarizes the benefits of using databases and SharePoint lists.

Benefits	Database	SharePoint list
Handles complex data relationships	Yes	No
Handles large numbers of items	Yes	No
Handles transactions	Yes	No
Is easy to use	No	Yes
Accommodates workflows	No	Yes
Includes a standard interface	No	Yes
Can easily add binary data	No	Yes

For examples of using SharePoint lists, see [The Training Course List](#), [Adding an Item to the Registration List](#), and [Updating a Registration List Item](#).

Using List Definitions

A list definition describes the schema of SharePoint lists. Typically, list definitions are used to add functionality to lists and to ensure that multiple instances of that list are consistent. In general, when you need custom activities and validations to be performed on a list, use a list definition. For more information about list definitions, see [List Definition](#) on MSDN.

Extending SharePoint List UIs

The registration action performs various functions, such as validation and the creation of a **SPListItem** instance in the registration list. One requirement of the Training Management application is that users should not directly create a new item in the registration list. This would allow them to bypass the validation and approval processes. In addition, if users manually create their own registration list items, they must know the internal **SPUser** ID and the training course ID. A user-friendly application should hide such details.

Instead, the application presents users with a list of courses. Users can select a course from the list and register. The following are three possible approaches to implementing the registration action:

- Use a custom field control.
- Use a custom DispForm.aspx page.
- Use a custom action.

The following sections discuss these approaches. For more information about the registration action, see [Register for a Course Use Case](#).

Using a Custom Field Control

One approach is to use a custom field control. Custom field controls are server controls that are used on the list view and display form. The Training Management application would use the control to display a button to initiate a registration procedure. One drawback to this approach is that custom field controls are meant to expose data in a custom field type. However, the Training Management application does not need any custom field types. Custom field controls also have a high development cost. For more information about custom field types and custom field controls, see [Custom Field Types](#) on MSDN.

Using a Custom DispForm.aspx

Another approach is to customize the DispForm.aspx page by placing a button on it that initiates the registration procedure. Writing custom .aspx pages is a common practice in SharePoint development. Initially, this approach seems to be the least expensive one. However, it requires custom code-behind. The DispForm.aspx page is a standard SharePoint page used by lists, and it already has default code-behind. Adding custom code-behind can cause unexpected behavior. Also, including the **Register** button on the DispForm.aspx page allows users to register only through the list view. It does not provide them with a way of seeing the list of courses and of initiating the registration process.

Using a Custom Action

A SharePoint custom action represents a link, toolbar button, menu item, or any control that can be added to a toolbar or menu that a user sees. Custom actions can be bound to a list type, a content type, a file type, or a programmatic identifier (ProgID). The custom action can invoke JavaScript, redirect the user to an URL, or perform custom code-behind procedures. The Training Management application places a custom action on the shortcut menu of the list view and on the DispForm.aspx toolbar. It redirects users to a custom .aspx page where they confirm that they would like to register for a course and they complete the registration. By presenting users with a confirmation page, the registration function can occur during postback. This is preferable to updating lists during an HTTP_GET operation, which requires that the **AllowUnsafeUpdates** property be set to **true** and may be an unsafe practice. In addition, the development cost of a custom action is minimal. For more information about custom actions, see [Custom Action Definitions](#) on MSDN.

Enforcing Custom List Item Data Validation

Two approaches were considered for validating the Training Course SharePoint list items. One approach was to edit the standard NewForm.aspx and EditForm.aspx pages to add custom validation controls. However, editing standard SharePoint files is not an ideal solution because you lose some of the standard functionality that SharePoint provides, such as the standard appearance and behavior (look and feel) and built-in client side validations. Also, users can circumvent the validation by using the SharePoint Web services or Web-based Distributed Authoring and Versioning (WebDAV).

Another approach is to add custom list item event receivers to the list that holds the items that you want to validate. This approach enables you to use custom business logic that either allows or cancels the addition or editing of a list item. Also, as part of the cancellation process, you can specify custom error messages that appear in the standard error page.

Because standard list item fields provide only basic data types and essential field validations, adding custom validation through list item event receivers also allows you to include functions such as performing comparisons with other fields, validating formats, and validating an item's uniqueness. One drawback to this approach is that list item event receivers can be implemented only as server-side code. This approach is not appropriate when client-side validation is required.

For an example of custom validation, see [The List Item Event Receivers](#).

Using Standard and Custom Web Parts

Standard Web Parts are provided by SharePoint and developers can write custom Web Parts. Custom Web Parts can serve as wrappers for custom ASCX user controls. For an introduction to Web Parts, see [ASP.NET Web Parts Controls](#) on MSDN.

Using Standard SharePoint Web Parts

Some of the most widely used standard Web Parts includes the List View, Library View, User Tasks, and Page Viewer. The List View Web Part is included in both the manager and training dashboards. To learn how the manager dashboard is implemented, see [View the Manager Dashboard Use Case](#). To learn how the training dashboard is implemented, see [View the Training Dashboard Use Case](#). For more information about SharePoint Web Parts, see [Web Parts in Windows SharePoint Services](#) on MSDN.

Using Custom Web Parts

Use custom Web Parts when no standard SharePoint Web Part is suitable. In many cases, a custom Web Part is necessary because the Web Part relies on an external data source or a specialized user interface is required to display the data. For example, custom Web Parts display the manager's direct reports and accounting information on the manager dashboard.

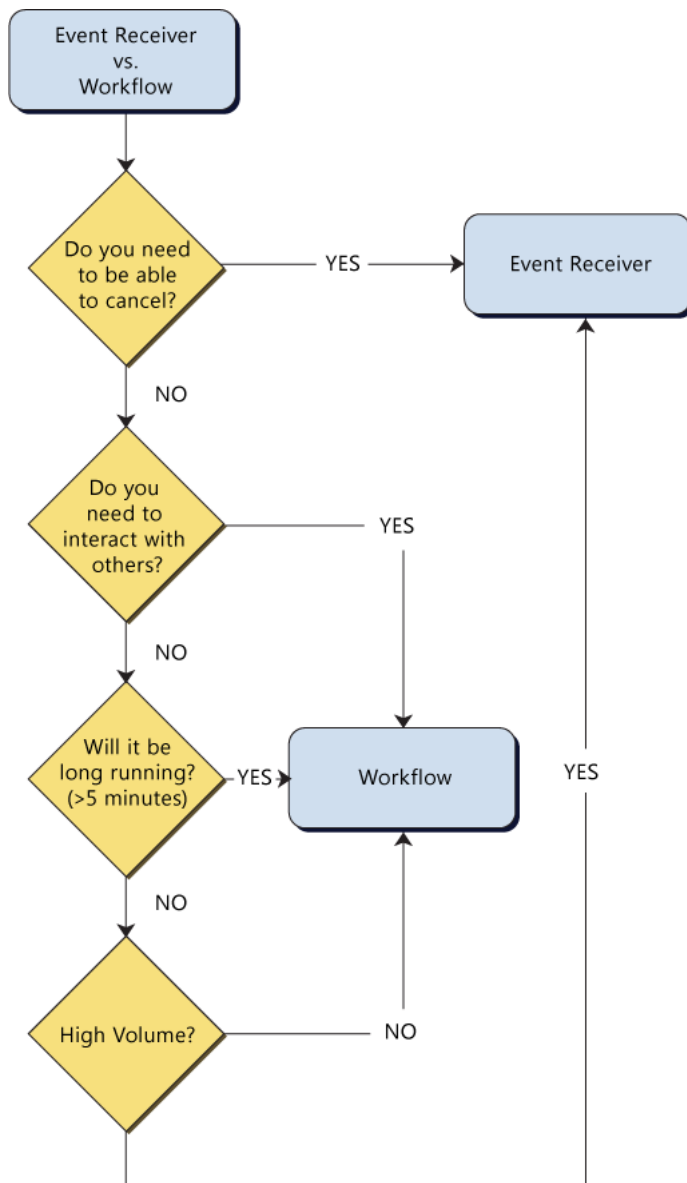
A custom Web Part can be used as a wrapper to display a custom user control. An advantage to user controls is that you can use the Visual Studio design view to develop them. This option is not available when developing Web Parts server controls. If you decide to use a custom user control, you must deploy the ASCX file and the Web Part. The manager dashboard uses a custom user control with a Web Part wrapper to display the manager's direct reports. To learn how to wrap a user control in a Web Part, see [How to: Wrap a User Control Inside of a Web Part for SharePoint](#).

A custom server control is more appropriate than a user control if the functionality it provides is highly reusable and if the control does not need to be developed using the Visual Studio design view.

Using Workflows and Event Receivers

The Training Management application uses the Windows Workflow Foundation to process a registration from the employee's initial request until the manager's approval and the accounting update. In general, SharePoint applications can use either workflows or event receivers to implement business processes. The following figure illustrates a flow chart that can help you decide if you should use an event receiver or a workflow.

Workflows and event receivers



The initial decision point is if you need the ability to cancel the process. If you do, you should use an event receiver. If you do not need this ability, you must decide whether the business process requires input from users. For example, the Training Management application requires a manager to approve or reject a pending registration, so it uses a workflow. Another issue is persistence. If the process takes longer than five minutes to resolve, that is another case where a workflow is more appropriate than an event receiver. For example, in the Training Management application, a manager may leave registration requests as pending for an unspecified amount of time. Finally, if the application is heavily used, you may need to consider using an event receiver instead of a workflow. For more information about Windows Workflow Foundation, see [Windows Workflow Foundation \(WF\)](#) on MSDN.

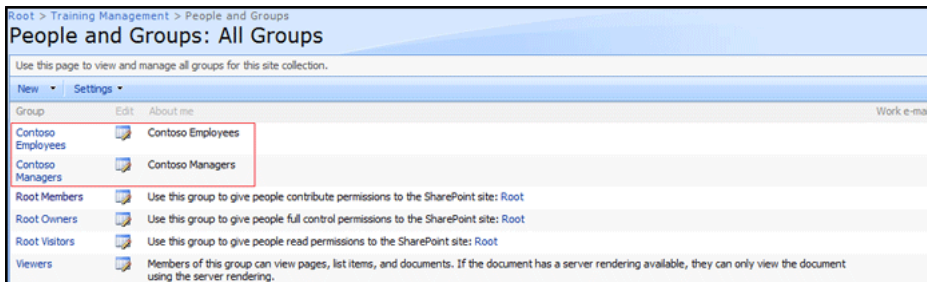
Using SharePoint Permissions

The Training Management application uses custom SharePoint groups and custom permission levels to enforce security. This topic discusses the design of the security model and how it is implemented.

SharePoint Groups

The Training Management application uses two custom SharePoint groups. They are Contoso Employees and Contoso Managers. Members of the Contoso Employees group can see the available training courses, register for those courses, and see the status of their registrations. The Contoso Managers have the same abilities as the Contoso Employees, and they can also view their registration approval tasks and approve and reject registrations. The following figure shows the two custom groups.

Training Management security groups



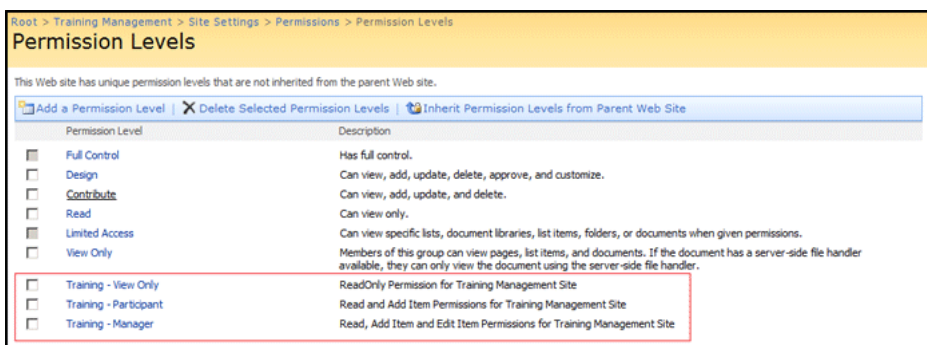
SharePoint Permission Levels

The Training Management application also uses custom permission levels to enforce the security requirements. The Training – View Only permission level allows users to view pages and list items. The Training – Participant permission level allows users to view pages, view list items, and add list items. The Training – Manager permission level allows users to view pages, view list items, add list items, and edit list items. These permission levels are applied to the Contoso Employees and Contoso Managers groups to control their access to the following lists on the Training Management site:

- The list of training courses
- The list of registrations
- The list of approval tasks

The following figure illustrates the different permission levels.

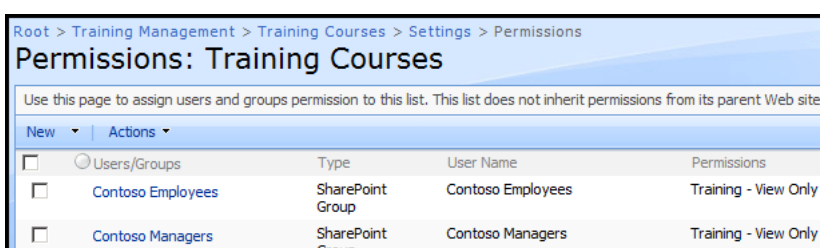
Training Management permission levels



Training Courses List Permissions

Both the Contoso Employees group and the Contoso Managers group have the Training – View Only permission for the training courses list. Only site administrators can add courses and edit current information. The following figure illustrates the permission levels for the list of training courses.

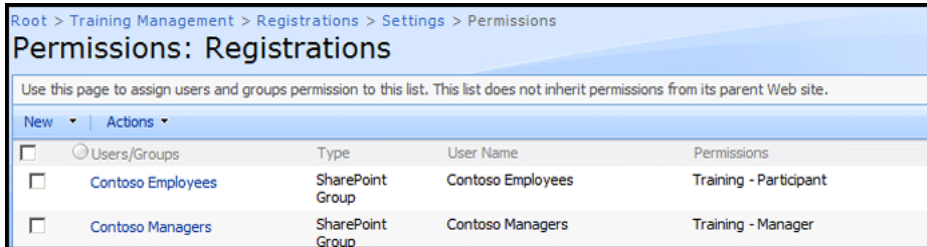
Permissions for the training courses



Registrations List Permissions

Members of the Contoso Employees group have the Training – Participant permission level for the registration list. This level allows them to register for courses by adding an item to the Registrations list but not to edit the list. Members of the Contoso Managers group have the Training – Manager permission level. This allows them to both add and edit the registration list items. They need these abilities to update the registration status. The following figure illustrates the permission levels for the registration list.

Permissions for the registration list

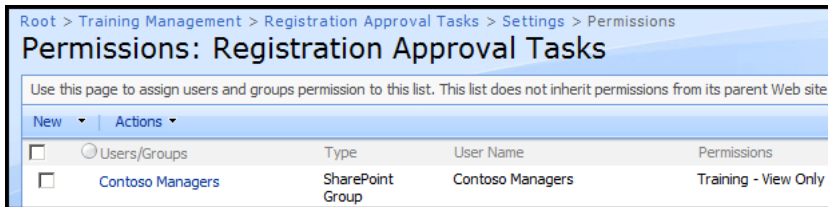


Users/Groups	Type	User Name	Permissions
Contoso Employees	SharePoint Group	Contoso Employees	Training - Participant
Contoso Managers	SharePoint Group	Contoso Managers	Training - Manager

Registration Approval Tasks List Permissions

Only members of the Contoso Managers group can access the registration approval tasks list. They need this ability to view their pending approval tasks. The following figure illustrates the permission level for r1

Permissions for the registration approval tasks list



Users/Groups	Type	User Name	Permissions
Contoso Managers	SharePoint Group	Contoso Managers	Training - View Only

Security and Permission Provisioning

A SharePoint feature receiver class performs the security provisioning for the Training Management application. This class uses the SharePoint object model to create the SharePoint groups and permission levels. Methods of this class are invoked when the site is created with the Contoso Training Management template.

The SharePoint feature receiver class is derived from the abstract class **SPFeatureReceiver**. For more information about the **SPFeatureReceiver** class, see [SPFeatureReceiver Class \(Microsoft.SharePoint\)](#) on MSDN.

The derived class is named **WebFeatureReceiver**. It implements a method named **FeatureActivated** that is called by SharePoint during the ContosoTrainingManagementWeb feature activation. SharePoint invokes this method because the developer specifies it in the **ReceiverClass** attribute of the Feature.xml file. To view the Feature.xml file, click the **WSP View** tab in the View pane in Visual Studio.

The following code is for the **FeatureActivated** method of the **WebFeatureReceiver** class. It is located in the FeatureReceivers\WebFeatureReceiver.cs file in the Contoso.TrainingManagement project.

```
C#
public override void FeatureActivated(SPFeatureReceiverProperties properties)
{
    SPWeb web = properties.Feature.Parent as SPWeb;
    this.ProvisionSecurity(web);
    UpdateTopLinkBar(web);
    this.UpdateQuickLaunch(web);
}
```

This code provisions the application's security model.

To create the security groups and permission levels, the **FeatureActivated** method invokes a helper method named **ProvisionSecurity**. This method performs the following tasks:

- It sets the current Web to use locally defined roles.
- It creates two groups named Contoso Employees and Contoso Managers.
- It gives site-read access to the newly created groups.
- It creates new permissions named Training - View Only, Training – Participant, and Training - Manager with permissions that are appropriate to each permission level.
- It sets the permission levels for the training course list, the registration list, and the registration approval task list.

Organizing Features

A SharePoint feature is a collection of SharePoint elements. It allows you to create server-side, file system-level customizations. In previous versions of SharePoint, developers modified the Onet.xml file to customize elements such as lists. In the current version of SharePoint, you can use features to make smaller and more modular changes that can be installed on new or existing sites. For example, you can use a single feature to add a set of lists and libraries combined with a workflow that, as a whole, enable a business process to function within SharePoint.

A feature contains one Feature.xml file. This file contains one or more feature elements. An element is an atomic unit within the feature. Examples of elements include content types, controls, and lists. A feature should contain all the elements that are required to provide some piece of functionality. Features are meant to be reusable and they should be as light and modular as possible. Features can have dependencies on other features, but these dependencies can make the activation and deactivation processes more complex.

A feature is activated for a single scope. The following are the four possible scopes:

- **Web.** The scope of the feature is the Web site.
- **Site.** The scope of the feature is the site collection.
- **WebApplication.** The scope of the feature is the Web application.
- **Farm.** The scope of the feature is the server farm.

A feature should contain only elements that are appropriate for its scope. For a list of the feature scopes and their appropriate elements, see [SharePoint Feature](#) on MSDN.

Organization of the Training Management Application

The Training Management SharePoint Solution is organized into two SharePoint features:

- One feature contains all the site collection-level elements, such as content types and Web Parts. It is named ContosoTrainingManagementSiteCollection.
- One feature contains all the site-level elements, such as list definitions, list instances, and custom ASPX forms. It is named ContosoTrainingManagementWeb.

The Registration Approval Workflow Solution

Special considerations were made in the packaging of the registration approval workflow. The registration approval workflow was intentionally packaged in a separate SharePoint solution. The registration approval workflow is essential for a fully functional training management application, but it should be treated differently because of its life cycle management. Registration approval workflow should be versioned and upgraded independently from the rest of the Training Management application. One benefit to this is that when the registration approval workflow is upgraded, any existing workflow instances can continue to run side by side with new versions of the registration approval workflows.

Web Solution Package

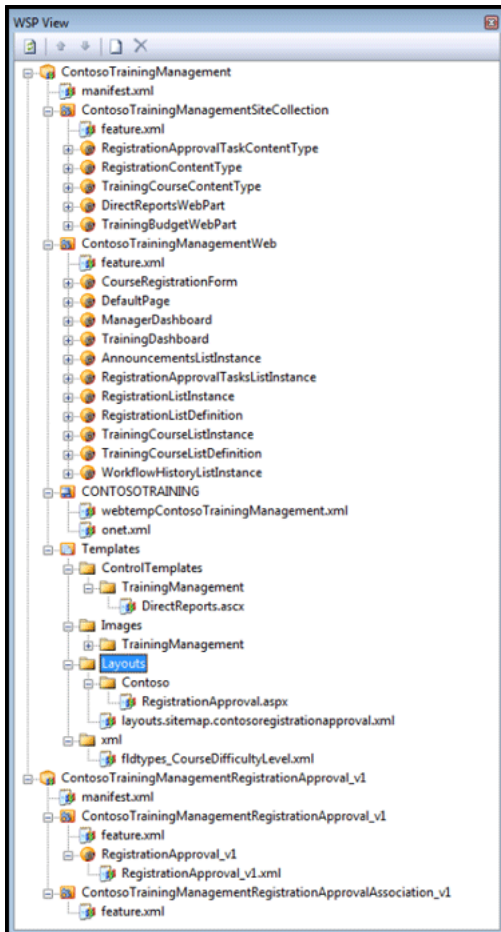
A Web solution package (WSP) is a deployable and reusable way to bundle components that extend SharePoint. A SharePoint solution is a file that has a CAB file-based format but has a .wsp file name extension. It contains a list of SharePoint features, site definitions, assemblies, template files, and root files. The Windows SharePoint Services solution framework provides many benefits for deploying solutions. For more information about the benefits of the SharePoint solution framework, see [Solutions Overview](#) on MSDN.

Using Visual Studio extensions for Windows SharePoint Services and WSP View to Organize Features

Visual Studio extensions for Windows SharePoint Services uses the WSP View add-in to provide developers with a graphical interface for managing and organizing SharePoint features. The WSP View organizes the SharePoint artifacts in a Visual Studio solution (a .sln file) by SharePoint features, site definitions, and template files. By default, the WSP View creates a SharePoint feature for each item in the SharePoint solution. You can restructure the features by dragging elements from one feature to another. The WSP View also allows you to rename and delete features.

The following figure illustrates the reorganized structure of the Training Management features.

Structure of Training Management solution



SharePoint Implementation Basics

This topic describes how to solve some of the fundamental implementation tasks that developers face when they create a SharePoint application. The following subtopics describe how to perform the following tasks:

- [Managing ASPX Pages in SharePoint](#). This topic describes how to manage _layout and content pages.
- [Associating Workflows with Lists](#). This topic describes how to programmatically associate a workflow with a list.
- [Using Custom Field Types and Field Controls](#). This topic describes how to implement custom field types and custom field controls.
- [Memory Management for SharePoint Objects](#). This topic describes how to dispose of SharePoint objects that use managed memory.
- [Development How-to Topics](#). This topic includes procedures for performing some basic SharePoint tasks. These include the following:
 - [How to: Create a Custom Content Type with Event Receivers](#). This topic demonstrates how to create a custom content type in Visual Studio using Visual Studio extensions for Windows SharePoint Services.
 - [How to: Wrap a User Control Inside of a Web Part for SharePoint](#). This topic demonstrates how to create a Web Part to serve as a wrapper for an ASP.NET user control that is hosted inside of a SharePoint application.
 - [How to: Debug SharePoint Applications](#). This topic demonstrates how to debug an application by using Visual Studio extensions for Windows SharePoint Services.
 - [How to: Implement a SharePoint Workflow with ASP.NET Forms](#). This topic is a high-level description of how to use Visual Studio to create ASP.NET custom workflow task forms.
 - [How to: Perform ASP.NET-Related Development with Visual Studio](#). This topic demonstrates how custom ASPX pages and custom ASCX controls can be developed in a Web Application project and then deployed using an extensions project.
 - [How to: Programmatically Configure a WCF Endpoint](#). This topic includes procedures that describe how developers can create a SharePoint Web Part that uses data from a Windows Communication Foundation (WCF) Web service. The Web Part programmatically configures the WCF clients.
 - [How to: Use the Hierarchical Configuration to Store an Endpoint Address](#). This topic demonstrates how to store and retrieve the endpoint addresses of the pricing and product catalog services with the Web part that is implemented in [How to: Programmatically Configure a WCF Endpoint](#).

Managing ASPX Pages in SharePoint

SharePoint has two types of ASPX pages:

- An application page, which is also known as a `_layouts` page
- A content page

Each of these page types is stored in a different location in SharePoint. When developers create custom ASPX pages, they must decide on the page type. For more information about page types, including ASPX pages and master pages, see [SharePoint Page Types](#) on MSDN.

Typically, application pages are created to allow users to manage settings for lists, sites, and site collections. These pages are available across the server farm and are stored in the `_layouts` virtual directory of SharePoint Web sites. This allows these pages to run on any site collection in the farm. Application pages are commonly used when a page must be accessible at any level of a site hierarchy and contain functionality such as managing settings or cross-site reporting. For more information, see [Application _layouts Page Type](#) on MSDN.

Content pages are associated with SharePoint Web sites and typically provide content for that site instead of providing management capabilities. Content pages are stored either in the root of the site or in a library where they can be edited from within the SharePoint user interface or with Microsoft Office SharePoint Designer 2007. Pages that have not been customized, also known as *uncustomized* or *ghosted* pages, are stored on the front-end Web server. Pages that have been customized, also known as *customized* or *unghosted* pages, are stored in the SharePoint content database. For more information, see [Content Page Type](#) on MSDN.

Associating Workflows with Lists

There are two ways to associate a workflow with a list in SharePoint. One way is to use the SharePoint user interface. For this option, some workflows provide specific association forms that capture specific information that the workflow requires. You can also programmatically perform the association. For information about how to do this, see [Using Custom Field Types and Field Controls](#), which uses the Training Management application as an example.

For the Training Management application, a SharePoint feature receiver associates the registration approval workflow with the registration list. The association occurs when the ContosoTrainingManagementRegistrationApprovalAssociation_v1 feature is activated. Programmatically associating a workflow with a list requires the following components:

- A workflow template
- A task list
- A workflow history list
- The list instance that will be associated with the workflow

The Training Management application retrieves the registration list, the registration approval task list, and the workflow history list that are defined in the ContosoTrainingManagementWeb SharePoint feature. The following code from the Training Management application demonstrates how to do this.

C#

```
SPList taskList = web.Lists[List.RegistrationApprovalTasks];
SPList historyList = web.Lists[List.WorkflowHistory];
SPList registrationList = web.Lists[List.Registrations];
```

After the lists are retrieved, the **SPWorkflowManager** class retrieves the course registration approval workflow template. You must specify the task list and workflow history list that are used by the workflow association. The last step is to add the workflow association to the registration list and enable the workflow. The following code from the Training Management application demonstrates how to do this.

C#

```
var existingAssociation =
registrationList.WorkflowAssociations.GetAssociationByName(WorkflowTemplates.RegistrationApprovalName_v1, CultureInfo.CurrentCulture);
if (existingAssociation == null)
{
    // Create a workflow manager and associate the Course Registration
    // Approval Workflow template to our Registrations list.
    SPWorkflowManager workflowManager = web.Site.WorkflowManager;
    SPWorkflowTemplateCollection templates =
workflowManager.GetWorkflowTemplatesByCategory(web, null);
    SPWorkflowTemplate template = templates.GetTemplateByBaseID(new
Guid(WorkflowTemplates.RegistrationApprovalBaseId_v1));
    SPWorkflowAssociation association = SPWorkflowAssociation.CreateListAssociation(template,
template.Name, taskList, historyList);
    association.AllowManual = true;
    association.AutoStartCreate = true;
    registrationList.AddWorkflowAssociation(association);
    registrationList.Update();
    association.Enabled = true;
}
```

Using Custom Field Types and Field Controls

Fields in SharePoint are metadata elements that can be added as columns to a SharePoint list. There are 32 different of field types in Windows SharePoint Services 3.0. For example, there is the [SPFieldAllDayEvent](#) class. This is a Boolean field that represents an all-day activity without a start time or an end time. Custom field types are necessary when the application data does not conform to one of these default types. You may also need custom field types in the following situations:

- The default validation rules are not applicable.
- The field requires custom logic.
- The field requires custom rendering.

For more information about standard SharePoint field types, see [SPField Class \(Microsoft.SharePoint\)](#) on MSDN.

The Difficulty Level Field

The Training Course Management application uses one custom field type named **CourseDifficultyLevelField**. It corresponds to the Difficulty Level column in the list of training courses that is displayed on the user's dashboard. The **CourseDifficultyLevelField** is set to a numerical value between 0 and 5. This is represented on the Web page by a series of green or gray squares. While users are in the Edit and New modes for the list, they can set the Course Difficulty Level field by clicking the squares.

The following figure illustrates the training course list with the Difficulty Level column set to different values for two courses.

Course difficulty field control

Training Courses						
Title	Code	Enrollment Deadline	Start Date	End Date	Cost	Difficulty Level
Beginning C# 	ProgC101	10/20/2008	11/10/2008	11/13/2008	\$200.00	 
Office 2007 Training 	OFFICE07	12/31/2010	1/1/2011	1/6/2011	\$100.00	
 Add new item						

The Course Difficulty Level Field Class

Custom field types require two components. One is the custom field class. A custom field class must derive either from the **SPField** class or a subclass of **SPField**. For example, the Training Management application's **CourseDifficultyLevelField** class derives from the **SPFieldNumber** class. It overrides the **FieldRenderingControl** property and the **GetValidatedString** method. The **FieldRenderingControl** property instantiates the custom field control. The following code demonstrates how to override the **FieldRenderingControl** property.

```
C#
public override BaseFieldControl FieldRenderingControl
{
    [SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
    get
    {
        BaseFieldControl fieldControl = new CourseDifficultyLevelFieldControl();
        fieldControl.FieldName = this.InternalName;

        return fieldControl;
    }
}
```

The **GetValidatedString** method validates that the value of the **CourseDifficultyLevelField** is a number between 0 and 5. This is demonstrated in the following code.

```
C#
[SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
public override string GetValidatedString(object value)
{
    string difficultyLevel = value.ToString();
    short difficultyLevelValue = Int16.Parse(difficultyLevel);
    if (difficultyLevelValue < 0 || difficultyLevelValue > 5)
    {
        throw new SPFieldValidationException("Invalid difficulty level. Difficulty level must be between 0 and 5.");
    }

    return difficultyLevel;
}
```

For more information about the **SPField** class, see [SPField Class \(Microsoft.SharePoint\)](#) on MSDN.

The CourseDifficultyLevelField Definition

The field type definition is an XML file (fldtypes_CourseDifficultyLevel.xml) that describes the assembly that contains the custom field class and any custom rendering instructions. All custom field type definition files start

with the text **fldtypes_** and are deployed to the Templates/xml folder in the SharePoint 12 hive. The **CourseDifficultyLevelField** definition contains a custom **RenderPattern** element for custom rendering. This element displays a series of squares that correspond to the difficulty level value specified in the custom field. For more information about patterns of custom field rendering, see [Patterns of Custom Field Rendering](#) on MSDN. The following code is a partial example of how to use the **RenderPattern** element.

XML

```
<RenderPattern Name="DisplayPattern">
  <HTML>
    <![CDATA[<div align='right'>]]>
  </HTML>
  <Switch>
    <Expr>
      <Column />
    </Expr>
    <Case Value="1">
      <HTML>
        <![CDATA[<img src='/_layouts/images/trainingmanagement/level1.gif' />]]>
        <![CDATA[<img src='/_layouts/images/trainingmanagement/level0.gif' />]]>
        <![CDATA[<img src='/_layouts/images/trainingmanagement/level0.gif' />]]>
        <![CDATA[<img src='/_layouts/images/trainingmanagement/level0.gif' />]]>
        <![CDATA[<img src='/_layouts/images/trainingmanagement/level0.gif' />]]>
      </HTML>
    </Case>
    ...
  </Switch>
  <HTML>
    <![CDATA[</div>]]>
  </HTML>
</RenderPattern>
```

The Course Difficulty Level Field Control

A custom field rendering control can render custom field types in the New and Edit modes.

The following figure illustrates the field rendering control as it appears in Edit mode.

Field rendering control in Edit mode

The screenshot shows a SharePoint 'New Item' form for 'Training Courses'. The form has a yellow header bar with the text 'Training Courses: New Item'. Below the header, there are 'OK' and 'Cancel' buttons. The form contains several fields: 'Title' (with value 'Beginning C#'), 'Code' (with value 'ProgC101'), 'Description' (with value 'This is an introductory course in C#. No programming experience is required.'), 'Enrollment Deadline' (with value '10/20/2008'), 'Start Date' (with value '11/10/2008'), 'End Date' (with value '11/13/2008'), 'Cost' (with value '200'), and 'Difficulty Level'. The 'Difficulty Level' field is rendered as a series of five colored squares (green, light green, light blue, light blue, light blue), with the first square being green, indicating the selected difficulty level. There are also 'OK' and 'Cancel' buttons at the bottom of the form.

A custom field rendering control must derive from the **BaseFieldControl** class or a subclass of the **BaseFieldControl** class. The **CourseDifficultyLevelFieldControl** class derives from the **BaseFieldControl** class and overrides the **CreateChildControls**, **UpdateFieldValueInItem**, and **RenderFieldForDisplay** methods. The **CreateChildControls** method adds a series of **ImageButton** controls that display images of green or gray squares. Users can click the squares to specify a difficulty level for a course. The **UpdateFieldValueInItem** method fires during a postback. Typically, this is when the value of the data held in the field is updated. The **RenderFieldForDisplay** method is only used to render the control in the Display mode.

For more information about the **BaseFieldControl** class, see [BaseFieldControl Class \(Microsoft.SharePoint.WebControls\)](#) on MSDN. For more information about custom field types and custom field rendering controls, see [Custom Field Types](#) on MSDN.

Memory Management for SharePoint Objects

Windows SharePoint Services objects, including **SPSite** objects and **SPWeb** objects, are created as managed objects. However, these objects use unmanaged code and memory to perform the majority of their work. The managed part of the object is small; the unmanaged part of the object is much larger. Because the smaller managed part of the object does not put memory pressure on the .NET Framework runtime's garbage collector, the garbage collector may not release the object from memory in a timely manner. Applications that use **IDisposable** objects in Windows SharePoint Services must dispose of those objects when they no longer need them.

For more information about this topic, see [Best Practices: Using Disposable Windows SharePoint Services Objects](#) on MSDN and [Roger Lamb's SharePoint Developer Blog](#).

For more information about common SharePoint coding issues, see [Best Practices: Common Coding Issues When Using the SharePoint Object Model](#) on MSDN.

Development How-to Topics

This topic describes how to perform some of the most common tasks that occur while developing a SharePoint application. The following subtopics describe how to perform the following tasks:

- [How to: Create a Custom Content Type with Event Receivers](#). This topic demonstrates how to create a custom content type in Visual Studio using Visual Studio extensions for Windows SharePoint Services.
- [How to: Wrap a User Control Inside of a Web Part for SharePoint](#). This topic demonstrates how to create a Web Part to serve as a wrapper for an ASP.NET user control that is hosted inside of a SharePoint application.
- [How to: Debug SharePoint Applications](#). This topic demonstrates how to debug an application by using Visual Studio extensions for Windows SharePoint Services.
- [How to: Implement a SharePoint Workflow with ASP.NET Forms](#). This topic is a high-level description of how to use Visual Studio to create ASP.NET custom workflow task forms.
- [How to: Perform ASP.NET-Related Development with Visual Studio](#). This topic demonstrates how custom ASPX pages and custom ASCX controls can be developed in a Web Application project and then deployed using an extensions project.
- [How to: Programmatically Configure a WCF Endpoint](#). This topic includes procedures that describe how developers can create a SharePoint Web Part that uses data from a Windows Communication Foundation (WCF) Web service. The Web Part programmatically configures the WCF clients.
- [How to: Use the Hierarchical Configuration to Store an Endpoint Address](#). This topic shows you how to store the WCF service's endpoint address as a configuration setting.

How to: Create a Custom Content Type with Event Receivers

This topic demonstrates how to create a custom content type in Visual Studio using Visual Studio extensions for Windows SharePoint Services.

To create a custom content type

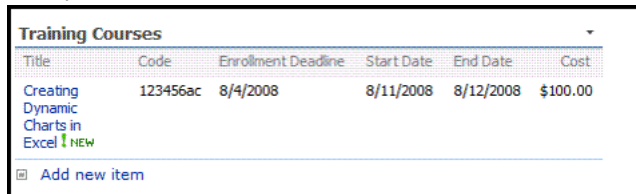
1. In Visual Studio, click the **File** menu, point to **New**, and then click **Project**.
2. Under **SharePoint**, click the Empty project template.
3. Select Full Trust (Deploy to GAC) in Select Trust Level window.
4. In Visual Studio, right-click the project name, point to **Add**, and then click **New Item**.
5. In the Categories pane, click **SharePoint**. In the Templates pane, click **Content Type**, enter the content type name (for example, the name of the content type for a training course is **TrainingCourseContentType**) and then click **Add**.
6. In the **Content Type Settings** drop-down box, click a base content type (for this procedure, use the Item base content type), select the **Add with Event Receiver** check box, and then click **OK**.
7. Visual Studio generates a code file for the list item event receiver and list event receiver and the Elements XML file, which is part of the feature that will define the content type. Open the XML file.
8. Change the **Name** and **Description** attributes to something more relevant than the default settings. For example, the **Name** attribute of the training course content type is **TrainingCourseContentType** and the **Description** attribute is **Training Course Content Type**.
9. The generated content type XML file includes a template for custom fields. Uncomment the **<Field>** element and provide a new name, display name, and type. The **StaticName** attribute should be deleted as this attribute is not used. The following is the definition for the **TrainingCourseEnrollmentDate** field.

XML

```
<Field ID="{43568365-8448-4130-831C-98C074B61E89}" Type="DateTime"
Name="TrainingCourseEnrollmentDate" DisplayName="Enrollment Deadline" Hidden="FALSE"
Required="TRUE" Sealed="FALSE" Format="DateOnly" />
```

10. Copy and paste the field definition from step 8 for each of your custom fields. For each field you create, you must generate a new GUID for the **ID** attribute. To do this, click **Create GUID** on the **Tools** menu, click **Registry Format**, click **Copy**, and then click **Exit**.
11. There is a single field reference included in the file. Field references correspond to the columns in the list view. For example, the following figure has six columns and six field references. Change the name of the field reference to something more relevant than the default setting. For example, the name of one of the field references for the training course content type is **TrainingCourseCode**.

Example of field references and columns



Title	Code	Enrollment Deadline	Start Date	End Date	Cost
Creating Dynamic Charts in Excel NEW	123456ac	8/4/2008	8/11/2008	8/12/2008	\$100.00

 Add new item

12. There should be a FieldRef ID for each column that appears on the list view. To add additional field references, go to the **FieldRefs** section and add a new field reference line. The following is an example.

XML

```
<FieldRef ID="" Name="TrainingCourseDescription"/>
```

13. For each FieldRef, the FieldRef ID must be an ID of one of your custom fields or an ID of an existing field.
14. Visual Studio extensions for Windows SharePoint Services will generate stubs for your list item and list event handlers. To implement the **ItemAdded** event handler, uncomment the following lines.

C#

```
public override void ItemAdded(SPItemEventProperties properties)
{
}
```

15. Implement your business logic.
16. Visual Studio extensions for Windows SharePoint Services automatically adds the **Receivers** element to the Content Type XML file the next time that you refresh the WSP view (to do this, point to **Other Windows** on the **View** menu, and then click **WSP View**) or deploy the solution (to do this, right-click the project, and then click **Deploy**).

To verify the deployment of the content type, browse the site where the content type was deployed. Browse to the Site Settings page (to do this, click the **Site Actions** tab, and then click **Site Settings**), and click **Site Content Types** under **Galleries**.

How to: Wrap a User Control Inside of a Web Part for SharePoint

This topic demonstrates how to create a Web Part to serve as a wrapper for an ASP.NET user control that is hosted inside of a SharePoint application. It also demonstrates how to create Web Part properties that are propagated to the user control. The topic uses [Windows SharePoint Services 3.0 Tools: Visual Studio 2008 Extensions, Version 1.3](#).

Creating the Web Part

This procedure demonstrates how to create an ASP.NET Web Part to wrap the user control.

To create the ASP.NET Web Part

1. In Visual Studio, point to **New** on the **File** menu, and then click **Project**.
2. In the Project types pane, click **SharePoint**. In the Templates pane, click **Web Part**. In the **Name** box, type **MyWebPart**, and then click **OK**.
3. In the Select Trust Level window, select **Full Trust (Deploy to GAC)**.
4. Rename the WebPart1 folder to MyWebPart. In the Rename WebPart window, click **Yes**.
5. Delete the constructor of the **MyWebPart** class. The following code shows the corrected file.

C#

```
public class MyWebPart : System.Web.UI.WebControls.WebParts.WebPart
{

    protected override void CreateChildControls()
    {
        base.CreateChildControls();

        // TODO: add custom rendering code here.
        // Label label = new Label();
        // label.Text = "Hello World";
        // this.Controls.Add(label);
    }
}
```

6. Open the MyWebPart.webpart file and replace all instances of WebPart1 with MyWebPart. The following code shows the corrected file.

XML

```
<webParts>
  <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <!--
        The following Guid is used as a reference to the web part class,
        and it will be automatically replaced with actual type name at
        deployment time.
      -->
      <type name="61ed6d82-91e5-4ef3-910e-33d0f771341e" />
      <importErrorMessage>Cannot import MyWebPart Web Part.</importErrorMessage>
    </metaData>
    <data>
      <properties>
        <property name="Title" type="string">MyWebPart Web Part</property>
        <property name="Description" type="string">MyWebPart Description</property>
      </properties>
    </data>
  </webPart>
</webParts>
```

7. Open the MyWebPart.xml file and replace all instances of WebPart1 with MyWebPart. The following code shows the corrected file.

XML

```
<Elements Id="61ed6d82-91e5-4ef3-910e-33d0f771341e"
xmlns="http://schemas.microsoft.com/sharepoint/" >
  <Module Name="WebParts" List="113" Url="_catalogs/wp">
    <File Path="MyWebPart.webpart" Url="MyWebPart.webpart" Type="GhostableInLibrary" />
  </Module>
</Elements>
```

Creating the ASP.NET User Control

This procedure demonstrates how to create an ASP.NET user control that uses SharePoint.

To create the ASP.NET user control

1. In Visual Studio, right-click the MyWebPart solution, point to **Add**, and then click **New Project**.
2. In the Project Types pane, click **Web**. In the Templates pane, click **ASP.NET Web Application**. Name the project MyUserController, and then click **OK**.

3. Delete the Default.aspx file.
4. Right-click the MyUserController project, point to **Add**, and then click **New Item**.
5. In the Categories pane, click **Web**. In the Templates pane, click **Web User Control**. Name the control MyUserController.ascx, and then click **Add**.
6. Delete the **CodeBehind** attribute in the MyUserController.ascx file. The following code shows the corrected file.

XML

```
<%@ Control Language="C#" AutoEventWireup="true" Inherits="MyUserController.MyUserController" %>
```

7. Right-click the MyWebPart project, point to **Add**, and then click **New Folder**. Name the folder Templates.
8. Right-click the Templates folder, point to **Add**, and then click **New Folder**. Name the folder ControlTemplates.
9. Right-click the **ControlTemplates** folder, point to **Add**, and then click **New Folder**. Name the folder MyWebPart.
10. Right-click the MyWebPart folder, point to **Add**, and then click **Existing Item**. Browse to the MyUserController.ascx file, and then click **Add as Link** in the drop-down box.
11. In the MyUserController project, open the MyUserController.ascx file. Replace the existing value of the **Inherits** attribute with **MyWebPart.MyUserController, MyWebPart, Version=1.0.0.0, Culture=neutral, PublicKeyToken=<your public key token>**. This allows the code-behind to be compiled into the MyUserController.dll. The following code shows the corrected file.

Note:

To get the specific public key token for your signed assembly, use the sn.exe -T command line tool. For more information, see [Strong Name Tool \(sn.exe\)](#) on MSDN.

XML

```
<%@ Control Language="C#" AutoEventWireup="true" Inherits="MyWebPart.MyUserController, MyWebPart, Version=1.0.0.0, Culture=neutral, PublicKeyToken=<your public key token>" %>
```

12. In the MyUserController project, open the MyUserController.ascx file in the designer view. Click the Toolbox, and then add a panel using a drag-and-drop operation.
13. In the MyUserController project, open the MyUserController.ascx.cs file and the MyUserController.ascx.designer.cs file. Change the namespace to **MyWebPart** in both files.
14. In the MyUserController project, right-click **References**, and then click **Add Reference**. In the **Add Reference** box, click **Windows SharePoint Services**. This adds a reference to the **Microsoft.SharePoint** namespace.
15. In the MyUserController.ascx.cs file, add a **using** statement for the **Microsoft.SharePoint** namespace and for the **System.Drawing** namespace. The following code shows the **using** statements.

C#

```
using Microsoft.SharePoint;
using System.Drawing;
```

16. In the MyUserController.ascx.cs file, if applicable, remove the **using** statements for System.Data, System.Linq, System.Xml.Linq, and System.Xml.
17. In the MyUserController project, open **References**. Right-click **System.Data**, and then click **Remove**. Do the same for System.Linq, System.Xml.Linq, and System.Xml, if applicable.
18. In the MyWebPart project, right-click **References**, and then click **Add References**. In the **Add Reference** box, click **System.Drawing**.
19. In the MyUserController project, open the MyUserController.ascx.cs file. Add a public **SPWeb** property named **Web** to the **MyUserController** class. The following code demonstrates this.

C#

```
public SPWeb Web { get; set; }
```

20. Add a public string property named **TextColor** to the **MyUserController** class. The following code demonstrates this.

C#

```
public string TextColor { get; set; }
```

21. Add the following code to the **Page_Load** method. This code modifies the **Style** property of the **Panel1** panel with the value of the user-defined **TextColor** property. It also displays the name of the current user inside of the panel.

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    if ( !String.IsNullOrEmpty(this.TextColor) )
    {
        Color myColor = Color.FromName(this.TextColor);
        if ( myColor != null )
        {
            Panel1.Style.Add("color", myColor.Name);
        }
    }

    string currentUser = this.Web.CurrentUser.Name;
    Panel1.Controls.Add(new LiteralControl(String.Format("Hello {0}!", currentUser)));
}
```

22. Right-click the **MyWebPart** solution, and then click **Build Solution**.

Wrapping the User Control and Connecting the Properties

This procedure demonstrates how to wrap the user control inside of the Web Part and connect the properties between SharePoint, the Web Part, and the user control.

To wrap the user control and connect the properties

1. In the **MyWebPart** project, open the **MyWebPart.cs** file.
2. Add a public string property named **TextColor** to the **MyWebPart** class. Add the following attributes to the **TextColor** property. These properties are specific to Web Parts. They customize the rendering of the property while the Web Part is in Edit mode in SharePoint. The following code demonstrates how to do this.

```
C#
/// <summary>
/// The color of the text to set in the child user control.
/// </summary>
[Personalizable(PersonalizationScope.Shared)]
[WebBrowsable(true)]
[WebDisplayName("Text Color")]
[WebDescription("Color of text in the main content of the Web Part.")]
[SPWebCategoryName("Development")]
public string TextColor
{
    get;
    set;
}
```

3. Replace the **TODO** comments in the **CreateChildControls** method with the following code. This code loads the **MyUserControl** user control and adds it to the Web Part. If the user control is not found, an error message is displayed.

```
C#
try
{
    // This loads a user control.
    MyUserControl myUserControl =
    (MyUserControl)Page.LoadControl("~/_controltemplates/MyWebPart/MyUserControl.ascx");
    myUserControl.Web = SPContext.Current.Web;
    myUserControl.TextColor = this.TextColor;

    // This adds it to the controls collection of the Web Part.
    this.Controls.Add(myUserControl);
}
catch ( HttpException ex )
{
    this.Controls.Add(new LiteralControl("<br />An unexpected error occurred loading Web
Part. " + ex.Message));
}
```

4. Add a **using** statement for the **System.Web** namespace. The following code demonstrates how to do this.

```
C#
using System.Web;
```

5. Rebuild the solution.

Deploying the Web Part and Testing Functionality

This procedure demonstrates how to deploy the Web Part and the user control and how to test their functionality. This procedure assumes that you already have an instance of a SharePoint site installed.

To deploy the Web Part

1. Right-click the **MyWebPart** project, and then click **Properties**.
2. On the **Debug** tab, type the URL of your SharePoint test site in the **Start browser with URL** box.
3. Right-click the **MyWebPart** project, and then click **Deploy**. (Make sure that the VSeWSS Services Application Pool ID has permissions for the URL that you typed in step 2). See the note at the end of the procedure.
4. Browse to your SharePoint test site home page.
5. In the **Site Actions** drop-down box, click **Edit Page**.
6. Click **Add a Web Part** in one of the Web Part zones on the page, click the **MyWebPart** Web Part (located in the **Miscellaneous** section), and then click **Add**.
7. Click **Exit Edit Mode**. You should now see a "hello" message with your user name in black text.
8. To change the color of the text, click the drop-down box on the Web Part, and then click **Modify Shared Web Part**.
9. Expand the **Development** group, enter the name of a valid color, such as **Blue**, in the **Text Color** box, and then click **Apply**.
10. Click **Exit Edit Mode**. You should now see a "hello" message with your user name in blue text.

Note:

The **VSeWSS Application Pool** account might need to be configured as a member of the Site Collection Administrators group. To do this, browse to http://YourSiteCollection/_layouts/mngsiteadmin.aspx and specify the application pool account.

For more information about this topic, see [Developing Web Parts in Windows SharePoint Services](#) on MSDN.

How to: Debug SharePoint Applications

You can greatly simplify debugging by using Visual Studio extensions for Windows SharePoint Services. This topic includes procedures that explain the following:

- [Debugging with Visual Studio extensions for Windows SharePoint Services](#)
- [Performing manual debugging](#)
- [Performing remote debugging](#)

Debugging with Visual Studio extensions for Windows SharePoint Services

Press the F5 key to begin debugging with Visual Studio extensions for Windows SharePoint Services. The following procedure demonstrates how to enable F5 debugging.

To enable F5 debugging

1. Locate and open the target SharePoint application's Web.config file. By default, it is located in C:\Inetpub\wwwroot\wss\VirtualDirectories\80.
2. Find the following line of code and change the **debug** attribute to **true**.

XML

```
<compilation batch="false" debug="false">
```

3. Save the changes to the Web.config file.
4. In Visual Studio, right-click the SharePoint project, and then click **Properties**.
5. Click the **Debug** tab, and then type the target SharePoint URL in the **Start browser with URL** box.
6. Place a breakpoint in the code, and then press F5.

Performing Manual Debugging

You can debug a SharePoint application without using Visual Studio extensions for Windows SharePoint Services by attaching it to the W3wp.exe process. The following procedure demonstrates how to do this.

To attach to the W3wp.exe process

1. Make sure that the SharePoint application is using the latest compiled source code. This may require that you recompile your source and install the assemblies into the global assembly cache. Alternatively, you may need to copy the assemblies to the bin folder of SharePoint's virtual directory. In either case, you must run the Microsoft Internet Information Services (IIS) command-line utility Iisreset.exe.
2. Place a breakpoint in the code you want to debug.
3. On the **Debug** menu, click **Attach to Process**.
4. In the list of available processes, find the W3wp.exe process, and then do the following, as applicable:
 - a. If the W3wp.exe process is not listed, make sure that the **Show processes from all users** and **Show processes in all sessions** check boxes are selected.
 - b. If the W3wp.exe process still is not listed, open a Web browser and navigate to the application's SharePoint site. After you navigate to the SharePoint site, return to Visual Studio. In the **Attach to Process** dialog box, click the **Refresh** button to locate the W3wp.exe process.
 - c. If you see multiple W3wp.exe processes listed and are not sure which one of them is running the code, click them all.
5. Click the **Attach** button.

Performing Remote Debugging

You can debug a SharePoint application when it is running on a remote computer by attaching it to the remote W3wp.exe process. The following procedure demonstrates how to do this.

Note:

For information about configuring a remote server for debugging, see [How to: Set Up Remote Debugging](#) in the Visual Studio 2008 Developer Center on MSDN.

To attach to a remote W3wp.exe process

1. Make sure that the remote SharePoint application is using the latest compiled source code. This may mean that you must recompile your source and install the assemblies into the global assembly cache of the remote server. Alternatively, you may need to copy the assemblies to the bin folder of SharePoint's virtual directory on the remote computer. In either case, you must run the Microsoft Internet Information Services (IIS) command-line utility Iisreset.exe.
2. Make sure that the Visual Studio 2008 Remote Debugger is installed on the remote computer.
3. Start the Visual Studio 2008 Remote Debugger on the remote computer. For more information about using the remote debugger, see [How to: Run the Remote Debugging Monitor](#) on MSDN.
4. In Visual Studio on the local computer, open the solution, and then set a breakpoint.
5. On the **Debug** menu, click **Attach to Process**.
6. In the **Qualifier** text box, type the name of the remote computer, and then click **Refresh**.
7. In the **Available Processes** list, find the W3wp.exe process, and then do the following, as applicable:
 - a. If the W3wp.exe process is not listed, make sure that the **Show processes from all users** and **Show processes in all sessions** check boxes are selected.
 - b. If the W3wp.exe process still is not listed, open a Web browser and navigate to the application's SharePoint site. After you navigate to the SharePoint site, return to Visual Studio. In the **Attach to**

- Process** dialog box, click the **Refresh** button to locate the W3wp.exe process.
- c. If you see multiple W3wp.exe processes listed and are not sure which one of them is running the code, click them all. Click **Attach**.

How to: Implement a SharePoint Workflow with ASP.NET Forms

SharePoint workflows use forms to handle interactions with users. SharePoint provides two approaches for implementing custom workflow association and initiation forms, modification forms, status forms, and task forms. One approach, which can be used with any Windows SharePoint Services 3.0 installation, uses ASP.NET forms. The other approach, which is only available with Microsoft Office SharePoint Server 2007, uses InfoPath forms. This topic discusses the first approach.

The following procedure is a high-level description of how to use Visual Studio to create ASP.NET custom workflow task forms. This procedure assumes that you can structure a solution in Visual Studio that can deploy a content type, a custom ASPX page, and a workflow. You can use Visual Studio extensions for Windows SharePoint Services and the Visual Studio workflow project template to help you develop and deploy these items.

To create ASP.NET workflow task forms

1. Using Visual Studio extensions for Windows SharePoint Services, create a custom content type for workflow tasks that is derived from the Workflow Task content type (0x010801). It must specify the URL of the form that displays the task. The following code is an example of a custom workflow task content type.

XML

```
<ContentType ID="0x01080100b3629edc2920465085800d1eb262d5ad"
  Name="My Workflow Task"
  Group="My Group"
  Description="My Workflow Task Content Type"
  Version="0">
  <FieldRefs/>
  <XmlDocuments>
    <XmlDocument
      NamespaceURI="http://schemas.microsoft.com/sharepoint/v3/contenttype/forms/url"
      <FormsUrl xmlns="http://schemas.microsoft.com/sharepoint/v3/contenttype/forms/url">
        <Display>_layouts/MyTaskForm.aspx</Display>
      </FormsUrl>
    </XmlDocument>
  </XmlDocuments>
</ContentType>
```

2. Associate the workflow with the custom workflow task content type. Use the **TaskListContentTypeID** attribute to do this. The following code shows an example.

XML

```
<Workflow
  Name="My Workflow"
  Description="Workflow that handles my business process."
  Id="d2eca905-b799-4ec2-aed6-783ee46337ad"
  CodeBesideClass="MyWorkflowAssembly.MyWorkflowClass"
  CodeBesideAssembly="MyWorkflowAssembly, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=9f4da00116c38ec5"
  TaskListContentTypeId="0x01080100b3629edc2920465085800d1eb262d5ad">
  <Categories/>
  <MetaData>
    <StatusPageUrl>_layouts/WrkStat.aspx</StatusPageUrl>
  </MetaData>
</Workflow>
```

3. In Visual Studio, create a new ASP.NET Web form that can be deployed to a SharePoint site with a feature. One method for creating this form is to copy one of the list forms that Visual Studio extensions for Windows SharePoint Services provides from the list definition project template. This file is already populated with content elements from the SharePoint master page. Copy the file and reference a custom code behind file that replaces the SharePoint code behind reference. To deploy the form to the _layouts virtual directory, do the following:
 - a. In your Visual Studio extensions for Windows SharePoint Services project, create a folder named Templates.
 - b. Inside the Templates folder, create a folder named Layouts.
 - c. Inside the Layouts folder, add the Web form. The Web solution package that Visual Studio extensions for Windows SharePoint Services creates will then deploy the form to the _layouts virtual directory.
4. Write the ASP.NET Web form code behind. It should derive from the **Microsoft.SharePoint.WebControls.LayoutsPageBase** class. The code should both process the workflow task and allow the user to perform the desired action for the task. The workflow task provides the data in a query string that can be used by the custom ASP.NET Web form. The following code shows an example of data that is provided by the query string.

C#

```
Request.QueryString["List"]; //the Guid Id of the workflow task list
Request.QueryString["ID"]; //the List Item Id of the workflow task
Request.QueryString["Source"]; //the URL of the referrer page
```

The following is more information about creating workflow forms:

- For more information about creating a custom content type, see [How to: Create a Custom Content Type with Event Receivers](#).

- For more information about workflow forms, see [Workflow Forms Overview](#) on MSDN.
- For more information about developing ASP.NET-related files using Visual Studio extensions for Windows SharePoint Services, see [How to: Perform ASP.NET-Related Development with Visual Studio](#).
- For more information about InfoPath workflow forms, see [InfoPath Forms for Workflows](#) on MSDN.
- For more information about SharePoint workflows, see [Developer Introduction to Workflows for Windows SharePoint Services 3.0 and SharePoint Server 2007](#) on MSDN.
- For more information about using Visual Studio extensions for Windows SharePoint Services, see the [Visual Studio extensions Windows SharePoint Services User Guide](#).

How to: Perform ASP.NET-Related Development with Visual Studio

Using Visual Studio extensions for Windows SharePoint Services (also referred to as "extensions" in this topic) simplifies SharePoint deployment. Two components that many SharePoint solutions deploy are custom ASPX pages and custom ASCX controls. The extensions' project templates do not provide the same support for developing these components that a Web Application project template does. This topic demonstrates how custom ASPX pages and custom ASCX controls can be developed in a Web Application project and then deployed using an extensions project.

To develop custom ASP.NET-related files with Visual Studio extensions for Windows SharePoint Services

1. In Visual Studio, create a new solution with one of the Visual Studio extensions for Windows SharePoint Services project templates.
2. In the new solution, create a new project with the ASP.NET Web Application project template. In this new project, delete the Default.aspx file, the App_Data folder, and the Web.config file.
3. Add a new ASPX or ASCX project item to the Web Application project.
4. Add the new file to the Visual Studio extensions for Windows SharePoint Services project as a link to the existing item. To link a file from another project, right-click the folder, click **Add Existing Item**, navigate to the file, and then click **Add As Link** in the **Add** drop down list. Do one of the following:
 - For an ASPX file, create a new Module project item and place the linked ASPX file in the new module. Edit the Module.xml file to include a **File** element. Update the **Path** and **Url** attributes to the file name of the .aspx file. This deploys the page when the SharePoint solution is deployed.
 - For an ASCX file, create a new Template project item, add a folder named ControlTemplates, add a subfolder so you can distinguish your files when they are installed in SharePoint, and add the linked .ascx file to that subfolder.
5. Add ASP.NET-related assembly references to the Visual Studio extensions for Windows SharePoint Services project. This allows the code-behind to compile.
6. In the Configuration Manager of the solution, clear the **Build for the Web Application project** check box.
7. Open the linked code-behind file from the Visual Studio extensions for Windows SharePoint Services project and change the namespace to match the namespace of the project.

Open the markup of the file from the Web Application project, remove the **CodeBehind** attribute from the markup, and then edit the **Inherits** attribute to match the assembly information for the Visual Studio extensions for Windows SharePoint Services project.

After you configure the solution items, use the following two methods to edit the files' markup and code-behind:

- To edit the markup of the ASP.NET-related files, use the Web Application project to open the markup file. This allows you to take advantage of the automatic generation of the designer's code-behind file.
- To edit the code behind of the ASP.NET-related files, use the Visual Studio extensions for Windows SharePoint Services project to open the code-behind file. This allows you to use IntelliSense for any code references that you have in these files.

After you build the extensions project, follow the extensions' deployment process to package the solution and deploy it to SharePoint. To learn more about how to use the extensions, download [Visual Studio 2008 extensions for Windows SharePoint Services 3.0, v1.3.](#)

How to: Programmatically Configure a WCF Endpoint

This topic includes procedures that describe how developers can create a SharePoint Web Part that uses data from a Windows Communication Foundation (WCF) Web service. The Web Part programmatically configures the WCF clients.

The procedures use the Partner Portal application's Contoso Pricing and Product Catalog services. They assume you have the **Contoso.LOB.Services** project installed.

Note:

Installing the Partner Portal application also installs the **Contoso.LOB.Services** project.

To create a new Web Part project

1. In Visual Studio, point to **New** on the **File** menu, and then click **Project**. Under **Project types**, click **SharePoint**. Under **Templates**, click **Web Part**. Use "MyWcfWebPart" as the project name. Click **OK**.
2. In the **Select Trust Level** dialog box, select **Full Trust (Deploy to GAC)**, and then click **OK**.

To add proxy classes for the Web services to the Web part project

1. Open a Visual Studio 2008 command prompt.
2. Use the **cd** command to change the working directory to the Web Part project directory.
3. Run the following command:
svcutil.exe https://<computername>:8686/contoso.lob.services/pricing.svc/noconfig /out:pricingproxy.cs
4. Include the generated proxy class into the project. To do this, do the following:
 - a. In Solution Explorer, right-click the project name, point to **Add**, and then click **Existing Item**.
 - b. Browse to the Pricingproxy.cs file, and then click it.
 - c. Click **OK**.
5. Run the following command:
svcutil.exe http://<machinename>:8585/contoso.lob.services/productcatalog.svc/noconfig /out:productcatalogproxy.cs
6. Include the generated proxy class into the project. To do this, do the following:
 - a. In Solution Explorer, right-click the project name, point to **Add**, and then click **Existing Item**.
 - b. Browse to the Productcatalogproxy.cs file, and then click it.
 - c. Click **OK**.
7. In Solution Explorer, right-click **References**, click **Add Reference**, and then add the following references:
 - a. System.ServiceModel
 - b. System.Runtime.Serialization

To implement the Web Part class

1. Add the following method to the **WebPart1** class by pasting it into the WebPart1.cs file.

C#

```
private PricingClient ConfigurePricingProxy()
{
    SecurityBindingElement securityElement =
    SecurityBindingElement.CreateUserNameOverTransportBindingElement();
    HttpsTransportBindingElement httpsTransport = new HttpsTransportBindingElement();
    httpsTransport.AuthenticationScheme = System.Net.AuthenticationSchemes.Ntlm;

    CustomBinding binding = new CustomBinding(securityElement, httpsTransport);
    binding.Name = "pricingBinding";

    // Replace <machinename> with the name of the development server.
    EndpointAddress remoteAddress = new
    EndpointAddress("https://<machinename>:8686/Contoso.LOB.Services/pricing.svc");

    PricingClient client = new PricingClient(binding, remoteAddress);
    client.ClientCredentials.UserName.UserName = "Partner1";
    using (HostingEnvironment.Impersonate())
    {
        client.ClientCredentials.Windows.ClientCredential =
        CredentialCache.DefaultNetworkCredentials;
    }

    return client;
}
```

2. Add the following method to the **WebPart1** class by pasting it into the WebPart1.cs file.

C#

```
private ProductCatalogClient ConfigureProductCatalogProxy()
{
    BasicHttpBinding binding = new
    BasicHttpBinding(BasicHttpSecurityMode.TransportCredentialOnly);
    binding.Security.Transport.ClientCredentialType = HttpClientCredentialType.Windows;

    // Replace <machinename> with the name of the development server.
```

```

    EndpointAddress remoteAddress = new
EndpointAddress("http://<machinename>:8585/Contoso.LOB.Services/productcatalog.svc");

    ProductCatalogClient client = new ProductCatalogClient(binding, remoteAddress);
    using (HostingEnvironment.Impersonate())
    {
        client.ClientCredentials.Windows.ClientCredential =
CredentialCache.DefaultNetworkCredentials;
    }

    return client;
}

```

3. Copy the following code into the **CreateChildControls** method of the **WebPart1** class.

```

C#
base.CreateChildControls();

const string PRODUCT_SKU = "1000000000";

Price price;
using (PricingClient pricingClient = ConfigurePricingProxy())
{
    price = pricingClient.GetPriceBySku(PRODUCT_SKU);
}

Product product;
using (ProductCatalogClient productCatalogClient = ConfigureProductCatalogProxy())
{
    product = productCatalogClient.GetProductBySku(PRODUCT_SKU);
}

LiteralControl content = new LiteralControl();
content.Text = string.Format(CultureInfo.CurrentCulture, "Product: {0}<br>Price: {1}",
product.Name, price.Value);

this.Controls.Add(content);

```

4. Highlight the unresolved class names, and then press CTRL+PERIOD. Select the namespace from the list. This automatically adds the **using** statements.
5. Update the Web Part definition file. To do this, do the following:
 - a. Open the WebPart1.webpart file.
 - b. Update the **Title** property to My Pricing and Product Web Part.

To test the Web Part

1. Specify a start debug URL. To do this, do the following:
 - a. Right-click the project file, and then click **Properties**.
 - b. Click the **Debug** tab. Under **Start Action**, set the **Start browser with URL** option to the address of a valid SharePoint site.
 - c. Press F5.
2. Add the Web Part to a page. To do this, do the following:
 - a. In SharePoint, click **Edit Page** on the **Site Actions** menu.
 - b. In any Web Part zone, click the **Add a Web Part** button.
 - c. Under **Miscellaneous**, select **My Product** and **Pricing Web Part**, and then click **Add**.
 - d. Verify that you see Blood Pressure Kit with a price of 319.99.

How to: Use the Hierarchical Configuration to Store an Endpoint Address

One of the drawbacks of the approach described in [How to: Programmatically Configure a WCF Endpoint](#) is that the endpoint address is hard coded into the Web Part. For more flexibility, you can store the WCF service's endpoint address as a configuration setting. The following procedure demonstrates how to store and retrieve the endpoint addresses of the pricing and product catalog services with the Web part that is implemented in [How to: Programmatically Configure a WCF Endpoint](#).

Note:

This How-to topic is a continuation of the procedures in [How to: Programmatically Configure a WCF Endpoint](#). You must complete the procedures in that topic before you proceed.

To create a feature receiver class to store the endpoint addresses

1. Right-click the project, point to **Add**, and then click **New Item**. In the Categories pane, under **C#**, click **Code**. In the Templates pane, click **Class**. Use "MyWcfWebPartFeatureReceiver" as the class name. Click **Add**.
2. Copy and paste the following code into the new class file.

C#

```
using System;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using System.Text;
using Microsoft.SharePoint;
using Microsoft.Practices.SPG.Common.Configuration;
using Microsoft.Practices.SPG.Common.ServiceLocation;

namespace MyWcfWebPart
{
    public class MyWcfWebPartFeatureReceiver : SPFeatureReceiver
    {
        public override void FeatureActivated(SPFeatureReceiverProperties properties)
        {
            SPSite site = properties.Feature.Parent as SPSite;

            IConfigManager configMgr =
                SharePointServiceLocator.Current.GetInstance<IConfigManager>();

            // Replace <machinename> with the name of the development server.
            configMgr.SetInPropertyBag("PricingEndPointAddress",
                "https://<machinename>:8686/Contoso.LOB.Services/pricing.svc", site);
            configMgr.SetInPropertyBag("ProductCatalogEndPointAddress",
                "http://<machinename>:8585/Contoso.LOB.Services/productcatalog.svc", site);
        }

        public override void FeatureDeactivating(SPFeatureReceiverProperties properties)
        {
        }

        public override void FeatureInstalled(SPFeatureReceiverProperties properties)
        {
        }

        public override void FeatureUninstalling(SPFeatureReceiverProperties properties)
        {
        }
    }
}
```

3. Replace the <machinename> literal with name of the development server.
4. Add references to the **Microsoft.Practices.ServiceLocation** and **Microsoft.Practices.SPG.Common** assemblies.
5. Add a **GUID** attribute with a feature ID. To do this, do the following:
 - a. Open the WSP View.
 - b. Open the Feature.xml file for WebPart1.
 - c. Copy the GUID value of the **Feature Id** attribute.
 - d. Open the new class file. Immediately before the class declaration, add a **GUID** attribute that uses the GUID from step 5c. The following code is an example.

C#

```
[Guid("ab8d624a-f2ad-4c62-bb74-0cf60cad9096")]
public class MyWcfWebPartFeatureReceiver : SPFeatureReceiver
```

6. Modify the **ConfigurePricingProxy** method in your WebPart1 class. To do this, do the following:
 - a. Locate the following line of code.

C#

```
EndpointAddress remoteAddress = new  
EndpointAddress("https://<machinename>:8686/Contoso.LOB.Services/pricing.svc");
```

- b. Delete the preceding code, and then replace it with the following code.

C#

```
IConfigManager configMgr = SharePointServiceLocator.Current.GetInstance<IConfigManager>();  
    EndpointAddress remoteAddress = new  
EndpointAddress(configMgr.GetFromPropertyBag<string>("PricingEndPointAddress",  
SPContext.Current.Site));
```

7. Modify the **ConfigureProductCatalogProxy** method. To do this, do the following:

- a. Locate the following line of code.

C#

```
EndpointAddress remoteAddress = new  
EndpointAddress("http://<machinename>:8585/Contoso.LOB.Services/productcatalog.svc");
```

- b. Delete the preceding code, and then replace it with the following code.

C#

```
IConfigManager configMgr = SharePointServiceLocator.Current.GetInstance<IConfigManager>();  
    EndpointAddress remoteAddress = new  
EndpointAddress(configMgr.GetFromPropertyBag<string>("ProductCatalogEndPointAddress",  
SPContext.Current.Site));
```

8. Press F5 to deploy and debug the project. It may be necessary to reset the Information Internet Services (IIS) or to recycle the application pool.

Managing the Application Life Cycle

SharePoint offers many ways to create applications and includes customization features that allow site administrators, designers, and end users to change the look, feel, and structure of those applications. This flexibility poses particular challenges during the application's life cycle for developers, solution architects, and IT administrators. For example, the components that developers create are maintained in a source control system, but customizations are stored in a content database. The following are some content-related components:

- Customizations that are made by site designers and site administrators who use SharePoint Designer and the SharePoint Web user interface (UI)
- Text and images that are created by content authors
- Components that are personalized by end-users

This topic discusses ways to deploy and upgrade SharePoint applications. The guidance uses a topology that is recommended for small-sized to medium-sized companies. Some typical scenarios in a SharePoint application's life cycle are the following:

- Deploying a new application
- Adding functionality that does not change the application's current functionality
- Adding functionality that changes the application's current functionality

All the recommended approaches in this topic take into account that different artifacts are stored in different locations. For more information, see [Deployment Scenarios](#).

Note:

This guidance uses the terms *SharePoint solution* to refer to the Web solution package (WSP) that is specific to installation and deployment. The term *SharePoint Web application* refers to a load-balanced Web application that is based on Internet Information Services (IIS). The term *application* is used as a general term to describe a set of functionality that solves a particular business problem. For example, the Training Management application demonstrates how Contoso Pharmaceuticals uses SharePoint to manage its training courses and enrollments. In many cases, an application represents a much smaller capability, such as a Web Part that can be used on many sites.

Often, more than one SharePoint solution deploys an application's functionality. For example, the Training Management application includes two SharePoint solutions to allow existing workflow instances to complete after the application upgrade is applied.

This section includes the following topics:

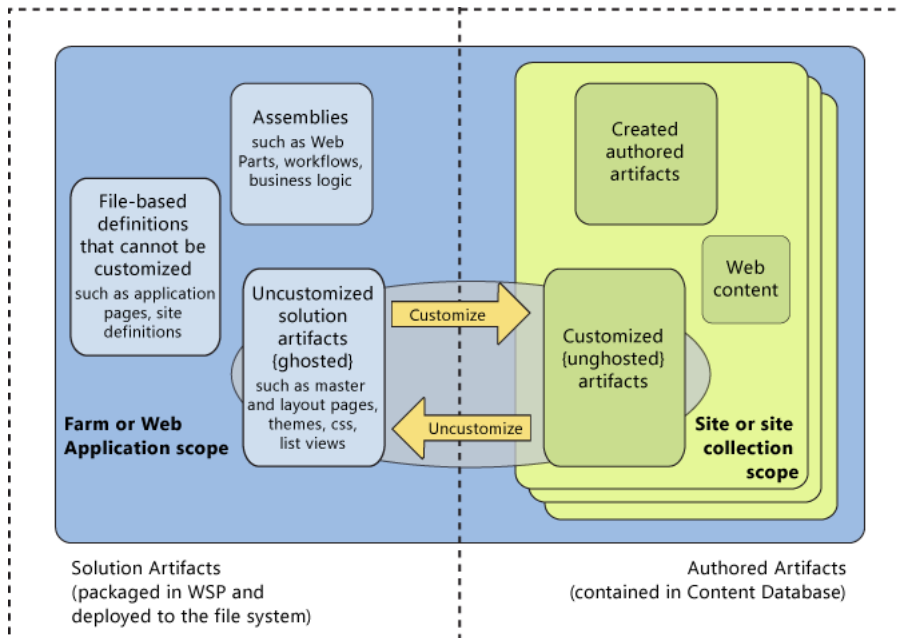
- [Defining Artifact Terminology](#). This topic explains the terminology that defines SharePoint artifacts.
- [The SharePoint Application Life Cycle](#). This topic is an overview of the stages that comprise a SharePoint application's life cycle.

Defining Artifact Terminology

To understand the relationships between the different artifacts that comprise a Windows SharePoint application, it is important to be clear about terminology. The following definitions are based on those from the online book, [Design and build sites for Office SharePoint Server 2007](#), on TechNet. They have been tailored to suit this guidance.

The following illustration shows the relationships between the artifacts that comprise a SharePoint application and where each artifact is stored.

SharePoint artifacts



The artifacts that comprise a SharePoint application include the following:

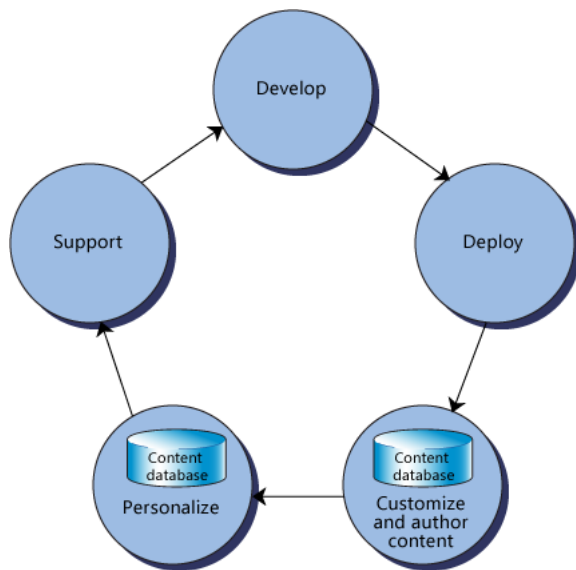
- **Solution artifacts.** Solution artifacts are created by developers or designers as files and are packaged into a WSP. After the solution deploys, its artifacts apply across all sites in the deployed scope. The deployed scope can either be in a SharePoint farm or a SharePoint Web application, depending on the option that is chosen at deployment. A subset of the solution artifacts is customizable at the site or site collection level using the browser or SharePoint Designer. An *uncustomized* solution artifact is often referred to as *ghosted*, and a *customized* solution artifact is often referred to as *unghosted*. Although the terms "ghosted" and "unghosted" have been used in SharePoint documentation and blogs, this documentation uses the terms customized and uncustomized because the terms are clearer and consistent with current versions of the SharePoint documentation.
- **Authored artifacts.** Authored artifacts are either customizations of solution artifacts or new artifacts that are created using SharePoint Designer or the Web browser after a solution is deployed. Typically, authored artifacts are created by designers or site administrators. Authored artifacts affect the look, feel, and structure of an application. Authored artifacts reside in the content database. Customizations are scoped at the site collection or site level, depending on the type of authored artifact.
- **Web content.** Web content is content, such as text and images, that is created by content authors. Typically, Web content is created using an application such as Microsoft Word or with the browser. It is either added directly to the site or deployed through a Web content management system.

When deciding on a deployment strategy, it is important to understand the scope of the different artifacts, where they are stored, and the artifacts that can be customized.

The SharePoint Application Life Cycle

The following figure illustrates the stages in the life cycle of a typical SharePoint application.

SharePoint application life cycle



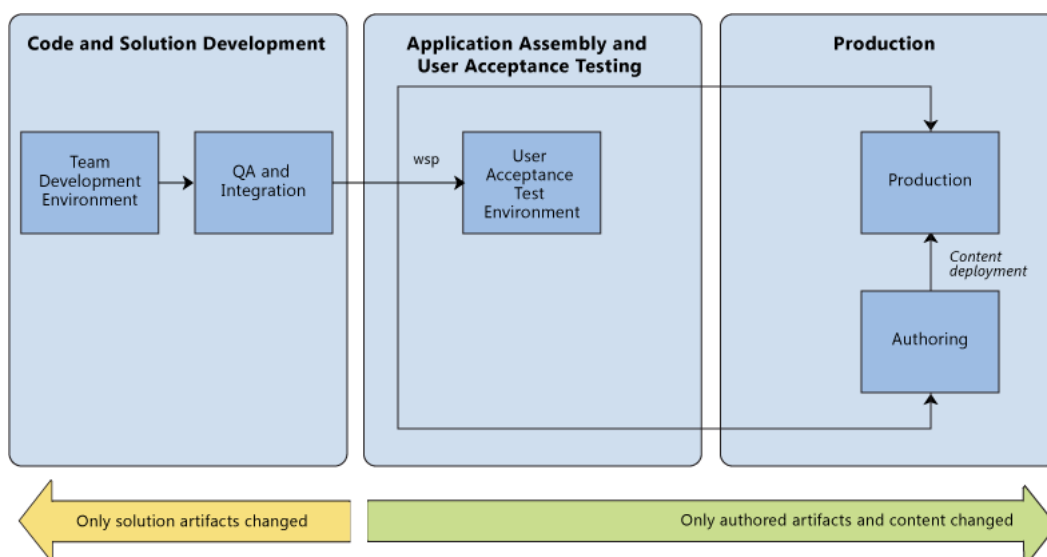
When you plan your deployment and upgrading strategy, consider each stage of this cycle. For example, you need to understand how the application will be used, how you will develop the application, how you will package it for deployment, and how you will manage data that is maintained in different environments. The following are some of the issues to consider:

- Will you ever need to upgrade this application?
- Will you only need to add functionality without changing the existing functionality?
- Will you package this application for broad distribution over multiple application versions, for example as an independent software vendor (ISV), or across the enterprise?
- Do you control all sites where the application is deployed?
- Do you need to preserve customizations from version to version?
- Will you need to create content after development? Will you need to coordinate publishing this content with upgrading application artifacts?
- What is the governance model? For example, consider the following:
 - Do you have sensitive data that developers should not access?
 - Can you make changes in production as part of the development process?

A Typical Development, Testing, and Deployment Topology

The following illustration shows a typical topology for developing, testing, and deploying a SharePoint application.

A typical topology



The following are the environments and servers that comprise this topology:

- **Code and solution development environment.** The following environments are used to develop and test the code:
 - **The team development environment.** The development and test teams use the team development environment to develop, build, unit test, and validate the application before deploying it to shared environments such as the quality assurance (QA) and integration test environment. For more information, see [Team Development Overview](#).
 - **QA and integration environment.** The test team and product owners use the QA and integration environment to perform functional testing, performance testing, and integration testing. This environment can contain more than one server, and it can also contain multiple configurations if the actual destination environment is not known. Several applications that are at different stages of completion can be simultaneously tested here. Because this environment is more representative of a production environment, developers may also need it to debug integration problems.
- **Application assemble and user acceptance testing (UAT) environment.** This environment resembles the production environment as closely as possible. It is used to test the deployment of an application before the actual deployment. Only production-ready applications should be deployed to a UAT environment. The UAT environment can be eliminated for smaller applications with lower governance requirements. This environment is sometimes referred to as a staging environment in older SharePoint guidance. However, in SharePoint, a staging environment can also mean a production site where content receives a final review before it is released. To avoid confusion, the "staging" term is no longer used to mean the UAT environment. WSPs are the only mechanisms that can move artifacts from the QA and development environments into the UAT environment.
- **Production environment.** The production environment is where released applications operate. Typically, if a company uses SharePoint's Web content management features, there is also an authoring environment within the production environment to create, review, and approve content. Once it is approved, the content deployment process moves the content to the production servers.

As shown in the earlier illustration, solution artifacts and authored artifacts should not be managed in the same environments. Solution artifacts should never originate in the UAT or production environment and then be sent to the code and solution development environment. Authored artifacts should never originate in the code and solution development environment and then be sent to the other two environments.

It is reasonable to have authored artifacts that are used in the code and solution development environment to perform tests. However, you should make sure that your solution has no dependencies that require that authored artifacts exist in that environment.

The following are the main recommendations for developed assets that are packaged as WSPs:

- Always convert authored artifacts to solution artifacts (these are file-based assets) to satisfy dependencies in the code. Code-based functionality should always be fully reproduced when you deploy a WSP.
- Preserve and version code-based assets and dependencies as solution artifacts.
- To maintain a consistent look for the site, consider converting production customizations (these are authored artifacts) to solution artifacts when you perform an upgrade. On subsequent releases, incorporate changes that have wide applicability into the code.
- Only change solution artifacts in the code and solution development environment.
- Only change authored artifacts and content that are intended for the production environment in the UAT and production environments.

Content deployment can be used with any application but it is most often used with a publishing application. Content deployment synchronizes all information from the source site, which is typically in an authoring farm, to the destination site, which is typically in a destination farm. SharePoint expects the destination site to be unchanged between content deployment jobs. This is when the content on the destination site is synchronized

with the content on the authoring site. If the two are not synchronized, the next content deployment job fails.

The implication is that the destination site should be read-only. No one should be able to change anything that is deployed from the authoring farm's content database. A read-only site can contain components that write to other systems. For example, you can use SharePoint to publish a product catalog. This application might have Web Parts that write data to a business system when a customer purchases an item. This is still a read-only site because no SharePoint data changes.

You must install the SharePoint solutions on the source site before you deploy content to a production site. The content deployment process activates SharePoint features for you on the destination site. For more information, see [Understanding Publishing and Content Deployment](#).

SharePoint has site templates that allow system administrators and end users to easily create new sites based on a predefined format. Site templates are convenient but are also difficult to upgrade. This can cause many maintenance issues. Common uses of site templates include the following:

- **For applications with limited life spans.** In terms of site templates, applications that are used for a limited amount of time have advantages. This is because you may never need to change the site instances. Instead, you can allow sites that are based on a previous template to expire. Then, you can apply the updated template (this is actually a new template) only to new instances of the site. A good example of this situation is an incident site. The incident has a limited lifetime—after the incident is resolved, the site is no longer needed.
- **For applications that will soon be updated.** Site templates are sometimes used to provide basic functionality for an application that will be updated and maintained in production within a very short period of time. The site template helps you to quickly create a basic site, but you must be certain that you will not update any of the sites instances.

Deployment Scenarios

This topic discusses the following three deployment scenarios:

- [Deploying a New Application](#)
- [Adding New Functionality to an Existing Application](#)
- [Updating Functionality for an Existing Application](#)

The first two scenarios are relatively straightforward because existing customizations and functionality are not affected by the deployment. The second scenario also discusses feature stapling. If the original application uses a site definition for site creation, you can optionally staple the new SharePoint feature to the site definition. For more information, see [Stapling Features to Site Definitions](#).

The third case, where the deployment affects the existing application, has more challenges. All the scenarios assume the topology that is discussed [Managing the Application Life Cycle](#).

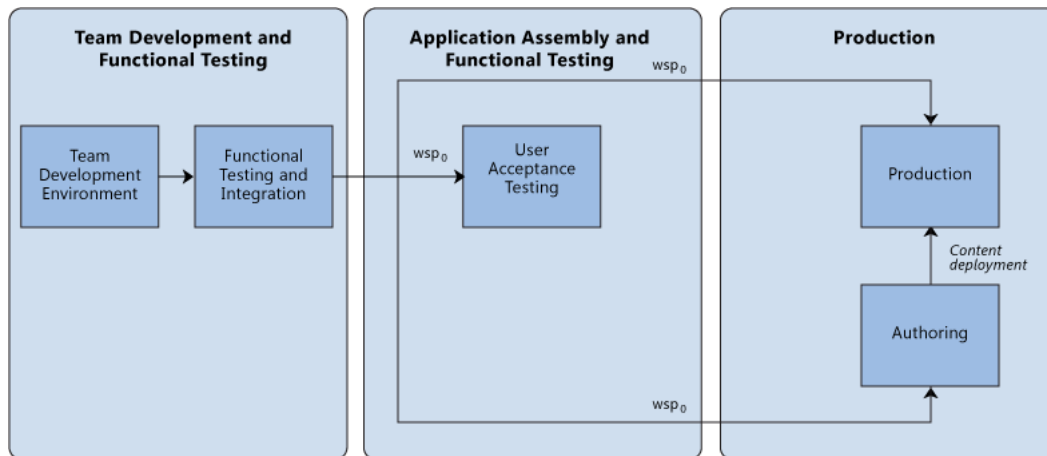
For an example of how to deploy a new application, see [An Example of Deploying a New Application](#). This topic explains how to deploy the Training Management application.

Finally, this topic includes [Upgrading an Application](#). This uses the Training Management application as an example.

Deploying a New Application

The following illustration shows the path that the SharePoint solution package (WSP) follows when you deploy a new solution. The path begins in the development environment and ends in the production environment. Although this configuration is a typical approach, there are many possible variations that are equally valid. In the illustration, the SharePoint solution is named `wsp0` and contains all the solution artifacts for the application.

Deploying a new application



This scenario assumes the following:

- This is the initial deployment of a new application. All the required solution artifacts are defined in one or more SharePoint solutions.
- All the SharePoint solution artifacts are version-controlled with a source control system.
- The application can ship independently of any authored artifacts.
- Because this is an initial deployment, there are no existing records in the SharePoint content database that can conflict with the elements that are defined by the solution artifacts.

The Functional Testing and Integration Environment

The functional testing and integration environment (the test environment) performs manual functional testing, automated functional testing, system testing, security testing, and performance testing. Generally, some level of validation testing should run in the development environment before deploying the application to the test environment. The patterns & practices SharePoint Guidance team runs continuous integration and build verification tests (BVT) before it deploys the SharePoint solution to the test environment. The test environment can also be used to debug integration issues that developers cannot locally reproduce.

The test team controls when the SharePoint solutions are deployed to the test environment. Often, tests that are conducted in the test environment are referred to as *black box* tests. This is because no knowledge of the application's internal implementation is necessary to test the application's functionality. The environment is relatively stable, but it may have several applications deployed that are in various stages of development. Therefore, you should expect to encounter and resolve conflicts between applications in the test environment in addition to any problems that are not discovered in the team development environment.

Depending on the criticality and scale of the application, the test team may conduct performance tests to identify and resolve bottlenecks. Initial user acceptance testing is often also conducted in the test environment. After the test team is satisfied that the quality of the application meets production requirements, the SharePoint solutions are deployed to the User Acceptance Test (UAT) environment. For more information about how to set up the development and testing process, see [Team Development Overview](#). For an example of how to deploy a new application, see [An Example of Deploying a New Application](#).

The User Acceptance Test Environment

The User Acceptance Test (UAT) environment tests the deployment of the production-ready application in an environment that closely resembles the production environment. In older guidance, the UAT environment is named the staging environment. The name was changed to UAT because, in SharePoint, a staging server deploys content that still requires a final review to the production environment. The purpose of the UAT environment is to identify any potential deployment issues and to allow content to be reviewed by business owners. If the application passes these tests, the business owners sign off on the application. Although the UAT environment is optional for smaller applications where deployment failures are not critical, it is still recommended.

Only production-ready applications are deployed to a UAT environment. This environment represents the destination production environment as closely as possible from the perspective of topology (for example, the server farm and database structure) and components (for example, the inclusion of the Microsoft Active Directory directory service).

After the code is in the UAT environment, it is often labeled or versioned as final. This ensures the reproducibility of the final build. In more complex environments, the application may also be branched into a new fork of the

revision control system. A recommended practice is to use a script to deploy and activate the application when you are deploying it to the UAT and production environments. This minimizes the opportunity for human error. The Partner Portal application includes installation scripts that serve as examples.

You should manually configure the SharePoint instance to duplicate the production configuration. Where governance permits, production data should be copied back to the UAT environment to create an environment that is as realistic as possible. Because the UAT environment does not need to be broadly accessible, you can secure the environment just as you would the production environment.

The Microsoft TechNet article, [Move content databases between instances of SQL Server \(Office SharePoint Server 2007\)](#), describes in detail how to move content databases between instances of SQL Server 2005. Use these instructions to copy the production SharePoint content database(s) to the UAT environment. Make sure to reattach the production content database(s) after the .mdf and .ldf files are in the UAT environment. The procedure in the article copies the entire content database that is associated with a site collection. If you have multiple site collections, it is advisable to copy all related content databases to the UAT environment.

If you want to copy only the contents of a specific SharePoint Web site, see [Using Stsadm.exe to Migrate Site Data](#) on MSDN. This may be useful if you have capacity limitations and it is not feasible to copy an entire content database.

For an example of how to deploy a solution to a UAT environment, see [An Example of Deploying a New Application](#).

The Production Environment

After the UAT tests are complete, the application can be deployed to the production environment. Because the UAT phase emulates the production deployment, the deployment steps are identical, aside from any required name changes. For an example of how to deploy an application to a production environment, see [An Example of Deploying a New Application](#). For publishing scenarios, there will often be an authoring farm for creating and approving content. This environment is also considered a production environment. Typically, solutions are applied to the authoring environment before or at the same time as the production environment. You may need to stagger the deployment if the changes impact the content in the production environment. In this case, you may need to perform the following steps:

1. Apply the solution to the authoring farm.
2. Make relevant updates to the production content in the authoring farm.
3. Apply the solution to the production farm.
4. Deploy the content to the production farm.

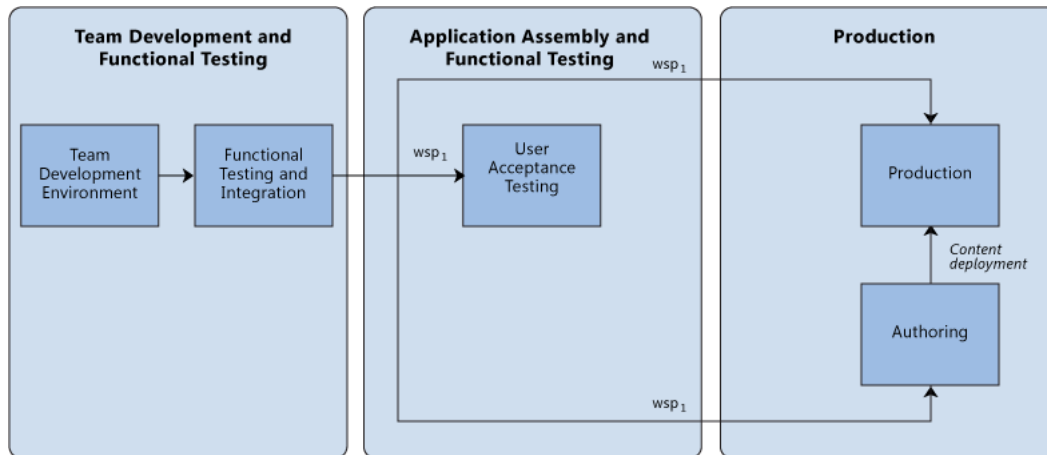
Adding New Functionality to an Existing Application

You can add functionality to an existing application by delivering a new feature that augments the original application. To do this, package that functionality into one or more new WSPs. A new SharePoint solution is defined by a new solution ID.

Before you use this approach, you should first ensure that you meet the decision criteria to create a new SharePoint feature. Indiscriminately using SharePoint features as a way to upgrade an application can lead to situations that are difficult to manage. You should upgrade the original SharePoint feature if the new functionality is an extension of the existing functionality instead of an independent function. For more information about how to decide whether you should use a SharePoint feature, see [Organizing Features](#).

The following illustration shows the path that the SharePoint solution (wsp1) follows from the development and functional testing environment to the production environment. The new functionality does not affect any of the existing artifacts. The new solution artifacts for the added functionality are packaged into wsp1.

Adding new functionality



The deployment of the new SharePoint solution is the same as in the previous case, [Deploying a New Application](#).

To add functionality to an existing application, develop the new functionality as a new SharePoint feature and package the new SharePoint feature into a new SharePoint solution. Site administrators manually activate the new SharePoint feature on the applicable sites after it is deployed. Often, this activation is scripted. The actual development and deployment of the SharePoint feature follows the same process as a new application. When a new SharePoint feature depends on a previous SharePoint feature, you should use SharePoint feature activation dependencies to guarantee that the dependencies are present. Also, if the original application uses a site definition for site creation, you can optionally staple the new SharePoint feature to the site definition. This ensures that subsequently created sites that use the site definition contain the new SharePoint feature. For more information about feature stapling, see [Stapling Features to Site Definitions](#). For more information about feature activation dependencies, see [Activation Dependencies and Scope](#) on MSDN.

Stapling Features to Site Definitions

Stapling SharePoint features to site definitions, which relies on the **FeatureSiteTemplateAssociation** element, allows developers to attach features to both standard and custom site definitions without directly modifying them. Every new site that is based on a site definition with a stapled feature is created with that feature activated. Feature stapling is particularly useful when you want to add functionality to standard site definitions because editing standard site definition files is not a recommended practice.

The Training Management application uses feature stapling to associate the Contoso theme with the CONTOSOTRAINING site definition. The following code from the Contoso.RI.UpdateTheme solution uses feature stapling for the Contoso theme feature. This feature deploys and applies the Contoso theme to a site.

XML

```
<Elements Id="ceaf7e5c-0d93-4fae-9426-b7311a8f574a"
xmlns="http://schemas.microsoft.com/sharepoint/">
  <FeatureSiteTemplateAssociation Id="1d7f7c32-9a58-4127-ac0b-d2b1742e9f4e"
TemplateName="CONTOSOTRAINING#0"/>
</Elements>
```

The **Id** attribute for the **FeatureSiteTemplateAssociation** element references the ContosoTheme feature. The **TemplateName** attributes reference site definitions. The site definitions are referenced in the Webtemp.xml file that is located in the %CommonFiles%\Microsoft shared\Web Server Extensions\12\TEMPLATE\1033\XML folder.

The following code is from the Feature.xml file of the Contoso theme feature stapler. The scope of the feature is **Farm**. This means that, when it is activated, the feature stapler staples the Contoso theme feature to the site definitions for any new sites in the SharePoint server farm.

XML

```
<Feature Id="1c67a1c7-6f25-4fba-9dd0-a36197a9f637" Title="Contoso Theme Feature Stapler"
Scope="Farm" Version="1.0.0.0" Hidden="FALSE" DefaultResourceFile="core"
xmlns="http://schemas.microsoft.com/sharepoint/">
  <ElementManifests>
    <ElementManifest Location="ApplyTheme\featureElement.xml" />
  </ElementManifests>
</Feature>
```

If the feature stapler is deactivated, the existing sites still have the Contoso theme feature activated, but no new sites will have the Contoso theme feature activated.

For more information about feature stapling, see the following resources:

- [Feature/Site Template Association](#) on MSDN
- [Feature Stapling](#) on MSDN
- [Feature Stapling in WSS V3](#) on Chris Johnson's blog.

For more information about site definitions, see [Site Definitions and Configurations](#) on MSDN. For more information about features, see [SharePoint Feature](#) on MSDN.

Updating Functionality for an Existing Application

After an application is deployed and operational, you may need to update the functionality of the deployed application. The following are examples of this scenario:

- You need to update a content type that is currently used.
- You need to update a Web Part to a new version.
- You need to change the logic of a workflow.
- You need to add or remove a list column on which code logic depends.

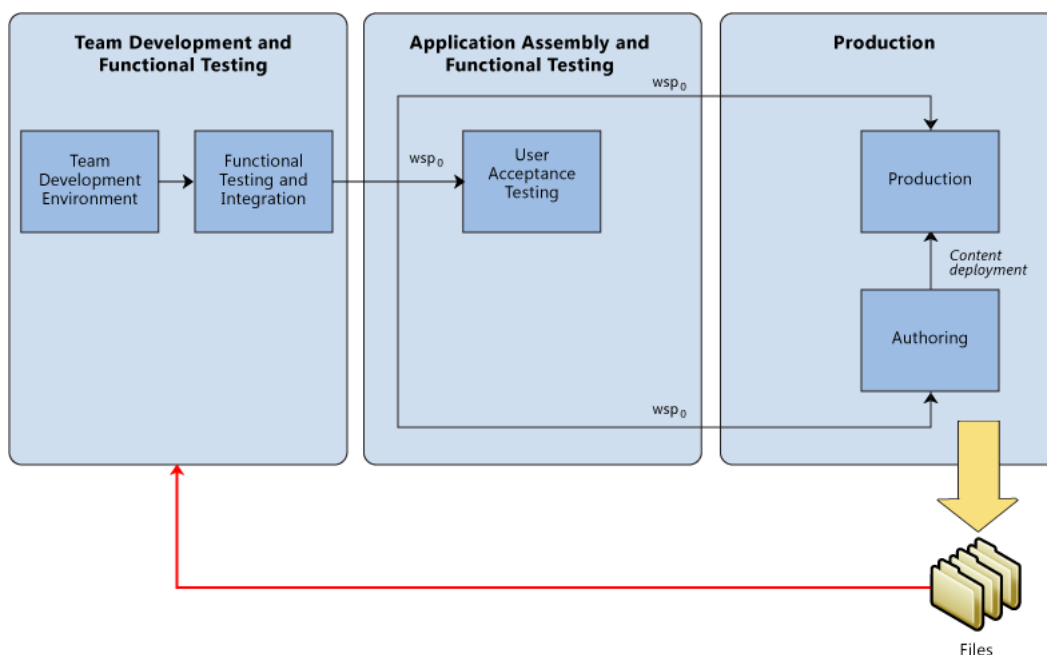
Deployment strategies for this scenario depend on your control of the already deployed application and the complexity of the updates. You can upgrade an application by incorporating the changes into the existing Windows SharePoint Services solution(s) for the application. This requires that you use either the upgrade solution or the retract-and-redeploy approach that is described in the next section of this topic. The alternative is to create the application as a new SharePoint solution with a new solution ID and migrate the previous application version instances to the new version. The following are the possible approaches:

- **Use the upgrade solution command to upgrade the application.** In this approach, the SharePoint command for a solution upgrade is used to deploy updates to your application.
- **Use the retract-and-redeploy approach on the SharePoint solution to upgrade the application.** In this approach, the SharePoint solution is retracted, removed, added, and deployed to upgrade the application.
- **Create a new SharePoint solution and migrate.** In this approach, the application is deployed as a new SharePoint Solution, and the application logic works side-by-side with the previous version. The data is migrated from the previous version instance to the new version instance to upgrade the application.

Implications of the Upgrade Solution Approach and the Retract-and-Redeploy Approach

The following illustration shows the path that the SharePoint solution follows from the development and functional testing environment to the production environment if you use either the upgrade solution command or the retract-and-redeploy approach. The solution artifacts that are contained in the SharePoint solution are named `wsp0` in the illustration. Because the application is already in operation, developers must consider how to approach customizations that have occurred in production through SharePoint Designer or the Web browser.

Adding functionality that affects existing functionality



If you understand how the application is deployed and the impact of incorporating or reverting customizations in the production environment, you can do the following:

- Add the new or updated logic to the application.
- Repackage the application using the same SharePoint solution (the SharePoint solution is defined by the solution ID).
- Deliver the application back to production.

When you upgrade the application, the updates are repackaged with the ID of the original SharePoint solution. The original files are overlaid and assemblies are replaced with the updated version. For updated assemblies deployed to the global assembly cache, the previous version is removed and the new version is added. Frequently, you must add application logic to update or change certain artifacts, such as when you replace a Web Part page or update a content type.

You must convert structural changes to the production site from authored artifacts back to solution artifacts if there are related logic changes in the code. Examples of these structural changes include adding or deleting lists and modifying content types. Because solution artifacts are scoped as either a SharePoint Web application or a SharePoint farm, although authored artifacts are either site collection or site scoped, converting changes from authored artifacts to solution artifacts impacts a broader range of sites than the original customization. If a solution artifact is customized differently on multiple sites and you want to keep elements of both customizations, you must reconcile and merge the differences.

Your application should not depend on end users manually applying customizations for structural changes that must be present for the new functionality to work. This creates an unnecessary dependency or coupling between your code and actions outside of source and change control.

Converting non-structural changes such as customized cascading style sheets to solution artifacts is optional. An example of when this is appropriate is to apply the appearance and behavior (look and feel) of a particular site to all sites. Another example is for performance improvements. There have been significant performance improvements to the management of customizations in the most recent version of SharePoint. These improvements have reduced the performance implications of customizations and the need to move customizations back to file-based definitions.

When updating a site or solution, you need to decide whether to revert customizations (this is sometimes referred to as *re-ghosting*). In cases where there is logic in place that you depend on, there is little choice but to revert the customizations to solution artifacts to ensure that your logic executes correctly. In other cases, you may not have logical dependencies but want to promote consistency with new updates. Reverting customizations for an artifact does not lose any personalization data. Often, organizations will have policies to move all customizations back into solution artifacts when a new version is released. For example, you may decide that you want the flexibility to build new page layouts for publishing pages between releases, but at each updated version, all created page layouts are brought back to solution artifacts for the new release. This balances between the need to be agile with content and keeping close control over reproducibility and application versioning.

In some cases, you may need to take more drastic measures that cause all customizations and personalization data to be lost. Examples include deleting a file when you deactivate a SharePoint feature or moving a file. Where possible, you should measure the impact of reverting any customizations and personalization. In many cases, a particular artifact is not intended to be heavily customized and the decision is straightforward. However, if the artifact is heavily customized in production, you should consider any design alternatives that would eliminate the need to revert the customizations.

You must carefully manage how solution artifacts are updated. In some cases, directly changing a definition in a solution artifact file-based definition is not supported. For example, this is true of content types. Instead, you can do this programmatically through the SharePoint object model.

In cases when a solution artifact is modified, additional logic that relies on the SharePoint object model may be required to repair information in the content database. For more information, see [Upgrading the Training Management Application](#).

An additional approach to converting customizations back to individual solution artifacts is to use custom site templates or custom list templates that are stored as template (.stp) files. This approach can be useful for minor changes across all new sites. A custom template captures the differences from a base site or a list definition. It uses a binary form that is not modifiable. This approach is simpler than migrating to individual solution artifacts. It is more common to use this approach at the custom list-template level than at the custom site-template level. Custom site templates are too coarse grained to provide a sufficient level of detail. Using custom site template or custom list templates precludes publishing the site in the future with a Web content management system. Therefore, you should carefully weigh this decision. For more information about the limitations of templates, see [Deciding Between Custom Templates and Definitions](#) on MSDN.

Note:

The Training Management and Partner Portal applications do not convert customizations to solution artifacts.

The updated application is repackaged as the same SharePoint solution. This means that the new WSP has the same solution ID as the original WSP. Two different methods can be used to deploy the updated application.

In both cases, modified solution artifacts are deployed to the file system and any logic to repair information in the content database must be executed through a feature receiver when the feature is activated. To cause this logic to execute, you must reactivate features on any affected sites through either scripting or site administration.

These approaches to upgrading are simpler than supporting independent versions of an application with migration. However, not all solution artifacts can be changed using this approach; for example, site definitions and side-by-side operation of application versions is not supported.

Using the Upgrade Solution Command to Upgrade Applications

To upgrade an application, you can use the **upgradesolution** command in the Stsadm.exe command-line tool. This approach is appropriate when you are not adding any new SharePoint features or you do not want to retract the original SharePoint solution. Generally, the **upgradesolution** command executes more quickly than using a retract-and-redeploy approach, which is discussed later in this topic. Using the **upgradesolution** command puts the updated SharePoint solution into the solution store and deploys the files to the file system. This is similar to adding and then deploying the initial SharePoint solution with a single command. However, unlike the initial deployment, the **upgradesolution** command does not install any new SharePoint features that are included in the SharePoint solution. Consequently, you must use the installation script to install any new SharePoint features after **upgradesolution** runs. You also need to force reactivation on any sites that use the SharePoint feature if any repair logic is required.

The **upgradesolution** command removes any previous assembly versions from the global assembly cache and installs new versions. This has implications for managing certain areas such as workflow updates when instances of the workflow are executing. The SharePoint Guidance team splits the deployment of the Training Management application into two SharePoint solutions. This allows instances of the previous workflow to run to completion when upgrading to a new version. For more information about how to organize features, see [Organizing Features](#).

In some cases, you may want SharePoint feature deactivation logic to execute before running the **upgradesolution** command. You also need to include SharePoint feature activation logic on sites where the SharePoint feature is activated in your script. The **upgradesolution** command can also retract a version and restore the previous files. However, if you have used any repair logic, you must also restore the previous settings. This can be complicated and requires that you store the previous state. In practice, reverting to a previous version is difficult for any changes, except for very simple changes. For more information about how the Training Management application uses the **upgradesolution** command, see [Upgrading the Training Management Application](#).

For more information about this command, see [Upgrading a Solution](#) on MSDN and [Upgrade Solution: Stsadm operation \(Windows SharePoint Services\)](#) on TechNet.

Using a Retract-and-Redeploy Operation to Upgrade Applications

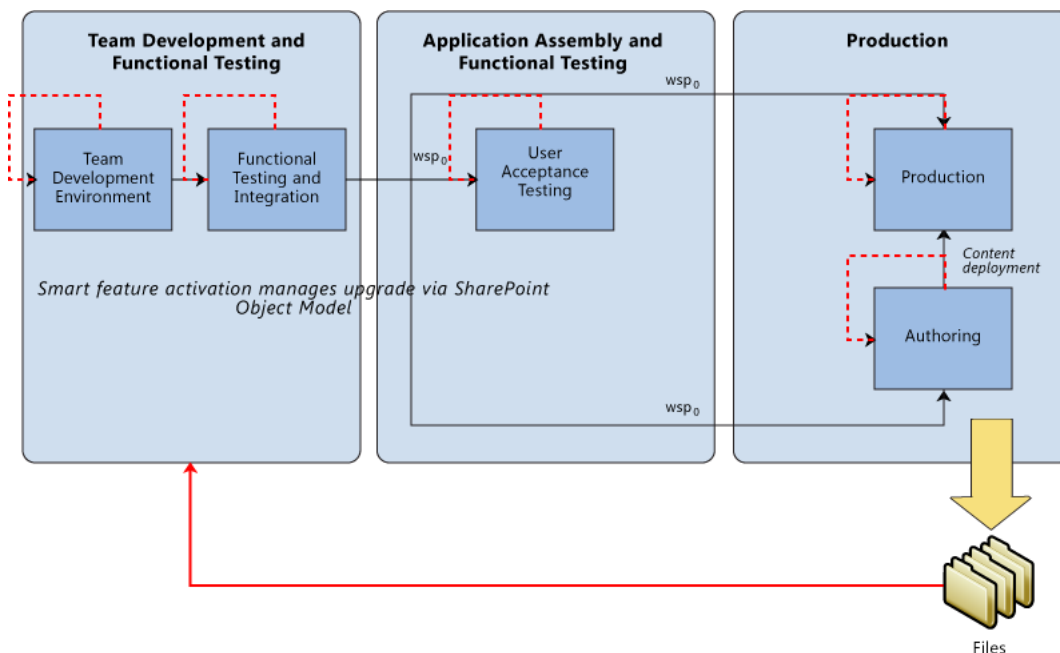
Retracting and redeploying the SharePoint solution retracts and removes the previously installed SharePoint solution with the **retractsolution** and **deletesolution** commands in the Stsadm.exe command-line tool. It then redeploys the SharePoint solution by re-adding and redeploying the updated SharePoint solution with the **addsolution** and **deploysolution** commands in the Stsadm.exe command-line tool. Unlike the **upgradesolution** command, retract and redeploy installs new SharePoint features when the SharePoint solution is redeployed. This may eliminate some steps in the installation script. Retracting a SharePoint solution does not remove any information from the content database. Therefore, sites are intact after redeployment. The retract-and-redeploy approach removes and replaces all files, whether or not they have changed. Therefore, it has a more significant performance impact than using the **upgradesolution** command.

As with the **upgradesolution** command, you must reactivate any installed features on any sites that use the SharePoint features if you must use any repair logic. Similarly, you must include SharePoint feature deactivation in the installation script if you want the deactivation logic to execute before you retract the original SharePoint solution. You should use caution with this approach if you cannot take the sites that use the solution offline. If a site is based on a site definition that is being removed and reinstalled, users will experience unpredictable behavior during the upgrade process.

Creating a New SharePoint Solution and Migrating the Application

The following illustration shows the path that the SharePoint solution follows from the development environment to the production environment when you are creating a new version of the application. The solution artifacts that are contained in the SharePoint solution are named `wsp0` in the illustration.

Creating a new solution



The alternative to upgrading a solution through either the retract-and-redeploy operation or the **upgradesolution** command is to programmatically provide a new version that is defined as a new solution with a new solution ID. Although upgrading a solution with this approach is reliable and supports any type of modifications, the migration of the previous application to the new application requires substantial development work. This approach is costly and difficult to generalize and is beyond the scope of this guidance. Situations where you may need this approach are when the new version must run side-by-side with earlier versions and when you have no control over where the

application is deployed, such as an independent software vendor.

If you need instances of the new version to run side-by-side within a site collection with instances of the previous version, you must take this approach. Side-by-side operation is challenging because certain elements, such as content types, are defined at the site-collection level. You must ensure that you do not alter definitions that the previous version requires. For example, you do not make structural changes to a content type that will break the original version. In many cases, you may need to define a logically different version of an artifact as a completely new artifact. This means it has a different ID.

If side-by-side operation is not required and you are deploying to a new site collection, it is possible to reuse definitions. You can migrate only the required artifacts from an existing site collection to the new site collection. In this case, it is not critical to prevent duplication of identifiers. Because the new version is a new solution, you must implement the logic to migrate the data and, optionally, customizations from a previous version instance to the new version.

 Note:

The [upgrade](#) command in the Stsadm.exe command-line tool upgrades a SharePoint installation from one version to the next. This command is specific to SharePoint version upgrade; it is not a general purpose mechanism for upgrading any solution. Using the **upgrade** command for a general solution upgrade is not a supported scenario for SharePoint and should be avoided. This operation internally performs a number of checks and operations that are related to the SharePoint version upgrade process. These checks and operations may not be appropriate for upgrading application versions.

An Example of Deploying a New Application

This topic demonstrates how to deploy a new SharePoint application. It uses the Training Management application as its specific example. The two major steps are to create a WSP and an Install.bat file. SharePoint provides tools that help you accomplish both of these tasks. For an overview of how to deploy a new application, see [Deploying a New Application](#). This topic also describes how to adapt the Training Management application's Install.bat file to your own situation.

Creating the WSP File

You must create a WSP file for each application that you want to deploy to a server farm. The following procedure explains this.

To create a WSP file

1. Open \Source\TM\Contoso.TrainingManagement.RI\Contoso.TrainingManagement.RI.sln.
2. Build the solution.
3. In Solution Explorer, right-click **Contoso.TrainingManagement**, and then click **Package Solution**.
4. Verify that ContosoTrainingManagement.wsp is created in the folder Source\TM\Contoso.TrainingManagement.RI\Contoso.TrainingManagement\bin\Debug.

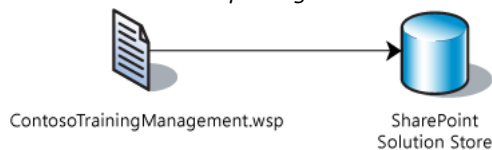
Creating the Install.bat File

After you create the WSP file, you can create the Install.bat file. This is an installation batch file. You should create an Install.bat file for each environment, such as the test environment and the UAT environment. An Install.bat file makes the installation process repeatable and more efficient. If you are deploying the Training Management application, perform the steps in the procedure for both Web solution package files. The optional steps can be performed either in the batch file or through the SharePoint user interface.

To create the Install.bat file

1. Add each WSP to the SharePoint farm solution store. For the Training Management application, add both the SharePoint feature and SharePoint workflow WSP files to the solution store. The following figure illustrates this.

Add a Web solution package to the solution store



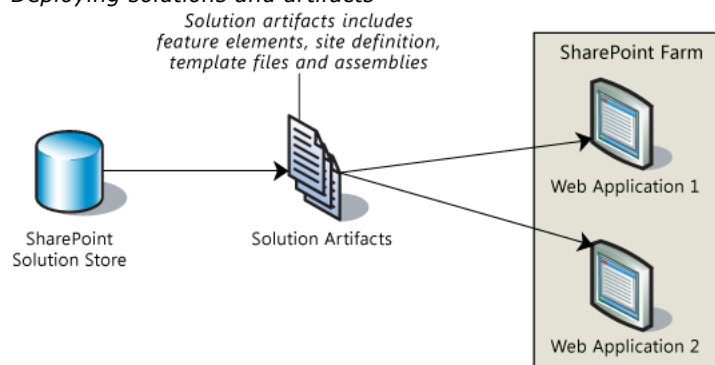
To add a solution to the store, include the following command in the Install.bat file:

stsadm -o addsolution -filename ContosoTrainingManagement.wsp

The Stsadm.exe program is located on the drive where SharePoint is installed. The path is %COMMONPROGRAMFILES%\Microsoft shared\web server extensions\12\bin. You must be an administrator on the local computer and have access to the SharePoint configuration database to use the Stsadm.exe command-line tool. For more information about the Stsadm.exe command-line tool, see [Stsadm command-line tool \(Office SharePoint Server\)](#) on TechNet.

2. Deploy each solution and its associated artifacts to each SharePoint Web application in the SharePoint farm. For the Training Management application, deploy both the Training Management solution artifacts and the Training Management workflow solution artifacts. The following figure illustrates this.

Deploying solutions and artifacts

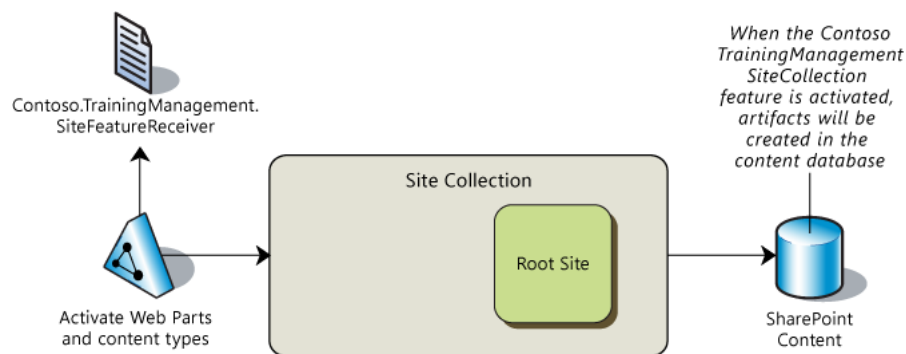


To deploy a solution, include the following command in the Install.bat file:

stsadm -o deploysolution -name ContosoTrainingManagement.wsp

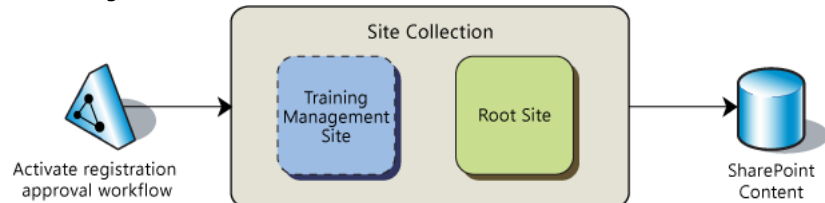
3. (Optional) Activate the features at either the site collection level or the site level. The scope should be the same as the scope of the feature. If you are using the Stsadm.exe command-line tool, the scope is defined by the **-url** parameter. For the Training Management site, activate the features ContosoTrainingManagementSiteCollection, ContosoTrainingManagementRegistrationApproval_v1, and ContosoTrainingManagementRegistrationApprovalAssociation_v1. The following figure illustrates activating the site collection feature.

Activating the site collection features



The following figure illustrates activating the workflow feature.

Activating the workflow

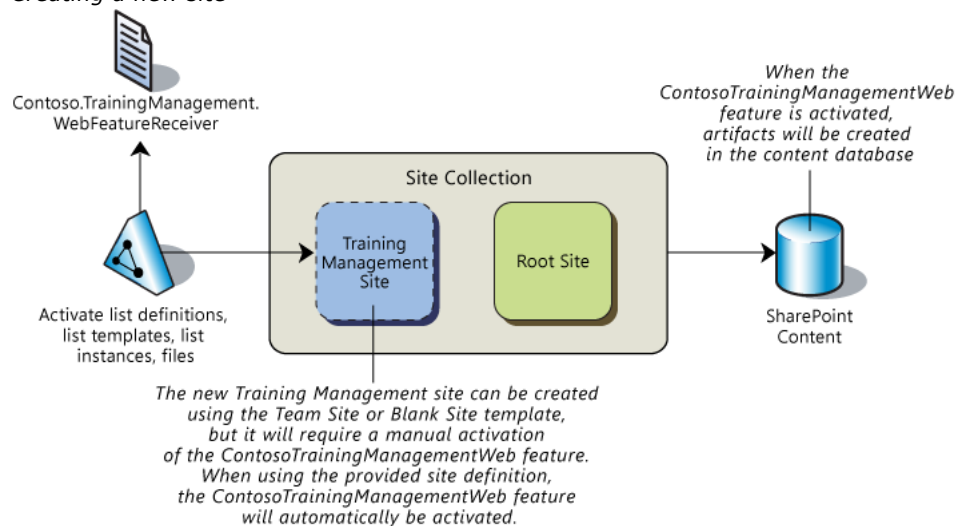


To programmatically activate features, include the following command in the Install.bat file for each feature:

stsadm -o activatefeature -id <guid> -url localhost

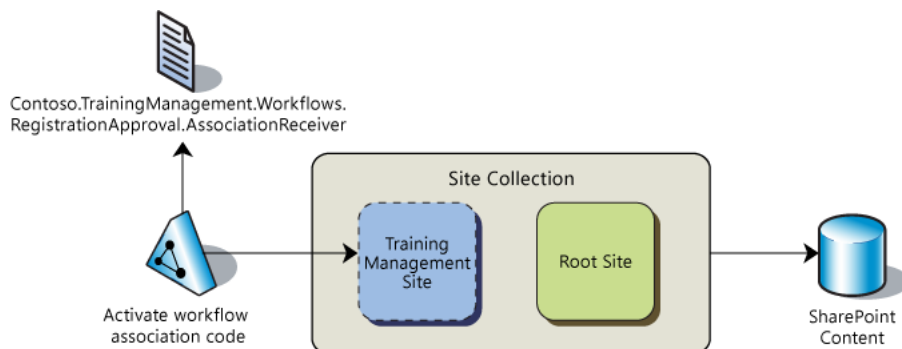
- (Optional) Create a new site that is based on a site definition or template. The following figure illustrates this.

Creating a new site



The following figure illustrates the SharePoint installation after you deploy the application.

Training Management SharePoint installation



Using the Training Management Install.bat File

You can manually create the Install.bat file from the steps that are described in [Creating the Install.bat File](#) or you can use the file from Training Management application as a template. The Training Management's batch file is designed to run from a folder on a Web front-end server. This folder needs to contain the Install.bat and Setup.bat files and the Web solution package(s). The Setup.bat file is generated by Visual Studio extensions for Windows

SharePoint Services, which is located in the ..\bin\debug folder. To adapt the file to your own application, you must do the following:

- Set the port number.
- Set the **TargetWebUrl** parameter.
- Set the **TargetSiteUrl** parameter.
- Enter the name of the Web site to create. Depending on your application, you may not want to create a Web site from a batch file. If you do, omit this step.
- If you have multiple solutions to deploy, you must modify the batch file to call each feature's Setup.bat file.

Note:

The Setup.bat file adds the Web solution package to the solution store, deploys the solution, and activates the features. This corresponds to step 1 through step 3 in [Creating the Install.bat File](#). If you want to run the Install.bat file from a folder on a remote computer, you must change the **localhost** parameter to the remote computer name in both the Install.bat file and the Setup.bat file.

This is the code for the Training Management application's Install.bat file. You can use the following batch file as a starting point to build your own Install.bat file.

```
@echo off
setlocal

set SPLocation=%CommonProgramFiles%\Microsoft Shared\web server extensions\12
set SPAdminTool=%SPLocation%\BIN\stsadm.exe
set Install=
set Uninstall=

set port=80
set AppPool="SharePoint - %port%"
set DefaultWebUrl=http://localhost:%port%
set DefaultSiteUrl=http://localhost:%port%
set TargetWebUrl=http://localhost:%port%/training
set TargetSiteUrl=http://localhost:%port%

set SPTemplateLocation=%SPLocation%\template
set SPFeaturesLocation=%SPTemplateLocation%\features
set SPSiteTemplateLocation=%SPTemplateLocation%\sitetemplates

set ValidationFailed=

echo.
echo.***** Resetting IIS *****
iisreset

echo.
echo.***** installing wsp *****
echo.*setup.bat /install /siteurl %TargetSiteUrl%
call setup.bat /install /siteurl %TargetSiteUrl%

echo.
echo.***** creating triaing web *****
echo.*%SPAdminTool%" -o createweb -url %TargetWebUrl% -sitetemplate "CONTOSOTRAINING"
-ttitle "Training Site"
call "%SPAdminTool%" -o createweb -url %TargetWebUrl% -sitetemplate "CONTOSOTRAINING"
-ttitle "Training Site"

echo.
echo.***** Resetting IIS *****
iisreset

pause
```

Deploying to Multiple Web Front-End Servers

If you want to deploy the application to multiple Web front-end servers, you need to modify the Setup.bat file. The following code is generated by Visual Studio extensions for Windows SharePoint Services.

```
echo Deploying solution %PackageName% ...
"%SPAdminTool%" -o deploysolution -name "%PackageName%" -local -allowGacDeployment -url
%TargetWebUrl%
```

The following code is the modified code. The **local** parameter is replaced with **immediate** and the file now includes an **execadmsvcjobs** command.

```
echo Deploying solution %PackageName% ...
```



```
"%SPAdminTool%" -o deploysolution -name "%PackageName%" -immediate -allowGacDeployment -url
%TargetWebUrl%
"%SPAdminTool%" -o execadmsvcjobs
```

Uninstalling Considerations

The following code is a batch file that uninstalls the Training Management application. This batch file uninstalls the existing version of the application. Run this file before you install a new version.

Because the Uninstall.bat file calls the Setup.bat file that is generated by Visual Studio extensions for Windows SharePoint Services, the Uninstall.bat file must be in the same folder as the Setup.bat file.

If the uninstall batch file fails, follow the manual steps that are listed in [How to: Manually Uninstall the Training Management Application](#).

Uninstall Batch File

```
@echo off

set SPLocation=%CommonProgramFiles%\Microsoft Shared\web server extensions\12
set SPAdminTool=%SPLocation%\BIN\stsadm.exe
set Install=
set Uninstall=
set port=80
set AppPool="SharePoint - %port%"
set DefaultWebUrl=http://localhost:%port%
set DefaultSiteUrl=http://localhost:%port%
set TargetWebUrl=http://localhost:%port%/training
set TargetSiteUrl=http://localhost:%port%

set SPTemplateLocation=%SPLocation%\template
set SPFeaturesLocation=%SPTemplateLocation%\features
set SPSiteTemplateLocation=%SPTemplateLocation%\sitetemplates

set ValidationFailed=

echo.
echo.***** resetting IIS *****
call iisreset /STOP

echo.
echo.***** deleting training web *****
echo.*"%SPAdminTool%" -o deleteweb -url %TargetWebUrl%
call "%SPAdminTool%" -o deleteweb -url %TargetWebUrl%

echo.
echo.***** uninstalling wsp *****
echo.*setup.bat /uninstall /siteurl %TargetSiteUrl%
call setup.bat /uninstall /siteurl %TargetSiteUrl%

echo.
echo.***** resetting IIS *****
call iisreset

pause
```

How to: Manually Uninstall the Training Management Application

This topic explains how to use the SharePoint Central Administration to manually remove previous versions of the Training Management solution before installing a new version. You must use this procedure if the setup/uninstall command in the Uninstall.bat file is unsuccessful.

To manually uninstall the application

1. Browse to the Training Management site.
2. Select **Site Actions**.
3. Select **Site Settings**.
4. Select **Site features** on the Site Administration tab.
5. Click the **Deactivate** button for Contoso Training Management.
6. Select **Site Actions**.
7. Select **Site Settings**.
8. Select **Go to top level site settings**.
9. Select **Site collection features**.
10. Click the **Deactivate** button for Contoso Training Management.
11. Browse to the SharePoint Central Administration Web site.
12. Click the **Operations** tab.
13. Under **Global Configuration**, click **Solution management**.
14. Click **contosotrainingmanagement.wsp**.
15. Click **Retract Solution**, and then click **OK**.

16. Return to the Solution Management page and refresh it.
17. Repeat step 6 until the retract operation is complete. This is when the status changes from **Deployed** to **Not Deployed**.
18. After the status changes, click **contosotrainingmanagement.wsp**.
19. Click **Remove Solution**.
20. In the message confirmation dialog box, click **OK**.
21. If contosotrainingmanagementregistratoinapproval_v1.wsp exists, perform step 4 through step 10.
22. Click the **Application Management** tab.
23. Under **SharePoint Site Management**, click **Delete site collection**.
24. Select the site collection for the Training Management application.
25. Click **Delete**.
26. In the message confirmation dialog box, click **OK**.

Upgrading an Application

The following sections include guidance on how to upgrade a SharePoint solution, using the Training Management application as an example. There is also a brief discussion of issues that you should consider when you plan an upgrade to a SharePoint application that has been deployed to a production environment. For more information about this topic, see [Deployment Scenarios](#).

Site Columns

Site columns that are not customized reference the site column SharePoint feature element file. Adding new field definitions to the SharePoint feature element file causes any new activations of the SharePoint feature to provision the new site column. Existing activations of the SharePoint feature do not automatically receive the new field definitions. You must reactivate the SharePoint feature for this to occur. Do not remove site columns from a SharePoint feature element file because this can cause list items that use the site columns to break.

The upgrade SharePoint solution for the Training Management application adds a new site column named Instructor. The new site column is part of the ContosoTrainingManagementWeb SharePoint feature. You must reactivate this SharePoint feature to allow existing sites to receive the new site column.

The new site column is added by including the file Contoso.TrainingManagement.RI.Upgrade\Contoso.TrainingManagement\ContentTypes\TrainingCourseContentType\TrainingCourseInstructorField.xml in the Contoso.TrainingManagement.RI.Upgrade Visual Studio solution. The following code shows the new field.

XML

```
<Elements Id="82fde2ad-e0ae-4889-b3c8-8ba72c822ac1"
xmlns="http://schemas.microsoft.com/sharepoint/">
  <Field ID="{65f64276-ef6f-4357-a33a-183e6e6a5863}" Type="Text"
Name="TrainingCourseInstructor" DisplayName="Instructor" Hidden="False" Required="FALSE"
Sealed="FALSE" DisplaceOnUpgrade="TRUE" />
</Elements>
```

Content Types

When upgrading a content type that is defined by a **ContentType** feature element, you must consider whether the content type has been customized in the content database or whether it is found only in the file system of the Web front-end server. The two points to remember are the following:

- If a content type is not customized, sites reference the content type definition that is located in the file system of the Web front-end server.
- When a content type is enabled by a list, a copy of the content type is created in the list definition in the content database, even if no customization is involved.

When content types are located in both the Web front-end server and the content database, you cannot simply upgrade the content type by providing a new content type definition on the server. For example, when an upgrade adds a new field to a content type definition, any list instance that is enabled for that content type does not automatically receive the new field. You should consider the following guidelines:

- The recommended approach for adding new fields to a content type is to add them programmatically using the SharePoint object model.
- When a content type is programmatically upgraded, the SharePoint object model provides the option to upgrade child content types, including content types in list instances.
- Removing fields or updating the type of a field is not recommended. The recommended approach for removing a field is to make the field hidden.
- The recommended approach for updating the type of a field is to create a new field and set the old field to hidden.
- It is important to understand that if you follow these guidelines, you cannot have a centralized XML file that contains the content type definition after the upgrade. The content type XML should remain untouched after it is initially deployed, and all upgrades should be done through code.

The upgrade SharePoint solution for the Training Management application contains code in the **SiteFeatureReceiver** class in the Contoso.TrainingManagement project to upgrade the Training Course content type with an **Instructor** field.

The following code is an example of how to add fields to a content type through the SharePoint object model.

C#

```
public override void FeatureActivated(SPFeatureReceiverProperties properties)
{
    SPSite site = null;
    object parent = properties.Feature.Parent;

    if (parent is SPWeb)
    {
        site = ((SPWeb)parent).Site;
    }
    else
    {
        site = (SPSite)parent;
    }
}
```

```

    }

    this.UpdateSiteMap(site);

    this.UpdateTrainingManagementContentType(site);

    this.UpdateListItemEventReceivers(site);
}

C#
private void UpdateTrainingManagementContentType(SPSite site)
{
    this.EnsureFields(site, ContentTypes.TrainingCourse, new
    Guid(Fields.TrainingCourseInstructor));
}

C#
private void EnsureFields(SPSite site, string contentTypeId, Guid fieldId)
{
    SPContentTypeId spContentTypeId = new SPContentTypeId(contentTypeId);
    SPContentType contentType = site.RootWeb.ContentTypes[spContentTypeId];

    bool fieldLinkExists = false;
    foreach (SPFieldLink fieldLink in contentType.FieldLinks)
    {
        if (fieldLink.Id == fieldId)
        {
            fieldLinkExists = true;
            break;
        }
    }

    if (!fieldLinkExists)
    {
        SPField fieldToAdd = site.RootWeb.Fields[fieldId];
        contentType.FieldLinks.Add(new SPFieldLink(fieldToAdd));
        contentType.Update(true);
    }
}

```

This code queries the SharePoint object model for the **SPField** object that matches the **fieldID** variable. It stores this in a local variable named **fieldToAdd**. The code then creates a new field link based on the field that is stored in the **fieldToAdd** variable. The code adds the link to the content type that is identified by the **spContentTypeId** variable.

Note:

Content type upgrades are performed through a SharePoint feature receiver. This means that you must activate the SharePoint feature. Most items that are defined in SharePoint feature element manifests must be reactivated. Components that are code in assemblies do not need reactivation. Files also do not need reactivation unless you are provisioning new Web Parts.

Web Parts

Web Parts can be upgraded by providing a new version of the assembly that contains the Web Part code. If the assembly that contains the Web Part code contains a new assembly version number, you must perform the following additional steps to ensure that existing instances of the Web Part function correctly:

- Upgrade the **SafeControl** element in the Web.config file to reflect the new assembly version. For more information about the **SafeControl** element, see [SafeControl Element \(Solution\)](#) on MSDN.
- Upgrade any references to the Web Part definition in pages that contain Web Parts.
- Upgrade the .webpart file for the Web Part with the new assembly version to ensure that any new instances of the Web Part are properly imported.

In the upgrade SharePoint solution of the Training Management application, updates to the Web Part properties are made to the Direct Reports Web Part and the Training Budget Web Part. In the **TrainingBudgetWebPart**, a Web Part property named **ShowTransactionGridLines** is added. In the **DirectReportsWebPart**, the **ShowLogin** Web Part property is removed.

Files and Modules

If a file has not been customized, it can be upgraded by replacing the file with the new version in the Web front-end servers. Any changes to the file are automatically detected after a process cache refresh. However, this approach is not recommended. When a new version of the file is copied over to the Web server, information about the file, such as its size and properties, may not correctly reflect the actual information about the file.

Additional consideration must be made when updating Web Part pages. Updates to Web Part pages can include list views and Web Parts. SharePoint always takes a non-destructive approach to upgrading Web Part pages. Any Web Part or list view that was provisioned through a SharePoint feature element is not removed or merged when the SharePoint feature is upgraded and reactivated. This may cause duplicate Web Parts or list views to appear

after a SharePoint feature is reactivated.

One approach to correcting this behavior is to provide a new name for the file being upgraded and using the [SPFile.MoveTo](#) method. For example, the Training Management Reference Implementation contains a class named `managerdashboard.aspx`. To upgrade this file with additional Web Parts, you can change the name of `managerdashboard.aspx` to `managerdashboard_v2.aspx`. You can then write a feature receiver file that will move the `managerdashboard_v2.aspx` file to `managerdashboard.aspx` URL.

Note:

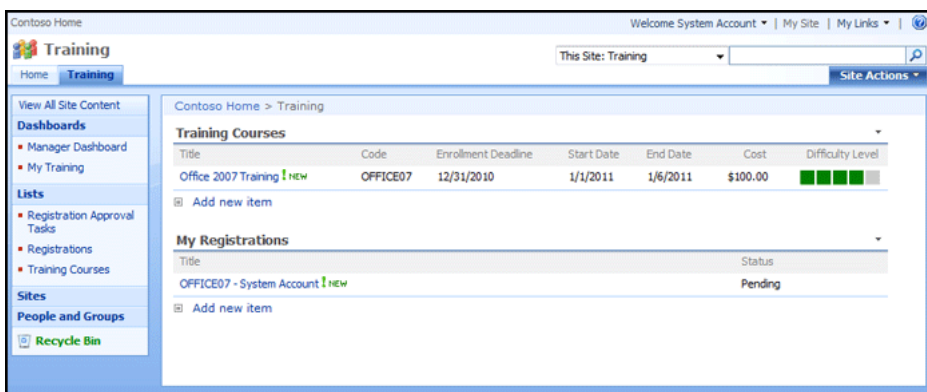
One word of caution is that this approach will cause the loss of all user customization to the file.

This approach is only recommended for files where heavy customizations by users will not occur. If preservation of user customization is required, other approaches should be used. For example, you can programmatically identify Web Parts and remove any duplicates. Alternatively, you can programmatically add the Web Parts to your Web Part page during feature activation. Neither of these approaches is demonstrated in the upgrade SharePoint solution. A file cannot be removed by removing it from the SharePoint feature. Any removal of files must be performed through the SharePoint object model's [SPFile.Delete](#) method.

The upgrade SharePoint solution of the Training Management application contains file changes and module changes. The upgraded application includes an image file that is displayed on several Web pages.

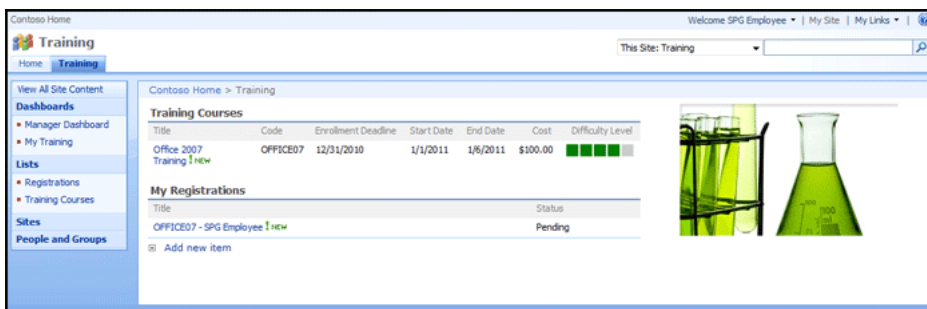
The following illustration shows the original Training.aspx page.

Original training page



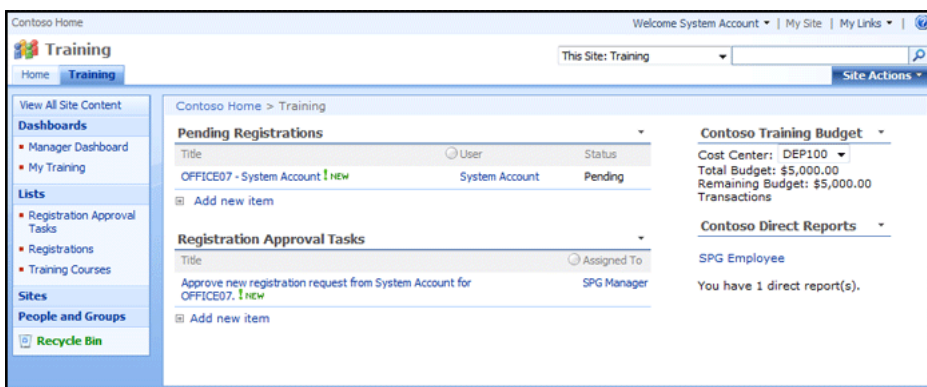
The following illustration shows the upgraded page with the new image.

Training page with upgrade



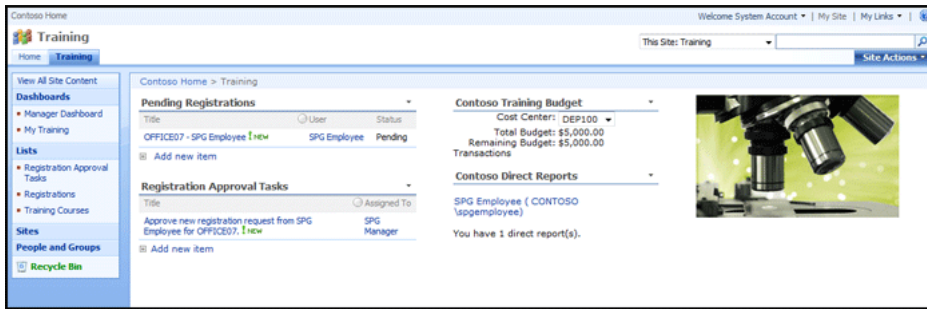
The following illustration shows the original management page.

Original management page



The following illustration shows the upgraded management page with the new image.

Upgraded management page



Application Pages

Application pages are .aspx files that are stored in the virtual _layouts folder of the SharePoint Web front-end server. These pages are deployed once to the Web server and cannot be customized on a site-by-site basis. The pages are available across all sites in a server farm. To upgrade application pages, you can deploy a new version of the .aspx file to the Web server. The new page appears after a process cache refresh. These pages are relatively easy to upgrade because they cannot be customized by users; therefore, they are never stored in the content database. The upgrade to the Training Management application adds an image to the registrationapproval.aspx page. The following illustration shows the original page.

Original registration approval page



The following illustration shows the upgraded registration approval page with the new image.

Upgraded registration approval page



User Controls

User controls are upgraded by replacing the existing .ascx files with new versions of the .ascx files on the Web front-end server.

The upgrade to the Training Management application includes changes to the DirectReports.ascx user control in the Contoso.TrainingManagement.Web project to include an **AlternatingItemStyle** property to the **DataList** control.

List Item Event Handlers

Upgrades to list item event handlers are implemented by installing a new version of the assembly with the code changes. If the version number of the assembly has been changed, the old version information stored in the content database must be upgraded. The assembly information for event handlers is persisted in the content database for each list instance. Upgrades to the assembly must be performed through the SharePoint object model for each list instance. The following code is an example of how to upgrade the **Assembly** property of a

SPEventReceiverDefinition instance, which is found in the SiteFeatureReceiver.cs file in the Contoso.TrainingManagement project.

C#

```
string newAssembly = "Contoso.TrainingManagement, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=9f4da00116c38ec5"; //our new assembly info
string newClass = "Contoso.TrainingManagement.TrainingCourseItemEventReceiver";
SPContentTypeId contentTypeId = new SPContentTypeId(ContentTypes.TrainingCourse); //our
custom ctype

foreach ( SPWeb web in site.AllWebs )
{
    using ( web )
    {
        for ( int i = 0; i < web.Lists.Count; i++ )
        {
            SPList list = web.Lists[i];
            SPContentTypeId bestMatch = list.ContentTypes.BestMatch(contentTypeId);
            if ( bestMatch.IsChildOf(contentTypeId) )
            {
                for ( int j = 0; j < list.EventReceivers.Count; j++ )
                {
                    SPEventReceiverDefinition eventReceiverDefinition =
list.EventReceivers[j];
                    if ( String.Compare(eventReceiverDefinition.Assembly, newAssembly, true)
!= 0 )
                    {
                        list.EventReceivers.Add( eventReceiverDefinition.Type, newAssembly,
newClass );
                        eventReceiverDefinition.Delete();
                        list.Update();
                    }
                }
            }
        }
    }
}
```

This code iterates through all lists in a site collection that is enabled for the **TrainingCourse** content type and updates the **SPEventReceiverDefinition** property to use the new assembly information. For more information about the **SPEventReceiverDefinition** class, see [SPEventReceiverDefinition Class \(Microsoft.SharePoint\)](#) on MSDN.

Workflow

Workflows can be upgraded by making changes in the workflow code and redeploying the workflow assembly. For simple upgrades that do not require changes to the workflow properties or workflow activities, the version number for the assembly should remain the same, and the new version of the workflow code should accommodate any old instances of the workflow. SharePoint serializes workflow instances and persists them to the content database. Any changes to the assembly can cause the workflow serialization engine to fail for the existing workflow instances.

In scenarios where significant changes to the workflow are required, including upgrades to the workflow properties, a best practice is to assign a new version number to the assembly. You should also provide a new version of the workflow template XML file. This approach also requires that a new workflow association be created.

By setting the workflow association of the old version of the workflow to **No New Instances**, you can prevent any additional workflow instances from being created with the old workflow. This allows you to introduce a new version of the workflow that runs side-by-side with the old version. The old workflow instances will complete, and the new workflow instances can use the upgraded code. One drawback to this solution is that if there are bugs in the old version of the workflow that are fixed in the new version, the old instances of the workflow still have the bugs in the code.

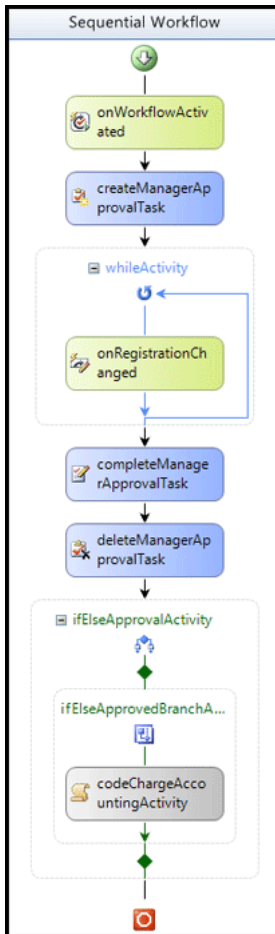
Note:

It is important that you consider upgrade scenarios when you initially create workflow solutions. Planning for upgrades early in the design and development phases prevents many difficulties during the maintenance phase.

For more information, see "Saving Workflow State and Upgrading Workflow Assemblies" in [Developing Workflow Solutions with SharePoint Server 2007 and Windows Workflow Foundation](#) on MSDN.

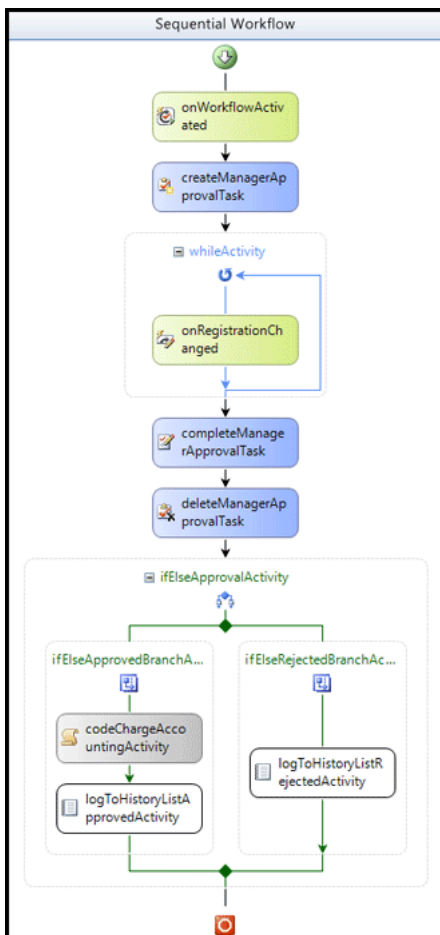
The upgraded Training Management application includes changes to the registration approval workflow. The upgraded workflow contains new custom code activity in the **ifElseApprovalActivity** to log to the workflow history list. The following illustration shows the original workflow.

Original workflow



The following illustration shows the upgraded workflow with the logging activity.

Upgraded workflow



Custom Actions

Custom actions can be upgraded by providing an upgraded SharePoint feature element. To upgrade an existing custom action, make changes to the SharePoint feature element without changing the **Id** attribute of the **CustomAction** element. To remove a custom action, add a **HideCustomAction** element to a SharePoint feature element. The upgrade to the Training Management application does not include any changes to custom actions. For more information about the **CustomAction** element, see [Custom Action Element \(Custom Action\)](#) on MSDN. For more information about the **HideCustomAction** element, see [HideCustomAction Element \(Custom Action\)](#) on MSDN.

List Templates

List templates can be upgraded by providing a new version of the list template definition in a SharePoint feature. To upgrade certain properties of a list template that are persisted in the content database, such as **OnQuickLaunch**, **DisplayName**, **Description**, and **Image**, use the SharePoint object model. Any new list templates in a SharePoint feature are automatically available when the feature is reactivated. The upgrade to the Training Management application does not include any changes to list templates.

List Definitions

List definitions can be upgraded to include new fields or upgrades to non-customized views by upgrading the Schema.xml file in the SharePoint feature. Removing fields from the Schema.xml file is not a recommended practice. Modifying a field type of a field is also not recommended. Instead, create a new field to replace the old field. The upgrade to the Training Management application does not include any changes to list definitions.

List Instances

New list instances can be created by providing a new **ListInstance** element in a SharePoint feature. Reactivating the SharePoint feature that contains the **ListInstance** element causes new list instances to appear. Any existing list instances that are provisioned from modified or deleted **ListInstance** elements are not affected. Any upgrades to existing list instances should be performed through code. Also, forcing the reactivation of a SharePoint feature that contains an upgraded **ListInstance** element with data items may cause duplicate list items. The upgrade to the Training Management application does not include any changes to list instances. For more information about the **ListInstance** element, see [List Instance Element \(List Instance\)](#) on MSDN.

Publishing Page Content Type

The guidelines for upgrading [content types](#) are applicable for upgrading publishing page content types. After a publishing page content type is updated, you will most likely need to update all publishing page layouts associated with this content type. For example, if you add a new field, you must also update the page layout to author and display content for the new field. Another example is that if you make a field hidden, you must also update the page layout to hide or remove the field.

Publishing Page Layouts

Publishing Page Layouts are typically deployed with a module. You can follow the guidelines for upgrading [Files and Modules](#) for publishing page layouts. If you need to hold new content for your publishing pages, you will mostly likely also need to update the associated publishing page content type. Publishing page layouts are stored in a site collection's master page gallery, and properties can be updated through the SharePoint Web user interface (UI).

Team Development Overview

This section includes the following topics:

- [Introduction to Team Development](#)
- [Using Continuous Integration](#)
- [How to: Create an Automated Build and Deployment Solution with Team Foundation Server Team Build](#)

Introduction to Team Development

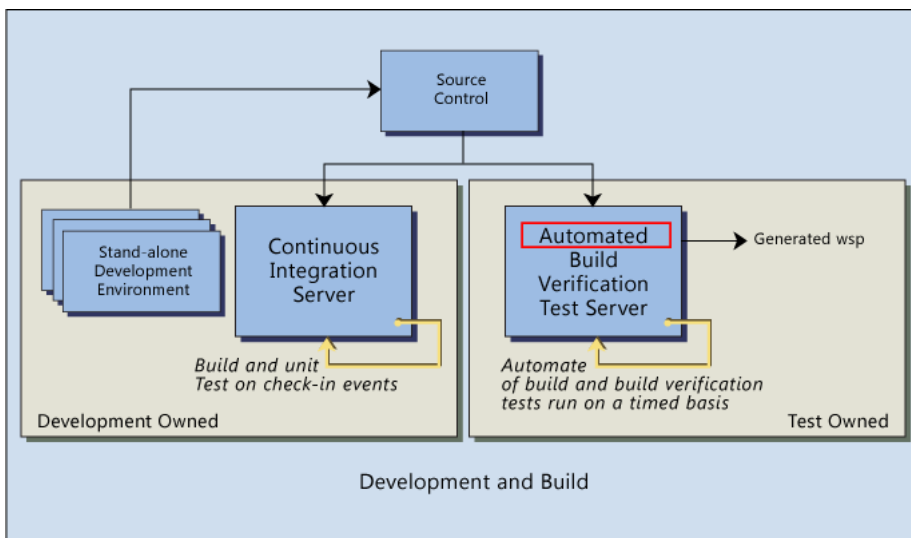
In many small-sized to medium-sized organizations, small teams of SharePoint developers simultaneously work on the same application. In addition, many small teams can be concurrently working on different applications that have interdependencies.

This topic discusses how the Partner Portal and Training Management applications were developed with a small team of developers and testers. It shows the process that the development and test teams use to develop and build the SharePoint solution package that is then deployed to the testing, staging, and production environments. For information about the end-to-end flow of an application from development to production, see [Deployment Scenarios](#).

The initial investment, both in time and money, to create stable development and test environments benefitted the project in the long term. The team used an agile approach to develop the application. They used Visual Studio Team System for source control, performing builds, testing, and managing work items.

The following illustration shows the development and build environments.

Development and build environments



The following describes the development and build environments:

- **Stand-alone development environment.** Each developer uses a stand-alone workstation. Each development workstation includes a running instance of SharePoint that is installed either directly on the workstation or on a virtual computer. This allows the code to be locally deployed and tested before checking it into source control.
- **Source control system.** The source control system manages and versions all files that are used in the development process, including source code, XML files, and resources.
- **Continuous integration server.** The Continuous Integration (CI) server performs builds on a frequent basis (the builds can be either time-based or event-based) and runs unit tests against the build. Unit tests run independently of SharePoint.
- **Build verification test server.** The build verification test (BVT) server performs builds and runs a suite of tests that exercise the application while running under SharePoint. It establishes that the application's basic functionality is correct. BVTs run less frequently than CI tests.

The Development Environment

The development team works locally to develop application logic and structures, as well as unit test cases. (The use of unit testing by developers is strongly encouraged.) Developers should synchronize their code with all the application items that are not checked out and make sure that all the unit test cases pass before they check in their code. After they are satisfied with their changes and additions, developers check in their changes to source control.

Depending on your experience and approach to software development, how you define unit testing may vary. The agile development community considers unit tests to be isolated tests of code that have no system dependencies. The patterns & practices SharePoint Guidance team replaced SharePoint dependencies with objects that return the specific set of responses that are required by the unit tests. These replacement objects are named *mock objects*. The mock objects were developed with a third-party tool named Typemock Isolator. For more information, see the [Typemock Isolator](#) Web site.

Mock objects eliminate any dependencies on SharePoint for developing and running unit tests. The reference implementation applications include the unit tests, which require Typemock Isolator to run, as an optional part of the guidance. For more information about unit testing with mock objects, see [Unit Testing with Mock Objects](#).

An alternative to mock objects is to write stubs, but this requires a significant effort for a complex object model such as SharePoint.

The Source Control and Build/Continuous Integration Environments

Source control is a necessity for any software development environment. It provides versioning and supports automated builds when they are checked in. Typically, build processes label the code to ensure reproducibility. The patterns & practices SharePoint Guidance team runs builds on both the CI and BVT servers. Both build processes label the code that is being built.

Although many organizations use a manual build and verification process, an automated CI environment is preferable. CI is popular with the agile development community. In conjunction with unit tests, it provides a rapid check on the quality of the code. The patterns & practices SharePoint Guidance team's CI process automatically builds the code when it is checked in and executes all the unit tests against the build. In this example, SharePoint is not installed on the CI server. It has only the SharePoint assemblies that are necessary to compile the application. CI ensures that the checked-in code does not conflict with any changes made by other team members. It also reveals conflicts that occur because of unsuspected interdependencies that can exist between different projects. Continuous integration can also run on a timed basis, depending on your needs.

Although setting up the source control and CI environments requires an initial investment of time and money, there are clear benefits to using them. In particular, build and integration issues are quickly discovered and resolved. There are tools available to help set up a continuous integration environment. The patterns & practices SharePoint Guidance team uses Microsoft Visual Studio Team System 2008 Team Foundation Server Team Build. For more information about how the team uses continuous integration, see [Using Continuous Integration](#). For information about Team Foundation Server Team Build, see [Team Foundation Build Overview](#) on MSDN.

The Build Verification Test Server

The patterns & practices SharePoint Guidance team builds and runs the BVTs on a separate server. Build verification tests exercise the application that is running under SharePoint and ensure that the baseline functionality works correctly before deploying the application to the next environment. BVTs can be automated or manual. Ideally, at least some automation is in place, although automating some areas, such as user experience design, can be costly. Like the continuous integration server, the BVT server uses the code in the source control system and builds the application. However, BVTs run less frequently than CI builds, and they take longer to execute. The BVT process creates the SharePoint Web solution package (WSP), deploys it to a local SharePoint installation, and executes a set of automated tests.

The patterns & practices SharePoint Guidance team's automated verification tests are built using Visual Studio Team System Web Tests. If all the automated verification tests pass, the WSP is copied to a share for use by other environments, such as staging and production. Because BVTs run against the real application, they typically take longer to execute than unit tests. The Training Management application's BVTs take approximately 30 minutes to run. This is in contrast to the unit tests, which take only a few minutes. BVTs can easily take several hours with more complex systems. Because the team wanted timely feedback from the continuous integration process, they kept the CI server and the BVT server separate. For more information about creating an automated build environment, see [How to: Create an Automated Build and Deployment Solution with Team Foundation Server Team Build](#).

Possible Alternatives

Many teams think that the CI and BVT environments are too costly. However, it is a best practice to institute some form of automation for building and validating code. It provides a way to detect integration conflicts early and ensures a minimum level of quality before sending the code to the test environment.

The patterns & practices SharePoint Guidance team performs builds using multiple build servers. Both the CI and BVT environments run their own builds. Isolating the environments from each other ensures that the CI builds can succeed without a SharePoint installation. The patterns & practices SharePoint Guidance team wanted to run CI without SharePoint as a proof point, but you can install SharePoint on your CI environment. In many cases, you may want to run all builds for all environments on one build computer.

CI relies on some form of unit testing, but there are alternatives to using mock objects or stubs. For example, you may prefer running unit tests against a live SharePoint installation. In the agile approach, tests that run against a complex system such as SharePoint are considered to be integration tests and would be considered part of build verification testing. Outside the agile community, such tests are often considered to be unit tests.

To develop tests that run against a SharePoint instance, you need to create *setup* and *teardown* logic. The setup logic creates the proper conditions on the SharePoint instance for your tests to correctly run. The teardown logic restores the SharePoint instance to a known state after the tests complete. There are both benefits and drawbacks to this approach. One benefit is that you are not anticipating SharePoint behavior with a mock object. Instead, you are testing against the actual behavior. Another benefit is that you do not require yet another software package.

One drawback is that writing robust setup and teardown logic is time consuming and difficult. Another issue is that the tests are more brittle. Finally, because the tests are complex and take longer to execute than unit tests that use mock objects or stubs, the delay between check-in and verification can be much longer.

You can consider the following simplifications to reduce the complexity of implementing a solution if you cannot invest in a complete environment:

- You can combine the CI and BVT processes on one server. In this case, be careful that the tests do not run at the same time because they may conflict. For example, if assemblies are being installed to the global assembly cache, running CI and BVT builds at the same time creates conflicts.
- If you do not implement unit tests, and therefore cannot run CI, you can still set up automated builds to quickly identify when a developer checks in build-breaking changes that fail compilation.

- You can develop one process that integrates the roles of BVT and CI by developing integration tests (these are unit tests that run against a SharePoint instance) as part of an automated build. In the agile community, these are considered to be build verification tests.
- You should set up a build verification process, even if it is done manually. Deploying builds that are not functional can waste time and frustrate the test team with inoperable or conflicting solutions that are difficult to roll back.

Developing with Visual Studio Team System 2008

Assemblies and many types of file-based artifacts can be developed with Visual Studio 2008. Developers use Visual Studio to write code and build assemblies that are based on the Microsoft .NET Framework. These assemblies reside either in the SharePoint server Web application's bin folder or in the global assembly cache.

Solution artifacts mainly consist of assemblies and content definition files and template files. For a more complete discussion of SharePoint artifacts, see [Defining Artifact Terminology](#). Assemblies can include any of the following components:

- Web Parts
- ASPX pages
- ASCX controls
- Server controls
- Custom field types and field controls
- List event receivers
- List item receivers
- Feature event receivers
- Workflow activities
- Web services
- Timer jobs
- STSADM extensions

File-based SharePoint Solution artifacts are not compiled, and they reside in the file system. These artifacts include items that are customizable and items that are not customizable.

The following artifacts are not customizable:

- Site definitions
- Feature site template association
- Custom actions
- List template definitions
- List Instance definitions
- Web Part definitions
- Images(.jpg, .gif, .png files)
- Themes
- Client-side scripts (JavaScript or JScript)
- Field type definition
- Workflow template definition

The following artifacts are customizable and can be persisted in the content database:

- List definitions
- List template definitions(certain properties can be customized and persisted in the content database)
- Web Part pages (changes in an .aspx file through the SharePoint Designer or object model can cause files to be customized and persisted in content database)
- Master pages
- Cascading style sheets
- Content type definitions
- Site columns

Many file-based artifacts can also be developed with Visual Studio 2008 in conjunction with tools, such as Visual Studio extensions for Windows SharePoint Services version 1.3. Visual Studio extensions for Windows SharePoint Services provides useful templates for many types of artifacts, such as content types definitions, list definitions, and site definitions. Including these artifacts in the Team Foundation Server source control makes them available to other members of the development team while controlling the code change and build processes.

There are other artifacts that are better suited to development outside of Visual Studio. This is particularly true of artifacts such as master pages that are considered SharePoint content. A useful tool for developing many types of artifacts related to content is the Microsoft Office SharePoint Designer.

Authored artifacts developed in SharePoint Designer can be exported to the developer's local file system. They then can be included in a Visual Studio solution (.sln file). Other types of artifacts can be authored in SharePoint by using the Web browser to access various SharePoint content. For example, after navigating to a SharePoint site, you can create custom lists by clicking **Custom Lists** on the **Create** menu. Items developed in SharePoint with the Web browser can be extracted with the SharePoint Solution Generator tool that is included as part of the Visual Studio extensions for Windows SharePoint Services installation. They then can be included in a Visual Studio solution (.sln file). For more information, see [Extracting Artifacts Created with SharePoint Designer and the Browser](#).

By maintaining both the assembly artifact code and file-based artifacts with Visual Studio Team System 2008, development teams have an environment that supports the entire development life cycle. This includes continuous integration and an automated build and deployment solution.

For more information about continuous integration, see [Using Continuous Integration](#). For an example of how to create an automated build and deployment solution, see [How to: Create an Automated Build and Deployment Solution with Team Foundation Server Team Build](#). For more information about using Visual Studio for SharePoint, see [Team-Based Development in Microsoft Office SharePoint Server 2007](#) on MSDN.

Using Continuous Integration

One of the challenges of team development is that team members work on separate parts of the code base but must eventually integrate their code with that of the other team members. Typically, a source control system is the main point of integration for source code and other developed assets. Code integration is simple if there are no or few conflicting changes, but it is difficult if there are many conflicting changes.

Understanding Conflicts

A conflict occurs when the source control system does not require an exclusive checkout and multiple developers try to check in modifications to the same file. For example, assume that developer A retrieves the latest version of a file from source control. The file has version number 1000. Then, developer B also retrieves the latest version of the file, which is still version number 1000. Next, developer B makes modifications to the file and checks it into source control. The source control system sees that developer B has modified version 1000, which is the current version. So far, there are no issues. The source control system accepts developer B's modifications and then updates the version number of the file to 1001. When developer A attempts to check in modifications to version 1000 of the file, source control informs developer A that there is a conflict because this is no longer the latest version of the file. Developer A must merge the modifications with any other modifications that have been made to the file since version 1000 was checked in.

The longer the time period before a developer synchronizes files with source control, the more likely it is that other developers have checked in conflicting changes. There are many factors that can prevent developers from performing frequent synchronizations. One issue is that many developers are unaccustomed to team development. They often wait to check in their code until they consider it to be perfect. This behavior is often driven by a fear of code reviews. Developers refuse to show their code to other team members until it is absolutely necessary. Long delays between check-ins can cause difficult merges. Another problem is that code that is not checked in is also not backed up.

Using Continuous Integration

Because long delays between synchronizations cause difficult and error-prone merge integrations, it is important to perform them frequently. It is also helpful to integrate small amounts of code at a time. One of the benefits of integrating small amounts of code is that there are fewer merge conflicts, and they are easier to resolve. Another benefit is that automated build and testing processes give the developer feedback about whether the code was successfully integrated. If the build breaks, the smaller the amount of code the easier it is to isolate the problem.

The Continuous Integration Build

The process of Continuous Integration (CI) encourages developers to frequently integrate their code. It also provides the benefits of using automated build and testing processes. A CI build is typically triggered by a check-in to source control. This build validates that the latest code in source control builds successfully and passes a suite of unit tests. The CI build must include a comprehensive set of unit tests to provide any value.

To provide timely feedback, CI builds should complete quickly. CI builds can also be configured to provide code coverage and static analysis feedback. However, these are secondary goals. If they cause the CI build to take too long, configure another build process that executes less frequently to provide this information.

The CI build environment for the reference implementation applications do not perform any tests that require a live instance of SharePoint. All the unit tests use mocks that replace the actual SharePoint instance and services. Building the code and running the unit tests only requires that the following SharePoint assemblies are installed on the CI build server:

- **Microsoft.SharePoint**
- **Microsoft.SharePoint.Security**
- **Microsoft.SharePoint.WorkflowActions**
- Any referenced assemblies

Recommendations for Checking In Code

The following procedure shows the recommended steps to check code into source control.

To check in code to source control

1. Verify that the CI build is working and passing all the unit tests. If it is not, notify the development lead. Do not check in code until the issue is resolved.
2. Test all code modifications in a local development environment. Make sure that the unit tests are passing.
3. Get the latest code from source control. Resolve any merge conflicts.
4. Verify that the unit tests are still passing in the local development environment.
5. Check in the modified files. There should be no merge conflicts because you synchronized them in step 3.
6. Review the code for missing comments while the CI build is running. After the CI build completes and passes all the unit tests, you can move on to next task. If the CI build fails, fix the issue because you are now blocking other developers from integrating their code.

Tool for Running Continuous Integration Builds

The patterns & practices SharePoint Guidance team uses Microsoft Visual Studio Team System 2008 Team Foundation Server Team Build to configure and run all automated builds including CI, build verification tests, and

automated acceptance tests. Another popular build platform is [CruiseControl.NET](#). One of the most important features that both build platforms offer is a system tool that immediately issues alerts when builds break. This feature is named [Build Notification](#) in Team Build and CCTray in CruiseControl.NET.

 Note:

The build notification tool is only available after installing the Team Foundation Power Tools.

More Information

For an example of how to set up an automated build environment, see [How to: Create an Automated Build and Deployment Solution with Team Foundation Server Team Build](#). For information about using Team Build while developing SharePoint Applications, see [Team Development Overview](#). For an explanation of unit testing, see [Unit Testing with Mock Objects](#).

For other information about CI, see [Continuous Integration](#) on Martin Fowler's Web site.

For information about setting up CI with Team Build, see [TFS 2008: A basic guide to Team Build 2008](#) on Buck Hodges's blog.

How to: Create an Automated Build and Deployment Solution with Team Foundation Server Team Build

Microsoft Visual Studio Team System provides a mechanism to implement continuous building and testing using the Build function under a Visual Studio Team System project. The Build function sets up and runs MSBuild on a targeted computer that is running the Team Foundation Server build agent.

This topic walks you through setting up a Build definition and setting up a computer to run the defined build. First, it explains the SharePoint Web Solution Package, and then it demonstrates how to automatically build and deploy SharePoint solutions with Visual Studio extensions for Windows SharePoint Services version 1.3. Next, it shows you how to set up the Team Foundation Server build agent to run the continuous testing. Then, it defines the build and attaches the build definition to the Team Foundation Server build agent. The procedures in this topic require the following to be installed on your computer:

- Visual Studio Team System 2008
- Visual Studio extensions for Windows SharePoint Services version 1.3

The following sections summarize the steps that are required to automatically build and deploy the SharePoint solutions:

- [Understanding the Web Solution Package](#)
- [Building, Packaging, and Deploying the SharePoint Solution](#)
- [Creating a Team Foundation Server Build Agent](#)
- [Creating a TFSBuild.proj File](#)

The next sections describe these steps.

Understanding the Web Solution Package

A Web solution package (WSP) is the primary way to deploy custom SharePoint features and components in Windows SharePoint Services 3.0. A WSP is a CAB file that has a .wsp file name extension. It contains a manifest that is named Manifest.xml and all the SharePoint solution artifacts that are necessary for deploying a SharePoint solution.

You can manually create a Web solution package and create and maintain the manifest file and the feature files yourself if you do not want to use the Visual Studio extensions for Windows SharePoint Services 1.3. After you are ready to deploy your SharePoint solution, use the Makecab.exe tool (MakeCab) to create the WSP. For more information about how to manually create WSP files, see [Creating a Solution](#) on MSDN.

Both the Training Management and Partner Portal applications' SharePoint solutions are created with Visual Studio extensions for Windows SharePoint Services 1.3. This means that Visual Studio extensions for Windows SharePoint Services 1.3 generates the manifest and the feature XML files. One advantage of using Visual Studio extensions for Windows SharePoint Services 1.3 is that developers do not need to maintain a large number of XML files.

The rest of this topic discusses how to create an automated build and deployment system. The code for the pertinent files and targets is included within the appropriate sections.

Building, Packaging, and Deploying the SharePoint Solution

Building and deploying a SharePoint solution is simplified if you create your Visual Studio project with Visual Studio extensions for Windows SharePoint Services 1.3. First, click **Build Solution** on the Visual Studio **Build** menu. After the Visual Studio solution compiles, click **Deploy Solution** on the **Build** menu. This deploys your solution to the SharePoint site you specified at the Start Action "start browser with URL" in the Project Property Page Debug section.

You can also create the WSP files without actually deploying the solution. To do this, right-click the Visual Studio extensions for Windows SharePoint Services project, and then click **Package**. The package command creates the WSP file and the Setup.bat file in the bin\debug or bin\release folder of your project.

You can use a command line script to package your solution. You can package all the Visual Studio extensions for Windows SharePoint Services projects into a solution or you can package an individual project in a solution. The following command packages an entire solution that is named YourVstsSolution.sln:

```
devenv YourVstsSolution.sln /Deploy debug /Package
```

The following command packages an individual project as a solution:

```
devenv YourVstsSolution.sln /Deploy debug /Package YourVseWSSProject
```

Creating a Team Foundation Server Build Agent

For information about how to create a Team Foundation Server build agent, see [How to: Create and Manage Build Agents](#) on MSDN.

Creating a TFSBuild.proj File

For information about how to create a Team Foundation Server build project file, see [Walkthrough: Creating a Build Definition in Team Foundation Build](#) on MSDN.

An Example TFSBuild.proj File

The following code is the TFSBuild file for the Partner Portal application's BVTs. These tests run each night. The build file performs the following tasks:

1. It runs ClearContosoSetup.bat to remove data left over from the previous test run.
2. It removes the old source folder so it can get the new source files from Team Foundation Server.
3. It removes any read-only attributes in the sources files before it creates a solution package.
4. It runs ContosoSetup.bat to install the reference implementation.
5. It runs the test lists Partners, PartnerPortal, and Services in /Test/PartnerPortal/Contoso.PartnerPortal.BVT.vsmDI. This file is generated when the solution file is opened.

The following XML is the TFSBuild file.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="DesktopBuild"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003" ToolsVersion="3.5">
  <Import
Project="$(MSBuildExtensionsPath)\Microsoft\VisualStudio\TeamBuild\Microsoft.TeamFoundation.B
uild.targets" />
  <ProjectExtensions>
    <ProjectFileVersion>2</ProjectFileVersion>
    <Description></Description>
    <BuildMachine>SPG-2K8-BVT32</BuildMachine>
  </ProjectExtensions>
  <PropertyGroup>
    <TeamProject>SPG</TeamProject>
    <BuildDirectoryPath>C:\Documents and Settings\v-appave\Local
Settings\Temp\1\SPG\SPGv2-2k8-BVT32</BuildDirectoryPath>
    <DropLocation>\\SPG-2K8-BVT32\BuildStageFolder</DropLocation>
    <RunTest>true</RunTest>
    <RunCodeAnalysis>Never</RunCodeAnalysis>
    <WorkItemType>Bug</WorkItemType>
    <WorkItemFieldValues>System.Reason=Build Failure;System.Description=Start the build using
Team Build</WorkItemFieldValues>
    <WorkItemTitle>Build failure in build:</WorkItemTitle>
    <DescriptionText>This work item was created by Team Build on a build
failure.</DescriptionText>
    <BuildLogText>The build log file is at:</BuildLogText>
    <ErrorWarningLogText>The errors/warnings log file is at:</ErrorWarningLogText>
    <UpdateAssociatedWorkItems>true</UpdateAssociatedWorkItems>
    <AdditionalVCOVERRIDES></AdditionalVCOVERRIDES>
    <CustomPropertiesForClean></CustomPropertiesForClean>
    <CustomPropertiesForBuild></CustomPropertiesForBuild>
  </PropertyGroup>
  <ItemGroup>
    <SolutionToBuild
Include="$(BuildProjectFolderPath)/../../../../Setup/SetupSource/ContosoSetup.sln">
      <Targets></Targets>
      <Properties></Properties>
    </SolutionToBuild>
    <SolutionToBuild
Include="$(BuildProjectFolderPath)/../../../../Source/Microsoft.Practices.SPG2/Microsoft.Prac
tices.SPG2.sln">
      <Targets></Targets>
      <Properties></Properties>
    </SolutionToBuild>
    <SolutionToBuild
Include="$(BuildProjectFolderPath)/../../../../Test/Contoso.PartnerPortal.BVT/Contoso.Partner
Portal.BVT.sln">
      <Targets></Targets>
      <Properties></Properties>
    </SolutionToBuild>
  </ItemGroup>
  <ItemGroup>
    <ConfigurationToBuild Include="Release|Any CPU">
      <FlavorToBuild>Release</FlavorToBuild>
      <PlatformToBuild>Any CPU</PlatformToBuild>
    </ConfigurationToBuild>
  </ItemGroup>
  <ItemGroup>
    <MetaDataFile
Include="$(BuildProjectFolderPath)/../../../../Test/Contoso.PartnerPortal.BVT/Contoso.Partner
Portal.BVT.vsmDI">
      <TestList>Partners;PartnerPortal;Services</TestList>
    </MetaDataFile>
  </ItemGroup>
</Project>
```

```

</ItemGroup>
<PropertyGroup>
  <workspaceSetupFolderPath>C:\Bvt2\Trunk</workspaceSetupFolderPath>
</PropertyGroup>
<Target Name="BeforeEndToEndIteration">

  <Message Text="Step:1***** Run ClearContosoSetup.bat to clean previous test"/>
  <Exec Command="$(workspaceSetupFolderPath)\setup\ClearContosoSetup.bat"
IgnoreExitCode="true" />

  <Message Text="Step:2***** Remove Source folder so we can get new source files from
TFS"/>
  <RemoveDir Directories="C:\BVT2\Trunk\" ContinueOnError ="true"/>
</Target>
<Target Name="AfterCompile" Condition=" '$(IsDesktopBuild)'!='true'">

  <Message Text="Step:3***** Removing Read only attributes in sources files before create
package"/>
  <Exec WorkingDirectory="C:\BVT\Drops" Command="RemoveReadOnly.bat" />

  <Message Text="Step:4***** Executing ContosoSetup.bat to install contoso RI "/>
  <Exec Command="$(workspaceSetupFolderPath)\setup\ContosoSetup.bat" IgnoreExitCode="true"
/>
</Target>
</ItemGroup>
</ItemGroup>
</Project>

```

Application and Design Patterns

Patterns are general solutions to commonly occurring problems. For example, Microsoft .NET Framework events and delegates implement the Observer pattern. This section describes software patterns that are useful for SharePoint applications. The Partner Portal and Training Management applications demonstrate the use of these patterns. In addition, several patterns are implemented in the SharePoint Guidance Library.

The following topics describe the design and application patterns for SharePoint applications:

- [Application Patterns](#). Application patterns describe general solutions to commonly occurring business problems in enterprise-scale architectures.
- [Design Patterns](#). Design pattern are general solutions to commonly occurring software problems.

Application Patterns

Application patterns describe general solutions to commonly occurring business problems in enterprise-scale architectures. Application patterns are one layer above design patterns. The Partner Portal application implements the following application patterns:

- [Event-Driven Site Creation](#). This pattern describes a model for creating a site in response to an event.
- [Application Instance Resolution](#). This pattern describes a model that automatically directs users to the appropriate area for information partitioned across site collections.
- [Business Event Coordination](#). This pattern describes a model for coordinating events between external systems and SharePoint.

Business Event Collaboration Overview

A business event is a representation of something that happens while a line-of-business (LOB) system executes a business process. The event is usually a standard occurrence that requires no intervention, such as when a product is shipped to a customer. However, some events represent exceptions to the standard process and require more action. An example is a shipping delay that occurs because a truck breaks down. Although events of this type still involve some computational processing, they also often require human intervention to resolve them.

The shipping delay illustrates how a business's structured processes can be intertwined with informal processes. (Informal processes are the actions employees take to solve problems.) Typically, coordination between the two is missing in businesses today.

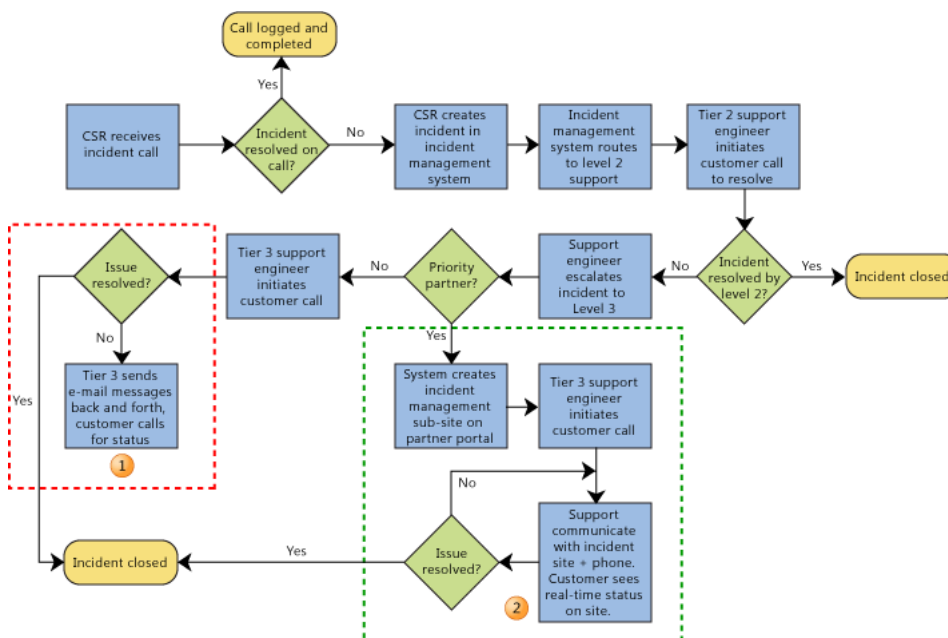
LOB systems support complex processes that are comprised of a series of procedural steps. For example, fulfilling an order can require integrated processes to build the product, package it, ship it, invoice for it, and receive payment. Companies use Enterprise Resource Planning (ERP), Material Requirements Planning (MRP), and logistical systems to manage these processes.

However, when a problem occurs, employees must often resolve the issue. They use telephone calls, send e-mail messages and documents, exchange opinions, and make decisions. For a delayed shipment, a customer representative may resolve the logistical issues and identify ways to compensate the customer for the late delivery. These decisions are later recorded in the LOB systems.

In some cases, informal interactions are part of the normal business process. A good example is a response to a proposal in a sales process, which can involve many people and conversations. The LOB system is the repository for the result that the informal but complex process produces.

The following illustration represents a simplified view of interactions that occur between formal and informal processes when a Contoso partner has a problem that escalates to the critical level of a tier three incident.

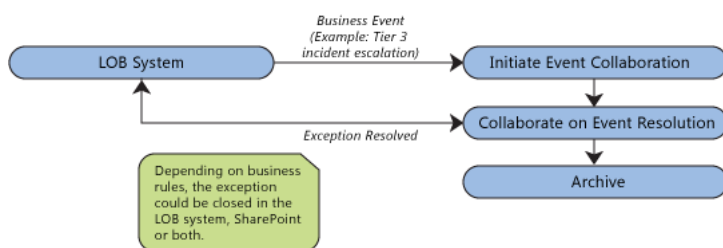
Interactions of processes



The area labeled 1 represents a typical informal process. Employees manage problems with phone calls, e-mail messages, and status updates that are relayed to the internal incident management system. Much of this activity is invisible to the customer.

The area labeled 2 represents a different approach. SharePoint automatically creates a collaboration space when the incident escalates to tier 3 based on an event from the incident management system. It displays data that is stored in the LOB system and centralizes the information from both the formal and informal processes. The following diagram illustrates the process of collaborating on a business event.

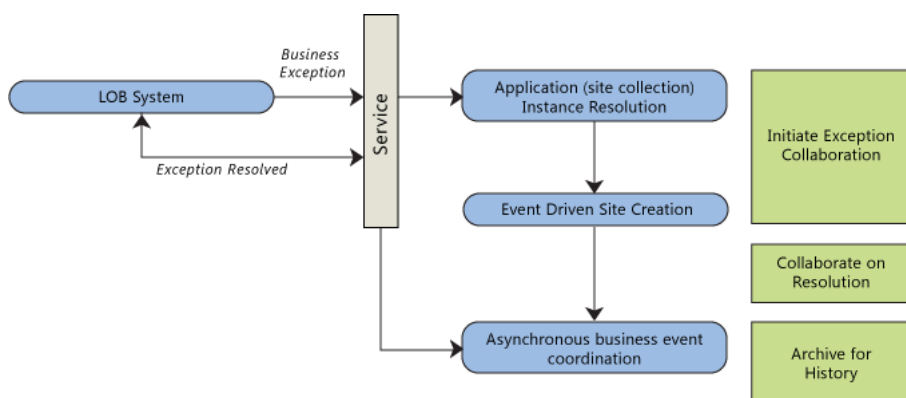
Collaborating on a business event



The diagram shows that either business rules within the LOB system or human action can trigger the collaboration. SharePoint creates a Web site where Contoso employees can work with the partner to resolve the problem. At some point, the incident is completed or closed. This action can happen in either the LOB system or on the collaboration system, depending on how the company structures its processes. For example, one company might decide that an incident closed is always closed in the LOB system because there is a set of policies that must be met and that are enforced by the system. Another company might decide that it always wants the customer to determine whether an incident is resolved. In this case, the incident is closed in the collaboration space. The end result is that either the LOB system or SharePoint must close the incident and inform the other system that it can do the same.

The following diagram shows how the Event-drive Site Creation pattern and the Business Exception Collaboration pattern satisfy the collaboration scenario. It illustrates a more complex case than what was described earlier. In this scenario, which is implemented in the Partner Portal application, the event can apply to one of several partners.

Patterns for collaboration



The combination of the Event-Driven Site Creation pattern and the Business Exception Collaboration pattern helps businesses solve the problem of integrating formal and informal processes. An additional pattern, the Multi-Site Resolution pattern, enhances the solution when multiple customers are involved in partitioned collaboration spaces.

The following sections describe each of these application patterns:

- [Event-Driven Site Creation](#)
- [Application Instance Resolution](#)
- [Business Event Coordination](#)

Event-Driven Site Creation

Context

There are business events in your organization that require collaboration between employees and/or customers. However, there is no central location for having discussions or storing documents. Activities that occur in the informal processes are unrelated to the LOB processes. When the event is resolved, the relevant information is scattered throughout the organization. You might have instituted a manual process for creating sites where people can collaborate on these incidents, but the sites are not standardized and there is no way to enforce business rules.

Objectives

- You want to automatically create collaboration sites that will help your company to efficiently resolve events.
- You want to allow users to request collaboration sites, but you must control when and where the sites are created.
- You want the information that people have gathered to be centrally organized and accessible.
- You want the ability to add, remove, and change tasks.

The next section describes how to achieve these objectives.

Solution

An implementation of the Event-driven Site Creation pattern creates a site that is appropriate for the particular type of event. For example, there can be one type of site for problems such as faulty equipment and another type of site for problems with orders. The site must be correctly located within the site hierarchy. Finally, the site must be populated with information that connects the incident with the site.

The following diagram summarizes these steps.

Event-driven site creation



An LOB system such as a customer relationship management (CRM) system tracks status and controls the incident management process. It stores and provides information such as the customer identifier, the incident identifier, the nature of the incident and the status of the incident. The collaboration site supplements the CRM system by providing a place to track tasks with the customer, exchange supporting documentation, and have discussions. It also allows the customer to monitor the status of the incident with information retrieved from the incident management system. The following sections discuss in detail the steps that are shown in the diagram.

Mapping a Site

Mapping a site determines two things:

- **It determines the location of the site within the site hierarchy.** Every collaboration site must be the child of a parent site.
- **It determines the type of site to create.** Effective collaboration sites are standardized for particular types of events. SharePoint has site templates and site definitions that allow you to create repeatable instances of a type of site. Mapping connects a business event to the appropriate type of site. As different events are identified, developers or business analysts can add new mappings.

After you create a site template or site definition, add a mapping definition that connects it to the event. For example, an event type of "tier 3 incident escalation" might map to the "Incident Site Definition." This type of subsite is created under the /partnersite/incidents location.

Creating a Site and Adding It as a Subsite

After the mappings are defined, the site must be created at the correct place in the site hierarchy. The wire and creation steps use the mapping to do this. For example, if a partner reports a problem with an X-ray machine's power supply, you can create an incident site named "Intermittent Power Delay in X-Ray Machine" that uses the Incident Site Definition and add it as a subsite of the parent site that is located at /partnersite/incidents.

Binding Information

Binding adds information to the site that relates it to the event. You can use this bound information to tie context between the collaboration site and the originator of the business event. There are different ways to bind identifying information. The simplest approach adds the information to the site's property bag. The Partner Portal application does this. Using the property bag is straightforward and works for many situations.

In the Contoso example, an incident identifier for the specific incident instance of "Intermittent Power Delay in X-Ray Machine" is placed in the site's property bag. A Web Part that is on the site uses the incident identifier and uses it to retrieve relevant information that is stored in the CRM system. It displays the results on the site. The partner who reported the incident, in addition to Contoso employees who are collaborating with the partner, can see real-time information about the incident's status.

In more complex scenarios, you could bind information at any location on the site. This can require that you develop a custom binding for the event.

Considerations

The Event-driven Site Creation pattern promotes design flexibility, control over site creation, and configurability in the infrastructure. The cost is increased operational complexity, at least in the short term. In the long term, operational complexity may decrease because the pattern fosters a consistent information architecture. This is because sites are created in a controlled fashion.

Site versioning can be complex. Event types that have multiple versions of the site template or site definition can lead to long-term maintenance issues. Use care when defining new site templates or site definitions. Put a control processes in place to make sure that the new versions are warranted. Neither site definitions nor site templates can be updated after they are deployed, although new capabilities may be added to a site definition through a mechanism named feature stapling. One approach is to wait until any active events that use the earlier version are resolved before you deploy the new version. Because business events are generally resolved within a fixed time frame, this strategy is often acceptable. If you use this approach, be careful not to break the features of the older version when the upgrade occurs.

The Event-driven Site Creation pattern can help create a site hierarchy that has a logical structure. A straightforward architecture is easy to partition, access, and organize. Although it is difficult to make a general rule that fits all situations, a parent/child relationship is a good choice for event-driven sites. For example, you can provide a dashboard for all incidents on the parent site that shows all open incidents to the customer. Another benefit is that customers always know to look under the parent site for information about a particular incident. An unplanned site hierarchy can make locating and organizing information difficult.

Application Instance Resolution

Context

Because of security or scale constraints, information in SharePoint often is logically partitioned between site collections. A set of site collections can make up a single logical group. The Application Instance Resolution pattern correctly identifies the location of information within this group.

For example, assume that a company maintains extranet workspaces for its partners. The company decides to partition the partners' workspaces into separate site collections. This allows the company to maintain security permissions and to distinguish the content intended for one partner from the content intended for another partner.

When information is partitioned, there must be a way to determine where a particular piece of information is located. In the scenario that is described in the [Event-Driven Site Creation](#) topic, one of the steps is to determine where a site should be created. Because it occurs automatically, the subsite creation process must be able to find the site collection of that partner to create and wire up the site in the correct partner collaboration space. Using identifying information to resolve the location of a partner's site collection streamlines the integration between the CRM system and the Partner Portal.

In other cases, you can provide the partners with a simple URL to go to and redirect to the correct partition instead of forcing users to use complex URLs. This prevents delays and errors. In the Partner Portal application, Contoso uses managed paths in the address of each partner's site collection. For example, Fabrikam, a partner of Contoso, uses the URL `http://extranet.contoso.com/partners/Fabrikam`. Without a redirect page, Fabrikam employees must remember the full URL every time they want to visit their collaboration portal. With a redirect page, partners can use a shorter address such as `http://extranet.contoso.com`.

Objectives

- You want to partition information into site collections to logically organize information and to enforce security.
- You want to programmatically add subsites to a site collection or you want to provide your customers with URLs that are easy to remember and shield them from the partitioning scheme for the site collections.

The next section shows how to achieve these objectives.

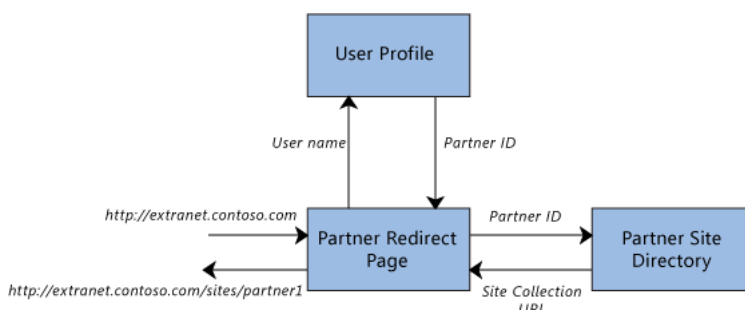
Solution

To implement the application instance resolution pattern, you must create a mapping that relates an entity with an URL. In the Contoso example, a partner is an entity. There is a mapping between the partner and a site collection. This mapping can be implemented as a SharePoint list. At a minimum, this list should contain a field for the entity that maps to a site collection and a field for the URL of a particular site collection.

Additionally, developers can use the Site Directory feature of Microsoft Office SharePoint Server 2007. The Site Directory is a site definition that is provided by Microsoft Office SharePoint Server. It maintains a central location for URLs of SharePoint sites and external links. The Site Directory is a list of URLs and related metadata, such as categories. It also provides a set of tabbed pages for displaying the URLs in different views. The Site Directory can be customized to include custom fields in the sites list and custom tabbed pages. For example a custom field for the partner identifier can be added to the Sites list in the Site Directory. By adding metadata in the form of fields to the sites list, you can extend the Site Directory into a mapping list that resolves sites. Although the site directory is often an appropriate choice when using Microsoft Office SharePoint Server, you could use a standard SharePoint list instead if you implement the pattern using Windows SharePoint Services.

After a list is defined, business logic must be in place to query the list. The following figure illustrates the Partner Portal implementation of the Application Resolution pattern.

Partner redirect example that uses the application instance resolution pattern



Considerations

In the Partner Portal application, the business logic that resolves a site collection is very simple. It queries the Sites list for a particular partner ID. However, if business requirements demand more complex business logic, a list may not be appropriate.

You should also consider security. One of the challenges of using a list as a mapping store is that users must be able to access it. A site collection mapping list contains sensitive information that should be restricted to administrators and those users who require access. There will be cases when a user with inadequate permissions must access the list. For example, on a partner redirect page, the business logic must query the Sites list to retrieve the site collection of the currently logged on user. In this case, the business logic must run with elevated privileges or a service must be created to retrieve this information on behalf of the user.

Business Event Coordination

SharePoint is a platform for managing content and collaborating on it. It does not replace LOB applications such as CRM systems. However, it does augment and support them. For example, components in a SharePoint collaboration site can coordinate with or expose information from external LOB systems. For this to occur, a piece of information that indicates the state of the information that is stored in the external system must be bound to the SharePoint site. When that status of the state changes based on activities in either SharePoint or the LOB system, the other system often needs to also take action based on the status change.

As an example, assume that SharePoint creates a subsite in response to an event from a CRM system. SharePoint provides a Web Part that exposes detailed information about the incident. This information comes from the CRM system. When a customer service representative closes the incident in the CRM system, this status is reflected in the SharePoint site and the incident is no longer shown as active.

There are different approaches to how to coordinate this event coordination information. A simple approach for storing information related to the business system, such as status for a site, is in its property bag. Property bags store metadata as key/value pairs. This makes them appropriate places to store information that uses this format, such as incident identifiers and incident status. The following figure illustrates how information travels from a CRM system to a SharePoint subsite.

Business exception collaboration pattern



A list should be used for more complex coordination interactions that need to bi-directionally synchronize activities between the LOB system and a SharePoint application. The guidance does not demonstrate coordinating events of this complexity in the Partner Portal application.

Design Patterns

A design pattern is a general, reusable solution to a commonly occurring software problem. A design pattern is not a finished specification that can be transformed directly into code. Instead, it is a template for solving a particular problem. Thinking in terms of design patterns allows you to formulate a high-level solution that is independent of the implementation details. Using established design patterns leads to a solution that is easily understandable, testable, and maintainable.

For example, the Partner Portal and Training Management applications use versions of the Model-View-Presenter (MVP) pattern. Designing the applications in an MVP pattern structures the applications' layers and increases the testability of the business logic that is in the Web Form code-behind files. Code-behind files are a convenient place to locate both the event handlers and the view logic. However, the event handlers are **private** methods, and the forms must be rendered to test the view logic. This makes it difficult to develop automated unit tests. Another problem is that, although it performs separate functions, the business logic is intermixed with the view logic. Factoring out the business logic into a separate presenter class allows you to unit test it in isolation and separates it from the view logic.

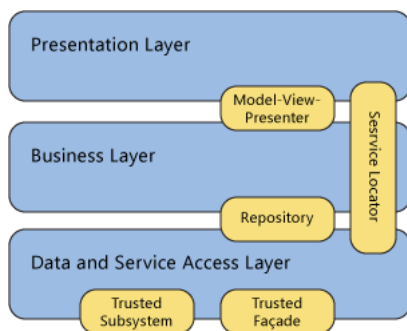
Enterprise-scale SharePoint applications include many cooperating services and application layers. The Partner Portal application implements several patterns that group related functions into discrete layers. As a result, the application is modular and supports the principle of separation of concerns. In turn, this encapsulation improves the application's testability.

The Partner Portal application implements the following patterns:

- [The Model-View-Presenter \(MVP\) Pattern](#)
- [The Repository Pattern](#)
- [The Service Locator Pattern](#)
- [The Trusted Façade Pattern](#)

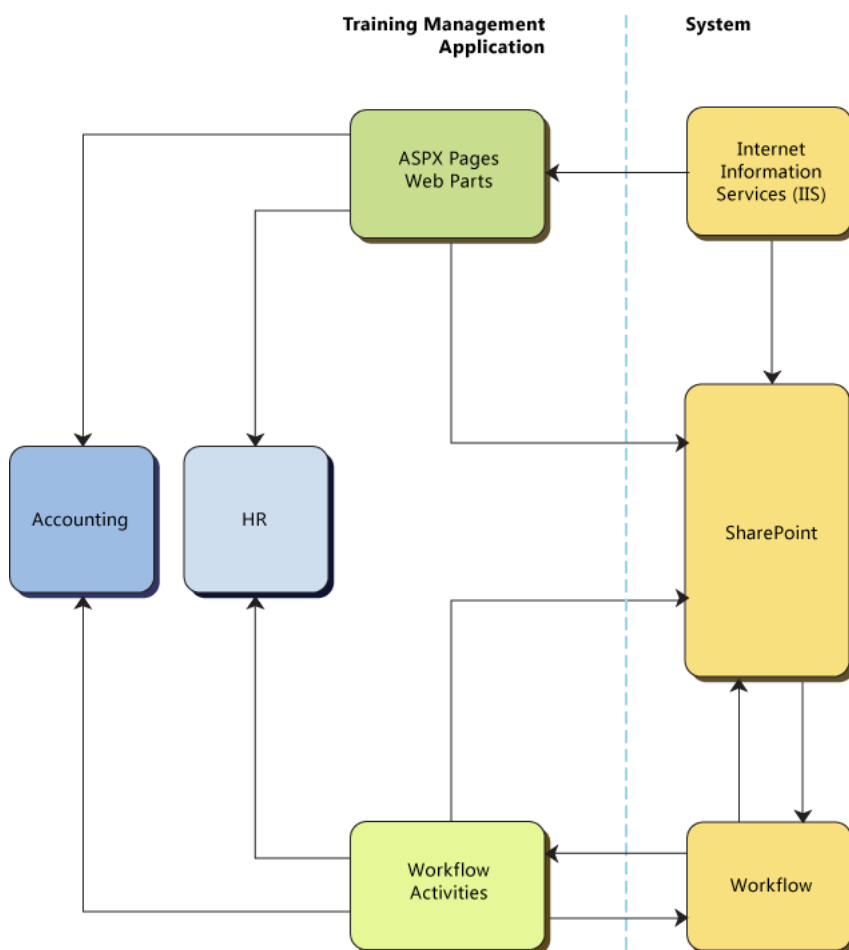
The following diagram illustrates the patterns that are used in the Partner Portal application and shows how they relate to the application layers.

Patterns used in the Partner Portal application



The Training Management application also shows how patterns can improve an application. The initial implementation did not use patterns. The following diagram illustrates the initial architecture of the Training Management application before it was refactored.

Unfactored Training Management application

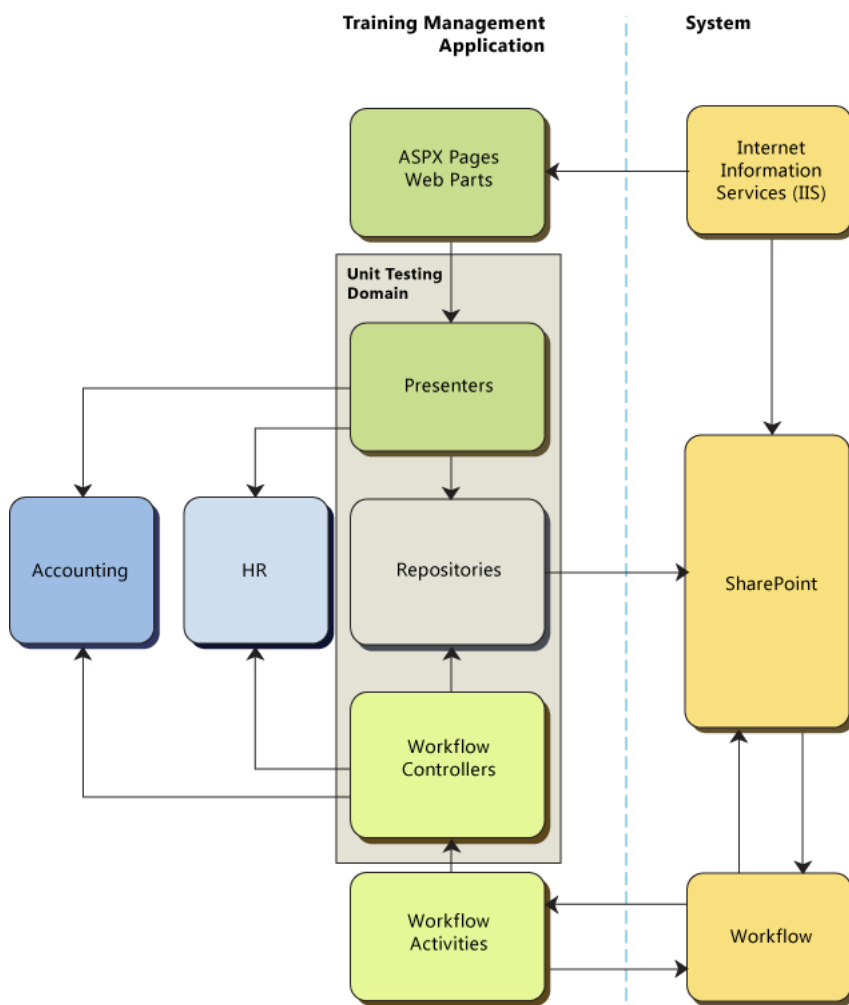


The initial implementation had the following problems:

- Both the .aspx pages and the Web Parts had view logic intermixed with business logic. This made it difficult to unit test the business logic.
- The ASPX Pages, the Web Parts, and the workflow activities had static references to service assemblies that were part of the line-of-business (LOB) systems. Fixed dependencies on LOB systems, which are typically not available in development or test environments, make unit testing difficult.
- The .aspx Pages, the Web Parts, and the workflow activities directly accessed data from SharePoint lists. This resulted in a great deal of duplicated list access code. Some of the code did not follow best practices for performance and for secure list access.

The following diagram illustrates the architecture of the Training Management application after it was refactored and reorganized according to the MVP pattern.

Factored Training Management application



The refactored application supports unit testing. The MVP pattern separates the view logic from the business logic. The SharePoint list repositories encapsulate the list access code, and the service locator places a layer of abstraction between the application and the dependent services. The end result produces results in increased test coverage of the business logic, and a better separation of the view and business logic. The refactoring and restructuring also help to remove duplicated logic and improve the long-term maintainability of the application.

The Model-View-Presenter (MVP) Pattern

Context

A Web page or Web Part in a SharePoint application contains controls that display application data. A user can modify the data and submit the changes. The Web page or Web Part retrieves the data, handles user events, alters other controls on the page in response to the events, and submits the changed data. Including the code that performs these functions in the Web page or Web Part makes them complex, difficult to maintain, and hard to test.

Objectives

- You want to maximize the amount of code that can be tested with automation. (Views are difficult to test.)
- You want to separate business logic from user interface (UI) logic to make the code easier to understand and maintain.

Solution

Separate the logic for the visual display from the event handling behavior by putting them into two classes that are named, respectively, the view and the presenter. The view (the Web page or Web Part) manages the controls on the Web page and forwards user events to a presenter. The presenter contains the logic to respond to the events, updates the model (both the business logic and the application data), and alters the state of the view.

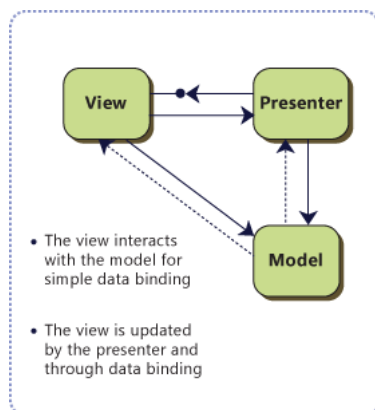
To make the presenter testable, define a view interface and have the presenter refer to the view interface instead of to the view implementation class. This allows you to replace the actual view with a substitute implementation for unit tests.

There are many variations to the MVP pattern. The Partner Portal application uses the [Supervising Presenter](#) approach (this is also named Supervising Controller).

In Supervising Presenter, the view interacts directly with the model to perform any simple data binding that can be declaratively defined, without the presenter. The presenter updates the model. It only changes the state of the view when complex UI logic that cannot be declaratively specified is required. Examples of complex UI logic might include changing the color of a control or dynamically hiding or showing controls. The Partner Portal application has many examples of MVP that generally follow the Supervising Presenter pattern.

The following diagram shows the logical representation of the Supervising Presenter approach.

Supervising Presenter



Compared to other variations of MVP, such as Passive View, the Supervising Presenter pattern makes simpler code a higher priority than complete testability. The Supervising Presenter pattern requires less code than other MVP patterns because it uses data binding. The code is easier to maintain because simple UI changes do not require code changes in the presenter to update the view. For a comparison of passive view and supervising presenter approaches, see [Model-View-Presenter](#).

Note:

Most descriptions of the MVP pattern describe how the model can notify the view or the presenter if something changes in the model. This behavior is usually relevant for rich-client applications so it is omitted here. In rich clients, the model is often kept in memory. As the user interacts with the application, the model is updated and it notifies the UI. In a Web application, this scenario is unlikely because of the stateless request/response nature of Web requests. Most of the time, the information is re-retrieved by the presenter from the model for every Web request. This usually eliminates the need for the presenters or views to be updated based on changes to the model.

If you need presenters and views to be notified of changes to the model, you can implement the Observer pattern. For more information, see [Related Patterns](#).

Implementation Details

An application usually has a model in place that represents the domain business logic. You can assume this for the purposes of the following example.

The first step in implementing the MVP pattern is to create a view. Typically, you want to create an interface for the view so that you can replace it with a test implementation. The following code is an example of a simple view interface that retrieves and displays parts for a product.

```
C#
public interface IRelatedPartsView
{
    IEnumerable<Part> Parts { get; set; }
    void DataBind();
    string ErrorMessage { get; set; }
}
```

The view is implemented by a user control's code-behind. This is shown in the following code.

```
C#
public partial class RelatedPartsControl : System.Web.UI.UserControl , IRelatedPartsView
{
    private RelatedPartsPresenter presenter;

    public RelatedPartsControl()
    {
        presenter = new RelatedPartsPresenter(this);
    }

    // ...

    public IEnumerable<Part> Parts
    {
        get
        {
            return this.PartsRepeater as IEnumerable<Part>;
        }
        set
        {
            this.PartsRepeater.DataSource = value;
        }
    }

    // ...

    public string ErrorMessage { get; set; }

    protected void LoadPartsButton_Click(object sender, EventArgs e)
    {
        this.presenter.LoadParts(this.Sku);
    }
}
```

Because the view directly constructs the presenter, it is bound to the presenter implementation. This is generally acceptable because the view is a relatively simple class that will not be tested, and the presenter is unlikely to be replaced. The view changes either when the presenter sets information on the view or when events are raised as a result of user actions. An example of the first case is when the presenter sets the **Parts** property. An example of the second case is when a user selects the **LoadPartsButton**. This action forwards the event to the presenter by calling the **LoadParts** method. In either case, all the business logic is located in the presenter.

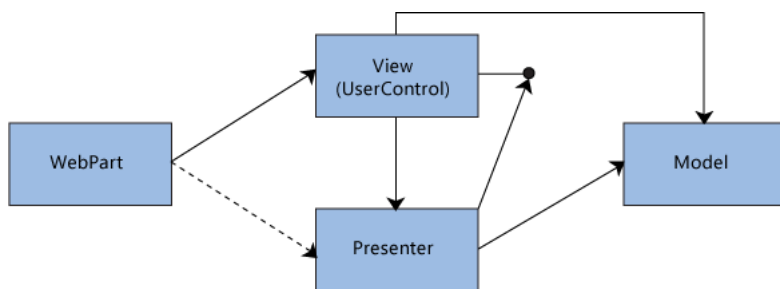
Partner Portal Application and the MVP

The Partner Portal application demonstrates two variants of the Supervising Presenter pattern. In one variant, a user control is contained within a Web Part. It is the user control that implements the view. The view constructs the presenter internally and calls methods on the presenter. This means that the presenter is hidden from the Web Part. The advantage to this approach is its simplicity, because the Web Part has no need to know anything about the presenter. The disadvantage is that any data that the Web Part wants to forward to the presenter must be forwarded through the view.

The **PartnerRollupWebPart** class demonstrates another variant on the Supervising Presenter. Here, the Web Part contains a user control to implement the view, and it also interacts directly with the presenter. The **PartnerRollupWebPart** instance constructs the view and passes the view into the presenter's constructor. The **PartnerRollupWebPart** instance can then set properties and call methods that are related to the business logic instead of calling these methods or properties on the view and having the view pass them to the presenter.

The following diagram shows the two Supervising Presenter variants. The solid lines represent the case where the Web Part does not interact directly with the presenter. The solid lines together with the dotted line represent the case where the Web Part does interact directly with the presenter.

Two variants on the Supervising Presenter pattern



Typically, user controls are the basis for developing Web Parts during design time. This means that the view logic is more commonly used in the user control and the user control is contained in the Web Part. In the Partner Portal application, the Web Part gathers context that is then passed either to the view (an instance of the user control) or directly to the presenter. In the case where the Web Part does not contain a user control that implements the view, the Web Part implements the view interface and directly interacts with the presenter.

The presenter only interacts with the view interface. This decouples the presenter from the view implementation. The view is passed into the presenter when the presenter is constructed. This approach is sometimes referred to as constructor injection. Because the view is passed to the presenter, you can also provide a mock view for unit tests. The following code demonstrates the constructor injection approach.

C#

```

public class RelatedPartsPresenter
{
    IRelatedPartsView view;

    public RelatedPartsPresenter(IRelatedPartsView view)
    {
        this.view = view;
    }

    public void LoadParts(string sku)
    {
        try
        {
            IProductCatalogRepository productCatalogRepository =
                SharePointServiceLocator.Current.GetInstance<IProductCatalogRepository>();

            IEnumerable<Part> parts = productCatalogRepository.GetPartsByProductSku(sku);
            if (parts != null && parts.Any())
            {
                view.Parts = parts;
            }
            else
            {
                view.ErrorMessage = Resources.NoPartsFoundError;
            }
            view.DataBind();
        }
        // ...
    }
}
  
```

The presenter retrieves the model (this is the list of parts) from the product catalog repository. The presenter then sets the model into the view and tells the view to perform a **DataBind** operation. The code also demonstrates the use of the Service Locator pattern to retrieve the repository. The service locator improves the application's testability. For more information, see [The Service Locator Pattern](#).

Considerations

There are several issues to consider if you want to use the MVP pattern:

- There are more solution elements to manage.
- You need a way to create and connect views and presenters.
- The model is not aware of the presenter. If the model is changed by any component other than the presenter, the presenter must be notified. Typically, notification is implemented with events. This is not a common scenario in a Web application.

The MVP pattern in the Partner Portal and Training Management applications is primarily for unit testing. An added benefit is the ability to separate each type of logic into its own layer. Both applications combine the MVP pattern and the Repository pattern with a software development technique named *unit testing with mock objects*. This technique breaks the tests' dependencies on external components, such as SharePoint. The mock objects emulate SharePoint behaviors during the unit tests. Mock objects substitute for views and repositories during the unit tests.

For more information about mock objects, see Martin Fowler's article [Mocks Aren't Stubs](#). For guidance about unit testing, see [Unit Testing with Mock Objects](#) in this documentation. For a discussion of the MVP pattern, see Derek

Greer's discussion, [Interactive Application Architecture Patterns](#).

Related Patterns

- [Application Controller](#). If presenters interact with an application controller, the presenters do not need page flow and screen navigation logic. This makes it easier to update the page flow.
- [Model-View-Controller \(MVC\)](#). The MVP and MVC patterns have similar goals, although there are differences in how they achieve those goals.
- [Observer](#). In this pattern, one object can observe changes on another object without referencing the implementation of the object being observed.

The Repository Pattern

Context

In many applications, the business logic accesses data from data stores such as databases, SharePoint lists, or Web services. Directly accessing the data can result in the following:

- Duplicated code
- A higher potential for programming errors
- Weak typing of the business data
- Difficulty in centralizing data-related policies such as caching
- An inability to easily test the business logic in isolation from external dependencies

Objectives

Use the Repository pattern to achieve one or more of the following objectives:

- You want to maximize the amount of code that can be tested with automation and to isolate the data layer to support unit testing.
- You access the data source from many locations and want to apply centrally managed, consistent access rules and logic.
- You want to implement and centralize a caching strategy for the data source.
- You want to improve the code's maintainability and readability by separating business logic from data or service access logic.
- You want to use business entities that are strongly typed so that you can identify problems at compile time instead of at run time.
- You want to associate a behavior with the related data. For example, you want to calculate fields or enforce complex relationships or business rules between the data elements within an entity.
- You want to apply a domain model to simplify complex business logic.

Solution

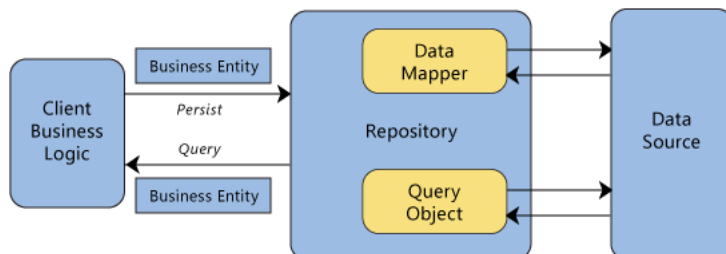
Use a repository to separate the logic that retrieves the data and maps it to the entity model from the business logic that acts on the model. The business logic should be agnostic to the type of data that comprises the data source layer. For example, the data source layer can be a database, a SharePoint list, or a Web service.

The repository mediates between the data source layer and the business layers of the application. It queries the data source for the data, maps the data from the data source to a business entity, and persists changes in the business entity to the data source. A repository separates the business logic from the interactions with the underlying data source or Web service. The separation between the data and business tiers has three benefits:

- It centralizes the data logic or Web service access logic.
- It provides a substitution point for the unit tests.
- It provides a flexible architecture that can be adapted as the overall design of the application evolves.

There are two ways that the repository can query business entities. It can submit a query object to the client's business logic or it can use methods that specify the business criteria. In the latter case, the repository forms the query on the client's behalf. The repository returns a matching set of entities that satisfy the query. The following diagram shows the interactions of the repository with the client and the data source.

Interactions of the repository



The client submits new or changed entities to the repository for persistence. In more complex situations, the client business logic can use the Unit of Work pattern. This pattern demonstrates how to encapsulate several related operations that should be consistent with each other or that have related dependencies. The encapsulated items are sent to the repository for update or delete actions. This guidance does not include an example of the Unit of Work pattern. For more information, see [Unit of Work](#) on [Martin Fowler's](#) Web site.

Repositories are bridges between data and operations that are in different domains. A common case is mapping from a domain where data is weakly typed, such as a database or SharePoint list, into a domain where objects are strongly typed, such as a domain entity model. One example is a database that uses **IDbCommand** objects to execute queries and returns **IDataReader** objects. Another example is SharePoint, which uses **SPQuery** objects to return **SPListItem** collections. A repository issues the appropriate queries to the data source, and then it maps

the result sets to the externally exposed business entities. Repositories often use the [Data Mapper](#) pattern to translate between representations. Repositories remove dependencies that the calling clients have on specific technologies. For example, if a client calls a catalog repository to retrieve some product data, it only needs to use the catalog repository interface. For example, the client does not need to know if the product information is retrieved with SQL queries to a database or Collaborative Application Markup Language (CAML) queries to a SharePoint list. Isolating these types of dependences provides flexibility to evolve implementations.

Implementation Details

This section discusses the implementation strategies for SharePoint list repositories and Web service repositories.

SharePoint List Repositories

The following diagram illustrates the interactions of a SharePoint list repository with SharePoint lists and the business logic.

Interactions of a SharePoint list repository



Using the Repository pattern in a SharePoint application addresses several concerns.

- SharePoint applications often store business information in SharePoint lists. To retrieve data from SharePoint lists requires careful use of the SharePoint API, knowledge of the GUIDs that are related to the lists and their fields, and a working knowledge of CAML. Repositories centralize this logic.
- The amount of code that is required to query or update a SharePoint list item is enough to warrant its encapsulation into helper methods. When Web Forms, event receivers, and workflow business logic all require access to the same lists, the code that accesses the SharePoint lists can be duplicated throughout the application. This can make the application prone to bugs and difficult to maintain. Repositories eliminate this duplication.
- Without a repository, the application is difficult to unit test because the business logic has direct dependencies on the SharePoint lists. Repositories centralize the access logic and provide a substitution point for the unit tests.

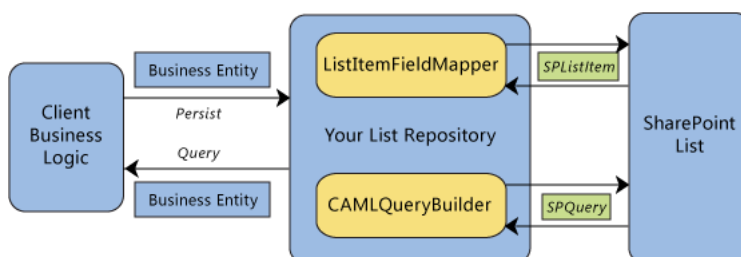
Externally, the repository exposes strongly-typed business entities. Internally, it works with SharePoint-specific objects, such as the **SPQuery** and the **SPListItem** objects. The SharePoint Guidance Library, which is a part of this guidance, provides classes for mapping and querying that make it easier to build repositories for SharePoint lists. The **ListItemFieldMapper** class converts strongly-typed business entities to and from **SPListItem** objects based on a set of mapping definitions. The **CAMLQueryBuilder** class builds **SPQuery** objects based on common query operations. The **SPQuery** object is used to query a SharePoint list.

The following sections show how the repository pattern is implemented in the SharePoint Guidance Library. For more information, see [List-Based Repositories](#).

SharePoint Guidance Library Helper Classes

The following diagram shows the major components of a SharePoint list repository.

Components of a SharePoint list repository



The list repository contains a query object and a data mapper object that are specific to SharePoint. These are the **ListItemFieldMapper** and **CAMLQueryBuilder** classes. The data mapper translates between an **SPListItem** and the business entity that is defined by the application. The query object internally constructs an **SPQuery** object and uses CAML to query the list.

Note:

When you design a SharePoint list repository, keep in mind that a list can contain fields from multiple content types. The logic that is implemented in the **ListItemFieldMapper** and **CAMLQueryBuilder** objects does not prevent fields from multiple content types from being retrieved. In some cases, if the content types have the same parent content type, you can use a single repository to project a common view across these content types.

However, it is generally inadvisable to create repositories that deal with dissimilar content types and return different business entities from the same repository. In this situation, create a repository for each content type because the content types logically represent different entities.

Implementation Variations

When you create a SharePoint list repository, you should consider how the repository locates the list that it is going to access. A list typically resides in a site, and it can be accessed either through its Uniform Resource Identifier (URI) or its GUID. The repository needs one of these, but passing this information to a repository can be challenging if you use the repository in conjunction with a service location. For more information, see [The Service Locator Pattern](#).

There are three ways in which a repository can access a list:

- **A list can be centrally located.** In this case, a repository is associated with a list that is at a fixed location. All sites retrieve the data from this central location. The **PartnerPromotionsRepository** class in the Partner Portal application is an example of a repository that uses such as list.
- **A list can be accessed relative to the current site context.** In this case, a repository is associated with a list whose location is relative to the current site. The **IncidentManagementRepository** class in the Partner Portal application is an example of such a repository.
- **A list can be accessed according to a context that is supplied by a consumer.** In this case, only the consumer of the repository knows which list the repository should access. There is no example of this in the Partner Portal application. However, you can extend the Training Management application to support this use case. You can implement a repository that accesses the list of training courses for your department on your local installation and also accesses related training courses that are located on another departmental site. In this case, the consumer instructs the repository to target a particular list.

The following sections describe more details about how to associate repositories with lists.

Lists That Are Centrally Accessed from a Fixed Location

In this case, a list is at a fixed location, and all sites access it from this central point. Its location cannot be determined based on the current context. Although it is possible to hard code the location of the list, this is not recommended, because the topology of the site can change. It is often better to make the location of the list a configuration setting. In that case, defining the list's location is an administrative task. The location is established when the site topology is set up.

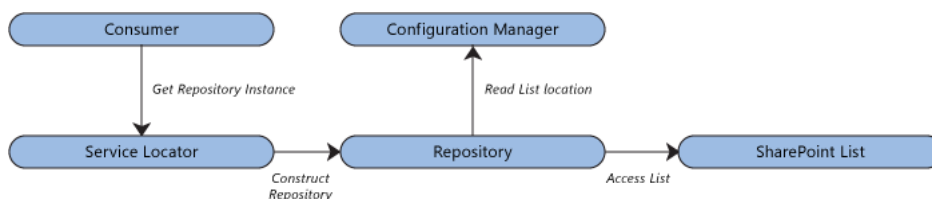
For example, the Partner Portal application centrally manages the published promotions for all partners by locating them on one site collection. Partners see their particular promotions on their collaboration home pages. Each partner collaboration site is hosted in its own site collection. This establishes security boundaries and isolates the data intended for one partner from the data intended for another partner. Because the relationship between the list and consumers of the list is based on the operational topology, the list location is defined with configuration data.

The following are characteristics of a list with a fixed location:

- There is typically only one instance of that list within the Web application scope.
- The location of the list is determined when the site topology is designed or when the site is installed.
- The list location should be retrieved from configuration data that is shared by all consumers. This data is typically at the Web application level or Web farm level.

The following diagram shows the flow of information among components that access a list at a central location.

Associating a repository with a list at a central location



The service locator constructs a repository object, which then reads the configuration information. The repository accesses the list based on this data. Because the repository relies on the configuration data, it can be constructed independently of the application context. It does not need any additional information from the list consumers.

This approach is susceptible to run-time exceptions because it depends on configuration data, which can be erroneous, lost, or corrupted. Make sure that you provide adequate diagnostics that inform IT administrators of any configuration errors. Problems that are caused by configuration errors are difficult to resolve without adequate logging information.

Lists with a Location That Is Fixed, Relative to the Current Context

In this case, the repository is associated with a list whose location is fixed, relative to the current context. For example, in the Training Management application, registration and course lists are located at the same relative location within a site. However, there can be several Training Management sites within a SharePoint farm. In this situation, the list repository is loaded from the current context.

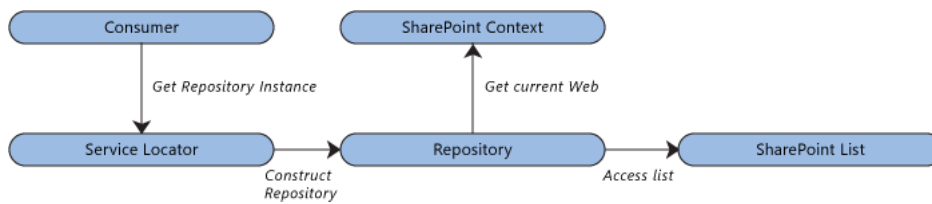
The following are characteristics of a list with a location that is relative to the context:

- The list has a fixed location that is relative to a site (an **SPWeb** object).
- The location is independent of the site topology.
- The list location is based on the current SharePoint context.

The following diagram shows the components and flow of information that access a list with a location that is

relative to the context.

Associating a repository with a list whose location is relative to the context



SharePoint often has a number of instances of the same Web application. In this situation, the repository gets the current site from the SharePoint context (the **SPContext.Current.Web** object) and loads the list information from this context. Because this relationship is fixed relative to the current site, the repository needs no additional information. The repository instance can be directly constructed by the service locator.

Lists Whose Context Is Supplied by a Consumer

In this case, the repository is associated with a particular content type and can access any list that has **SPListItems** of this content type. The consumer must provide context to the repository. For example, the consumer might provide the **SPWeb** object that holds the list or the GUID of the list. Although it is generally a good practice to keep technology-specific dependencies (such as a reliance on SQL Server) out of the repository interface, providing context when the repository is constructed is an accepted, widely used practice.

This scenario occurs with sites that have a dynamic topology, or where relationships are established by a user who supplies configuration information. If you add or remove sites at run time that contain lists that the repository accesses, you often have to provide the context. An example is if you use the Finance training site to view courses but you also want to see the courses on the Human Resources training site.

To provide this capability, you can build a general purpose Web Part to view the courses from other departmental training sites. You can add this Web Part to the Finance training site and configure it to view the related Human Resources department courses. The repository that the Web Part on the Finance site uses receives the location of the Human Resources course list as context information when the Web Part constructs it.

One challenge with this type of repository is using it in combination with the SharePoint Guidance Library service locator. The **ActivatingServiceLocator** class can only use parameterless constructors for the repositories. It is not possible to pass the contextual information (in this case, the location of the list) into the repository through the constructor. One way to solve this is to pass the location of the list with each method call, but this inserts a dependency on the list's URL into the interface definition. The Training Management application uses this approach.

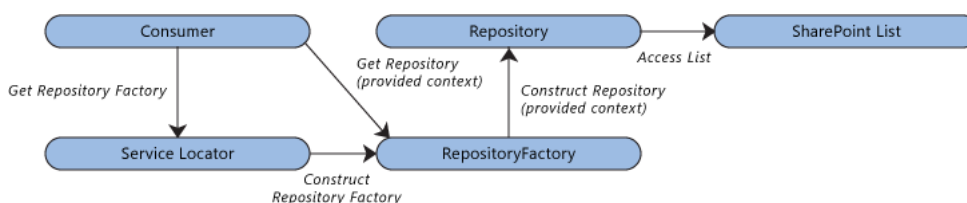
A better, but more complicated way to pass the location of the list to the repository is to use a factory. The factory includes a method that creates the repository. The consumer passes the location of the list to the method. The consumer then uses the **ActivatingServiceLocator** to access the factory and uses it to create the repository. With this approach, the consumer provides the location of the list to the factory, which in turn creates the repository. The factory passes the context through the repository constructor. This technique is known as constructor injection.

The following are characteristics of a list whose context is supplied by a consumer:

- The consumer can determine which list the repository should access.
- The location of the list is often determined at run time.
- This list location is derived from the current business context.

The following diagram shows the components and flow of information that are involved in accessing a list whose context is supplied by the consumer.

Associating a repository with a list whose context is supplied by the consumer



The consumer constructs the context for the repository. The consumer retrieves an instance of a repository factory from the service locator. The consumer then uses the repository factory to construct the repository. The consumer provides the context for the list. The repository uses this information to locate the list. Because the repository is decoupled from both the configuration data and the context, it is suitable for many scenarios. However, because the consumer provides the context, it increases the coupling between the consuming code and the repository.

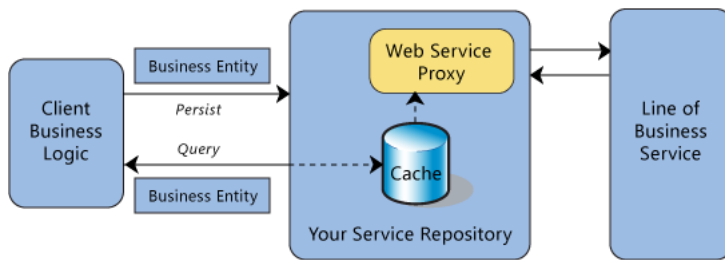
Web Service Repositories

A common backing store for data is a business service that is exposed by a line-of-business (LOB) application. Generally, these business services are at a higher level of abstraction than the standard Create/Read/Update/Delete (CRUD) semantics of a database or SharePoint list. However, from the perspective of the client, they often are equivalent to a data source. Like with SharePoint lists, accessing Web services can be

complex and prone to error. A repository centralizes the access logic for a service and provides a substitution point for unit tests. Note that services are often expensive to invoke and benefit from caching strategies that are implemented within the repository.

The following diagram shows a service back-end repository that uses caching.

Using a repository with a Web service



In this case, the query logic in the repository first checks to see whether the queried items are in the cache. If they are not, the repository accesses the Web service to retrieve the information. Although it is possible to access services directly, it is also possible to access them through the SharePoint Business Data Catalog (BDC). The BDC can aggregate several data sources, including Web services, and expose them through a uniform, generic interface. The BDC allows you to use standard Web Parts to display and modify data. For more information, see [Consuming Web Services with the Business Data Catalog \(BDC\)](#).

You may need more complex security options than the BDC supports. In this situation, you can use the Windows Communication Foundation (WCF). This requires that your own code and configuration data manage the service information and security context. For more information, see [Integrating Line-of-Business Systems](#).

Repository Examples

For an example of the list repository pattern, see [Development How-to Topics](#). Also, the Partner Portal application includes the following list repositories that can be used as starting points:

- The Partner Promotion Repository is in the `PartnerPromotionRepository.cs` file of the `PartnerPortal\Contoso.PartnerPortal.Promotions` directory. There is also a mock implementation for unit testing in the `PartnerPromotionsPresenterFixture.cs` file of the `PartnerPortal\Contoso.PartnerPortal.Promotions.Tests` directory.
- The Business Event Type Configuration Repository is in the `BusinessEventTypeConfigurationRepository.cs` file of the `Microsoft.Practices.SPG2\Microsoft.Practices.SPG.SubSiteCreation\BusinessEventTypeConfiguration` directory. There is also a mock implementation for unit testing in the `ResolveSiteTemplateFixture.cs` file of the `Microsoft.Practices.SPG2\Microsoft.Practices.SPG.SubSiteCreation.Tests` directory.
- The Subsite Creation Requests Repository is in the `SubSiteCreationRequestsRepository.cs` file of the directory `Microsoft.Practices.SPG2\Microsoft.Practices.SPG.SubSiteCreation\SubSiteCreationRequests`.

For an example of the data repository pattern using Web services, see the following areas of the reference implementation:

- The Incident Management Repository is in the `IncidentManagementRepository.cs` file of the directory `PartnerPortal\Contoso.LOB.Services.Client\Repositories`.
- The Pricing Repository is in the `PricingRepository.cs` file of the directory `PartnerPortal\Contoso.LOB.Services.Client\Repositories`.
- The Cached BDC Product Catalog Repository is in the `CachedBdcProductCatalogRepository.cs` file of the directory `PartnerPortal\Contoso.LOB.Services.Client\Repositories`. There is also a mock implementation for unit testing in the `ProductDetailsPresenterFixture.cs` file of the directory `PartnerPortal\Contoso.PartnerPortal.ProductCatalog.Tests`.

The Partner Portal application also contains two other repositories:

- The Full Text Search IncidentTask Repository uses SharePoint Search as its data source. This repository is found in the `FullTextSearchIncidentTaskRepository.cs` file of the directory `PartnerPortal\Contoso.PartnerPortal.Collaboration.Incident\Repositories`.
- The Partner Site Directory uses the site directory list to provide the Partner site collection URL and the user profile to provide the PartnerID. The repository is implemented in the `PartnerSiteDirectory.cs` file of the directory `PartnerPortal\Contoso.PartnerPortal.PartnerDirectory`.

For more information about the Repository pattern, Unit of Work pattern, and Data Mapper pattern, see [Repository](#) on [Martin Fowler's](#) Web site.

Considerations

The Repository pattern increases the level of abstraction in your code. This may make the code more difficult to understand for developers who are unfamiliar with the pattern. Although implementing the pattern reduces the amount of redundant code, it generally increases the number of classes that must be maintained.

The Repository pattern helps to isolate both the service and the list access code. Isolation makes it easier to treat them as independent services and to replace them with mock objects in unit tests. Typically, it is difficult to unit test the repositories themselves, so it is often better to write integration tests for them.

When caching data in a multithreaded environment, consider synchronizing access to the cache in addition to the cached objects. Often, common caches, such as the ASP.NET cache, are already thread safe, but you must also ensure that the objects themselves can operate in a multithreaded environment.

If you are caching data in heavily loaded systems, performance can be an issue. Consider synchronizing access to the data source. This ensures that only a single request for the data is issued to the list or back-end service. All other clients rely on the retrieved data. For more information, see [Techniques for Aggregating List and Site Information](#).

Related Patterns

The following two patterns are often used in conjunction with the Repository pattern:

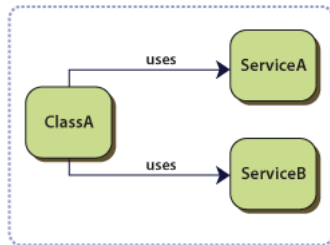
- [Data Mapper](#). This pattern describes how to map data to different schemas. It is often used to map between a data store and a domain model.
- [Unit of Work](#). This pattern keeps track of everything that happens during a business transaction that affects the database. At the conclusion of the transaction, it determines how to update the database to conform to the changes.

The Service Locator Pattern

Context

You have classes with dependencies on services whose concrete types are specified at compile time. In the following example, **ClassA** has compile time dependencies on **ServiceA** and **ServiceB**. The following diagram illustrates this.

Classes with dependencies on services



This situation has the following drawbacks:

- To replace or update the dependencies, you must change your classes' source code and recompile the solution.
- The concrete implementation of the dependencies must be available at compile time.
- Your classes are difficult to test in isolation because they have a direct reference to their dependencies. This means that these dependencies cannot be replaced with stubs or mock objects.
- Your classes contain repetitive code for creating, locating, and managing their dependencies.

The next section describes how to address these issues.

Objectives

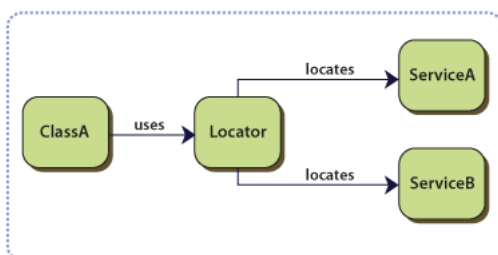
Use the Service Locator pattern to achieve any of the following objectives:

- You want to decouple your classes from their dependencies so that these dependencies can be replaced or updated with little or no change to the classes.
- You want to write logic that depends on classes whose concrete implementation is not known at compile time.
- You want to be able to test your classes in isolation, without the dependencies.
- You do not want the logic that locates and manages the dependencies to be in your classes.
- You want to divide your application into loosely coupled modules that can be independently developed, tested, versioned, and deployed.

Solution

Create a service locator that contains references to the services and that encapsulates the logic that locates them. In your classes, use the service locator to obtain service instances. The following diagram illustrates how classes use a service locator.

How classes use a service locator



The Service Locator pattern does not describe how to instantiate the services. It describes a way to register services and locate them. Typically, the Service Locator pattern is combined with the Factory pattern and/or the Dependency Injection pattern. This combination allows a service locator to create instances of services.

Note:

A service locator should be able to locate a service without knowing its concrete type. For example, it might use a string key or a service interface type. This allows you to replace the concrete implementation of the dependency without modifying the classes.

Implementation Details

The SharePoint Guidance Library offers an implementation of the Service Locator pattern. The

SharePointServiceLocator class provides access to a singleton **IServiceLocator** instance and manages that instance. The **SharePointServiceLocator** class includes a default implementation of the interface. This is the **ActivatingServiceLocator** class. This class can both create and locate services.

The Partner Portal application shows how to use the Service Locator to register and locate services such as repositories, logging services, and configuration management services. For more information, see [The SharePoint Service Locator](#).

Considerations

Consider the following points before you use the Service Locator pattern:

- There are more solution elements to manage.
- You must write additional code that adds service references to the service locator before your objects can use it.
- Your classes have a dependency on the service locator.
- The source code is more complex and difficult to understand.
- You can use configuration data to define run-time relationships.
- You must provide implementations of the services. Because the Service Locator pattern decouples service consumers from service providers, it might be necessary to provide additional logic. This logic ensures that the service providers are installed and registered before service consumers try to locate them.

Related Patterns

The following patterns are related to the Service Locator pattern:

- [Dependency Injection](#). This pattern solves the same problems as the Service Locator pattern, but it uses a different approach.
- [Inversion of Control](#). The Service Locator pattern is a specialized version of this pattern. It inverts an application's traditional flow of control. It is the object that is called instead of the caller that controls a process.

More Information

For more information about the Service Locator pattern, see the following articles:

- [Inversion of Control and the Dependency Injection pattern](#) on Martin Fowler's Web site
- [Service Locator](#) on MSDN

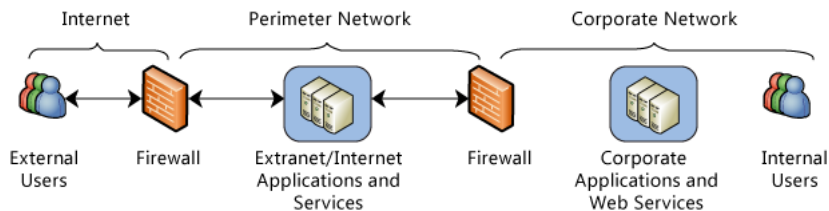
For more information about the Service Locator, see [The SharePoint Service Locator](#).

The Trusted Façade Pattern

Context

You want to authenticate external users that access a client application. Both the users and the application are outside the corporate firewall. The application is in the perimeter network (also known as DMZ, demilitarized zone, and screened subnet). It uses a Web service that is behind the firewall, in the corporate network. The application and the Web service also use different credential stores and authentication mechanisms. The following diagram shows the network topology.

Topology of the network



In the example that follows, the application in the perimeter network is a SharePoint application. The SharePoint application authenticates the external users. It also exposes information and operations that come from the corporate Web service to those authenticated users. The corporate Web service must apply authorization rules based on the identities of the external users.

Objectives

Use the Trusted Façade pattern with a SharePoint application to do the following:

- Authenticate external users of a SharePoint application that is located in a perimeter network. The SharePoint application uses a credential store that is also in the perimeter network.
- Authenticate the SharePoint application in a perimeter network to a Web service in the corporate network with credentials that are recognized in the corporate security domain.
- Ensure the confidentiality and integrity of messages that are exchanged between the SharePoint application and the Web service.
- Have a Web service in a corporate domain apply authorization rules that are based on the identity of the external user.
- Avoid creating additional credential stores to store passwords in the SharePoint application.

Note:

This pattern is implemented in the Partner Portal reference implementation to secure access to the pricing service and the incident management service.

Solution

The corporate Web service authenticates the SharePoint application. SharePoint supplies the identities of the authenticated external users to the Web service. The Web service trusts SharePoint to authenticate the external users and to provide the correct identity information. The Web service applies authorization rules based on the identities. In effect, SharePoint provides a security façade that is trusted by the Web service for external users in the perimeter network. The following example demonstrates how the trusted façade is used in the Partner Portal application.

Contoso uses a SharePoint application named the Partner Portal that allows Contoso partners to access information from an incident management business system. One of the partners is a company that is named Fabrikam. The Partner Portal application relies on SharePoint forms-based authentication to authenticate incoming users from Fabrikam. For example, there is a Fabrikam employee who is named Joe. When Joe accesses a Web page in the Partner Portal application to view incident information, SharePoint calls a Web service that is exposed by the incident management business system.

The Web service authenticates the Partner Portal application identity. Typically, this means that the Web service authenticates the service account under which the SharePoint instance runs. An example of this identity might be the **PartnerPortalServiceAccount**. After the Partner Portal application is authenticated, it provides the Web service with an identity that is based on the external identity, "Joe from Fabrikam." In this example, it is only important to the Web service to know which partner employs Joe, so the identity SharePoint provides to the service is "Fabrikam." Based on this identity, the Web service returns incident information that is only applicable to Fabrikam.

Implementation Details

The Partner Portal implementation of the Trusted Façade pattern has two layers of Web service security that use the Windows Communication Foundation (WCF). The outer layer authenticates the SharePoint application that accesses the WCF service. It also guarantees message integrity and confidentiality. Authentication can occur over HTTP, HTTPS, or TCP and typically uses Windows domain credentials or certificate credentials. HTTPS or TCP are preferable to HTTP because they are better suited to ensure the confidentiality and integrity of the messages. The

authentication of the SharePoint application to the Web service typically uses one of the following security mechanisms:

- Active Directory credentials. The credentials for the application pool that is used by the SharePoint application use an Active Directory identity that is known to the corporate network.
- Certificate credentials. The SharePoint application uses a client certificate that is trusted by the Web service, and the Web service uses a certificate trusted by the SharePoint application.

Although other Web service security mechanisms can be used for the outer layer, you should make sure that you maintain message confidentiality and integrity, and that the external user identity can be passed to the Web service.

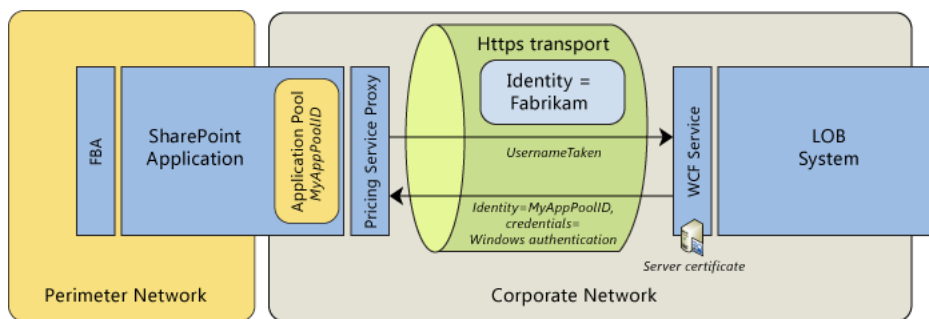
After the WCF service authenticates the calling SharePoint application as a trusted party, the inner security layer gives the Web service an identity that is related to the external user. The Web service trusts SharePoint to act as a façade that authenticates external users, and it allows SharePoint to vouch for the user's identity.

Note:

The trusted façade is conceptually similar to the Kerberos protocol transition with constrained delegation that implemented by the Windows Server 2003 and Windows Server 2008. For more information, see [Protocol Transition with Constrained Delegation Technical Supplement](#) on MSDN.

The following diagram represents the outer and inner layers of security that implement the trusted façade. This example uses Windows authentication and the HTTPS transport.

Trusted façade security layers



The next section describes the Partner Portal implementation of the trusted façade.

Partner Portal Implementation

The following section demonstrates the general approach to do the following:

- Secure the communication channel between the SharePoint application and the WCF service.
- Convey end-user identity information from the SharePoint application to the back-end service, over the secure channel.

Communication with the back-end service uses the HTTPS protocol. The protocol uses the Secure Sockets Layer (SSL) to encrypt messages at the transport level. A certificate is installed on the WCF service server to authenticate the SSL messages from the WCF service client. SharePoint uses Windows authentication to authenticate to the server. This establishes a trust relationship between the Web service and the SharePoint application. The SharePoint application authenticates the external user identity. It then transmits it in a **UsernameToken** token. It uses the secured outer layer to do this. The **UsernameToken** does not contain a password because SharePoint has already authenticated the external user.

Participants

The following are the participants in the authentication and authorization process.

- **External user.** The external user uses the SharePoint application that provides the credentials for authentication. SharePoint maintains a security context for the external user during the user session.
- **SharePoint application.** This authenticates the external user based on the credentials the user provides. The SharePoint application authenticates to the Web service and gives it the external user identity. The SharePoint application then exposes the functionality and data that it receives from the Web service to the external user.
- **External user identity store.** This stores the external user credentials for a particular identity in the perimeter network. Common examples are a SQL Server database, Active Directory Application Mode (ADAM), or Lightweight Directory Services (LDS).
- **Service.** This is the Web service that first authenticates SharePoint before granting it access. It makes authorization decisions that are based on the external user identity that SharePoint provides.
- **Internal user identity store.** This stores the user credentials for a particular identity in the corporate security domain. A common example is Active Directory.

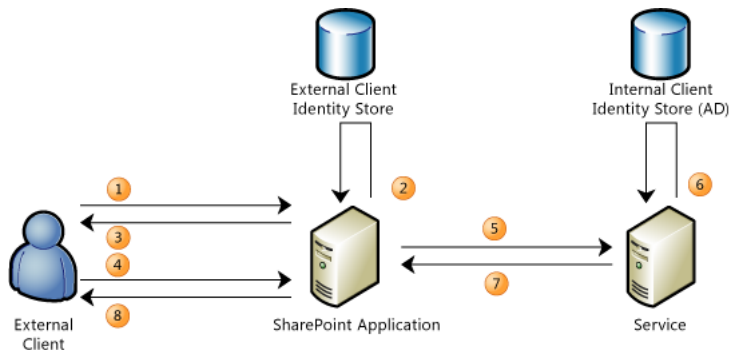
Note:

The internal client identity store can also be based on a certificate authority that establishes a cryptographically strong identity with a digital certificate. In this case, the identity does not need to be validated with a central identity store at run time, but the certificate authority that issued the certificate must be trusted. The certificate authority is conceptually equivalent to the internal client identity store.

Interaction Sequence

This section describes the authentication process. The following diagram outlines the sequence of steps.

Steps for authenticating a user and accessing business information



Step1: The User Authenticates to SharePoint

In the first step, the user authenticates to the SharePoint application, typically by providing a user name and password. SharePoint validates the information against credentials that are in the external user identity store, which is located in the perimeter network. SharePoint returns a security context to the client. This is an encrypted cookie. The result of this step is that SharePoint has established the identity of the external user and established a security context that is used in subsequent requests.

Step 2: The User Browses to a Page that Contains Business Data

The user submits a request to SharePoint for a page that includes data from the line-of-business (LOB) system. SharePoint derives the client's identity from the security context that was established in step 1. SharePoint retrieves the data from the LOB system (see step 3) and creates a page that contains the data that the external user is authorized to view. SharePoint provides the content in response to the browser request.

Step 3: SharePoint Retrieves Data from the Service

In this step, SharePoint uses a Web service to retrieve the business information from the LOB system. The following three events occur during this step:

1. The Web service authenticates the identity of the SharePoint application.
2. The Web service accepts the identity of the external user that it receives from SharePoint.
3. The Web service processes the operation that is contained in the message.

There are several ways to implement this sequence. The Partner Portal application has the following characteristics:

- SSL ensures message confidentiality and integrity.
- The Web service authenticates the SharePoint identity with NTLM.
- The Web service receives the external user identity as a **UserName** token.
- The implementation uses the WCF infrastructure.

The following explanation of the code shows how the steps are implemented.

Assume that the service has an SSL certificate and therefore accepts requests over HTTPS. The following XML is from the SharePoint application's Web.config file. It is mirrored on the Web service.

XML

```
<customBinding>
  <binding name="TrustedFacadeBinding">
    <security authenticationMode="UserNameOverTransport"/>
    <httpsTransport authenticationScheme="Ntlm" />
  </binding>
</customBinding>
```

The XML creates a custom binding. A custom binding is necessary because there are two types of authentication information. One type uses the outer security layer, and the other type uses the inner security layer. The **httpsTransport** setting configures the outer security layer. It instructs WCF to use the HTTPS protocol and to use NTLM to authenticate to the Web service. HTTPS secures the transport layer, and NTLM conveys the Windows credentials for the SharePoint application pool to the Web service.

The following code, which is in the SharePoint application, shows how to set a partner identity that is derived from the user identity on the client proxy.

C#

```
PricingClient client = new Contoso.LOB.Services.Client.PricingProxy.PricingClient();
client.ClientCredentials.UserName.UserName = this.PartnerId;
return new DisposableProxy<IPricing>(client);
```

WCF sends a **UserName** token to the Web service to establish the inner security layer. The identity used in this token is established by setting the **UserName** property on the WCF proxy.

After the security context is established, the Web service validates the identity, based on the context. The Partner Portal application uses a stub service to simulate a pricing engine. Validation occurs in two steps. The first step adds a **PrincipalPermission** attribute to the pricing operation to ensure that the calling service is authorized to access the Web service. This is shown in the following code.

C#

```
[PrincipalPermission(SecurityAction.Demand, Role=SecurityHelper.AdministratorsRole)]
[PrincipalPermission(SecurityAction.Demand, Role=SecurityHelper.AuthorizedRole)]
public Price GetPriceBySku(string sku)
{
    ...
}
```

The code checks that the accessing identity belongs to a permitted role; in this case, the code checks that the identity is one of two defined roles, AdministratorRole or AuthorizedRole. The SharePoint application pool belongs to the AuthorizedRole. After NTLM establishes the identity, SharePoint can access the Web service. At this point, the outer security layer is established and the Web service trusts the end-user identity that SharePoint provides. The following code shows how to retrieve the partner ID from the claim set that the WCF security context provides.

C#

```
public class SecurityHelper : ISecurityHelper
{
    ...
    public string GetPartnerId()
    {
        string clientUserName = string.Empty;

        foreach (ClaimSet claimSet in
            ServiceSecurityContext.Current.AuthorizationContext.ClaimSets)
        {
            foreach (Claim claim in claimSet)
            {
                if (claim.ClaimType == ClaimTypes.Name && claim.Right == Rights.Identity)
                {
                    clientUserName = (string)claim.Resource;
                    break;
                }
            }
        }
        return clientUserName;
    }
}
```

After it retrieves the security context, the business logic applies authorization rules that are based on the identity. In the following code, the partner identity determines the pricing discounts.

C#

```
[PrincipalPermission(SecurityAction.Demand, Role = SecurityHelper.AdministratorsRole)]
[PrincipalPermission(SecurityAction.Demand, Role = SecurityHelper.AuthorizedRole)]
public IList<Discount> GetDiscountsBySku(string sku)
{
    List<Discount> discounts = new List<Discount>();
    string query = string.Format(CultureInfo.CurrentCulture, "ProductSku = '{0}' AND PartnerId = '{1}'", sku, this.securityHelper.GetPartnerId());
    PricingDataSet.DiscountRow[] match = this.pricingDataSet.Discount.Select(query) as PricingDataSet.DiscountRow[];
    ...
}
```

Implementation Variations

The implementation variations of the Trusted Façade pattern primarily focus on how the security context is established between the SharePoint application and the Web service. There are many ways to do this, but the following are two recommended alternatives:

- **Use TCP transport and Windows stream security.** The WCF documentation provides an example of how to use the TCP transport in conjunction with the **windowsStreamSecurity** element to implement the Trusted Façade pattern. This approach requires a stream-oriented protocol such as TCP or named pipes. Typically, you must host the service as a Windows service. However, if you are using Windows Server 2008, you can take advantage of its process activation features. For more information, see [Windows Process Activation Service Support for Application Server](#) and [How to: Host a WCF Service in WAS](#) on MSDN. For the WCF example, see [Trusted Façade Service](#) on MSDN.
- **Use certificate-based authentication.** You can use certificate-based authentication if you configure a client certificate to authenticate the SharePoint application to the Web service. Use certificate-based authentication instead of Windows Challenge/Response (NTLM).

Considerations

The benefits of using the Trusted Façade pattern with WCF include the following:

- The pattern shows a way to communicate between two different authentication domains. It allows you to apply fine-grained authorization rules that are based on the identity of the external user. This is in contrast to a trusted subsystem, where the Web service has no information about the external user. Therefore, it applies the same coarse-grained authorization rules to all users.
- The pattern eliminates the need to store user passwords in SharePoint in order to authenticate with the LOB Web service. This is preferable to a custom single sign-on (SSO) implementation, such as SharePoint SSO.
- The pattern can be implemented with several transport security options, including TCP and HTTPS.

The following are some issues to consider if you use the Trusted Façade pattern:

- If the security of the SharePoint application is compromised and a client identity is impersonated, the Web service may return information to an unauthorized user. Web service operations are still limited to the operations that are permitted for the SharePoint application. This issue is the same as for the Trusted Subsystem pattern.
- You must configure identities and authorization rights based on identities that are derived from external users. In some cases, the system might not have this flexibility and additional logic that is required in the Web service.

Related Patterns

The Partner Portal implementation of the Trusted Façade pattern is a variation of the [Trusted Façade Service](#) that is discussed on MSDN. It also builds on the [Trusted Subsystem](#), which is also discussed on MSDN.

The SharePoint Guidance Library

The SharePoint Guidance Library contains reusable components that help you build robust SharePoint applications.

Note:

The SharePoint Guidance Library is distributed as source code. To build and install the library, you should either run the automatic configuration script for the Partner Portal reference implementation or perform the following steps to install the library without installing the reference implementation.

Use `gacutil.exe` to place the **Microsoft.Practices.ServiceLocation.dll** assembly into your system's global assembly cache. The assembly is included as a binary with this guidance.

In Visual Studio, open the `Microsoft.Practices.SPG2.sln` solution file, and then build the solution. This creates several assemblies and a SharePoint solution package. It creates the following three assemblies:

Microsoft.Practices.SPG.Common, **Microsoft.Practices.AJAXSupport**, and **Microsoft.Practices.SubSiteCreation**.

The following sections describe the components that are provided by the SharePoint Guidance Library:

- [SharePoint Service Locator](#). The SharePoint service locator decouples the consumers of an interface from the implementations of that interface. Instead of creating an object by invoking the constructor of a class, you request an object with a specified interface from the service locator. The SharePoint service locator uses SharePoint property bags to store the mappings of interfaces to corresponding implementation classes.
- [Configuration Manager](#). This is a hierarchical configuration manager that can safely store and retrieve configuration settings for a SharePoint farm, Web application, site collection, or site. The configuration manager uses SharePoint property bags as its backing store.
- [List-Based Repositories](#). A repository is a design pattern that separates a data source from its associated business logic by encapsulating the data access details. Repositories translate the underlying data representation into an entity model that fits the problem domain. The SharePoint Guidance Library includes classes that help you to implement repositories for SharePoint lists.
- [Workflow-Driven Site Creation](#). This component programmatically creates a subsite when a business event is added as a list item to a special SharePoint list. Adding an item to the list triggers a SharePoint sequential workflow. The workflow creates a collaboration Web site from a site template or site definition that is configured for that type of event.
- [Safe Script Manager](#). The **SafeScriptManager** class is an ASP.NET custom control that loads the .NET Framework's **ScriptManager** control if it is not already present on the current Web page. This control is useful for Web parts with child controls that require the script manager control because only one instance of the **ScriptManager** control can be added to a Web page. The **SafeScriptManager** control also helps you properly configure the AJAX **UpdatePanel** control for your SharePoint application.
- [SharePoint Logger](#). The SharePoint logger is a basic logging and tracing component that is tailored to the SharePoint environment. Logging is directed toward system administrators who typically rely on the Windows event logs to monitor deployed applications. Tracing is intended for developers.

The SharePoint Service Locator

The SharePoint Guidance Library includes an implementation of the service locator pattern named the SharePoint service locator. The SharePoint service locator is a reusable component that you can include in your own SharePoint applications. It allows you to decouple consumers of an interface from the implementations of that interface. Instead of creating an object by invoking the constructor of a class, you request an object with a specified interface from the service locator. Decoupling your code from a concrete implementation makes your code easier to test and more modular. For more information about the Service Locator pattern, see [Service Locator Pattern](#).

Components that provide implementations of interfaces can register those implementations with the SharePoint service locator. A SharePoint feature that provides a specific implementation typically registers the implementation with the service locator in its feature receiver.

Internally, the SharePoint service locator has a dictionary that maps an interface and an optional key string to the name of a class that implements the specified interface. The following table illustrates this. Each row in the table is a type mapping.

Interface	Registered implementation class
ILogger	SharePointLogger
IProductCatalogRepository	ProductCatalogRepository

For example, suppose that at run time you need to find a service that supports the **ILogger** interface. You invoke the service locator and the service locator responds by looking up a matching implementation class in its table of type mappings. By default, the corresponding implementation class is **SharePointLogger**.

The **SharePointServiceLocator** class provides access to a single service locator instance that is used by all application components. This instance implements the **IServiceLocator** interface from the Common Service Locator project. By default, the **SharePointServiceLocator.Current** property returns an implementation of **IServiceLocator** that is based on the **ActivatingServiceLocator** class. It is also possible to substitute other service locator implementations, such as the [Microsoft patterns & practices Unity Dependency Injection](#) container.

This section includes the following topics:

- [Key Scenarios](#)
- [Design of the SharePoint Service Locator](#)
- [Development How-to Topics](#)

Key Scenarios

The following sections discuss the key scenarios for the SharePoint service locator:

- [Getting Services from the SharePoint Service Locator](#). The most common scenario occurs when a client wants to use a service.
- [Registering a Service with the SharePoint Service Locator](#). You have created a feature that provides the implementation of an interface that you want to register for the service locator.
- [Setting Up the SharePoint Service Locator for Use in Unit Tests](#). You want to run a unit test in isolation. You must set up the service locator so that it returns the appropriate mock objects.
- [Creating a Custom Service Locator](#). This scenario occurs when you want to configure your application to use a service locator implementation other than the one provided by the SharePoint Guidance Library.

Getting Services from the SharePoint Service Locator

Typical Goals

In this scenario, you want to use a service that is managed by the SharePoint service locator. You need to retrieve an instance of the implementation class that is registered for an interface.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**, and to **Microsoft.Practices.ServiceLocation.dll**. Retrieve the current SharePoint service locator using the **Microsoft.Practices.SPG.Common.ServiceLocation.SharePointServiceLocator** static class's **Current** property. Invoke the current service locator's **GetInstance** method with the interface as the type parameter.

Getting Services from the SharePoint Service Locator

The following code shows several ways to get a service from the SharePoint service locator.

C#

```
using Microsoft.Practices.ServiceLocation;
using Microsoft.Practices.SPG.Common.ServiceLocation;

IServiceLocator serviceLocator = SharePointServiceLocator.Current;

// Get the default service.
IService1 service1 = serviceLocator.GetInstance<IService1>();

// Get a named service.
IService2 service2 = serviceLocator.GetInstance<IService2>("alternate");

// Get the default service, with types specified at run time.
IService3 service3 = (IService3)serviceLocator.GetInstance(typeof(IService3));

// Get a named service, with types specified at run time.
IService4 service4 =
    (IService4)serviceLocator.GetInstance(typeof(IService4), "alternate");

// Get a collection of all services registered for an interface type.
IEnumerable<IService5> services5 = serviceLocator.GetAllInstances<IService5>();

// Get a collection of all services registered for specified interface types
// with types specified at run time.
Type type1 = typeof(IService6);
IEnumerable<object> services6 = serviceLocator.GetAllInstances(type1);
```

Usage Notes

The **GetInstance** method of the SharePoint service locator either creates a new instance with every request or it uses a single shared instance for all requests. This behavior is set by an argument to the **RegisterTypeMapping** method of the **ServiceLocatorConfig** class.

Registering a Service with the SharePoint Service Locator

Typical Goals

In this scenario, you want to register the implementation for a specific interface with the SharePoint service locator. For example, you want to register a mapping of the **IPricingRepository** interface to the **PricingRepository** class that provides an implementation of **IPricingRepository**. This scenario occurs when a SharePoint feature is deployed that provides a new implementation of a class.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**, and to **Microsoft.Practices.ServiceLocation.dll**. Use the SharePoint service locator to get a reference to a **Microsoft.Practices.SPG.Common.ServiceLocation.IServiceLocatorConfig** interface and invoke the **RegisterTypeMapping** method. Use the interface and the implementation class you want your application class to use at run time as type parameters to the **RegisterTypeMapping** method.

Registering a Service with the SharePoint Service Locator

The following code shows how to add a service to an application's configuration from within a feature receiver class.

```
C#
using Microsoft.SharePoint;
using Microsoft.Practices.ServiceLocation;
using Microsoft.Practices.SPG.Common.ServiceLocation;

[CLSCompliant(false)]
[Guid("8b0f085e-72a0-4d9f-ac74-0038dc0f6dd5")]
public class MyFeatureReceiver : SPFeatureReceiver
{
    /// ...

    [SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
    public override void FeatureActivated(SPFeatureReceiverProperties properties)
    {
        // Get the ServiceLocatorConfig service from the service locator.
        IServiceLocator serviceLocator = SharePointServiceLocator.Current;
        IServiceLocatorConfig typeMappings =
            serviceLocator.GetInstance<IServiceLocatorConfig>();

        typeMappings.RegisterTypeMapping<IPricingRepository, PricingRepository>();
    }
}
```

The **FeatureActivated** method registers the **PricingRepository** class as the configured implementation of the **IPricingRepository** interface.

For more information about using feature receivers, see [Working with Features](#) on MSDN.

Usage Notes

The **RegisterTypeMapping** method has three overloaded versions. You can use these to provide a name for the type mapping and to indicate whether a new instance of the implementation class is provided with every service location request or whether a single shared instance is used for all requests. By default, a new instance is created with every service location request.

Note:

It is generally recommended that you do not use the service locator to register types as singletons. Registering a service as a singleton requires the service to be thread safe. Unless you have verified that your service is thread safe, you should not share a single instance of your object across all application threads.

If you want to remove a type mapping, you must either explicitly remove it using code or register a new type mapping to replace it. Type mappings are not automatically removed when a feature is deactivated or uninstalled.

If an interface appears in more than one type mapping, the most recently registered type mapping will be used to return implementation instances.

For an example of how to get a service instance based on a registered type mapping, see [Getting Services from the SharePoint Service Locator](#).

Setting Up the SharePoint Service Locator for Use in Unit Tests

Typical Goals

In this scenario, you want to run unit tests on isolated sections of code that use services from the service locator. To isolate the code, the service locator must return mock objects that you specify instead of the real implementation.

You can do this by replacing the current service locator instance with a service locator that you have created. It is also possible to create your own mock service locator that implements the **IServiceLocator** interface, but it is often easier to use the existing service locator such as the **ActivatingServiceLocator** class.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**, and to **Microsoft.Practices.ServiceLocation.dll**. Create an instance of the **Microsoft.Practices.SPG.Common.ServiceLocation.ActivatingServiceLocator** class and invoke its **RegisterTypeMapping** method. Use the interface and the implementation class you want your application class to use at run time as type parameters to the **RegisterTypeMapping** method. Call the **ReplaceCurrentServiceLocator** static method of the **Microsoft.Practices.SPG.Common.ServiceLocation.SharePointServiceLocator** class to set the current service locator.

Setting Up the SharePoint Service Locator for Use in Unit Tests

The following code shows a unit test's initialization method that replaces a configured service with a test-specific class. This example comes from the PricingFixture.cs class in the Contoso.LOB.Services.Tests project.

C#

```
[TestClass()]
public class PricingFixture
{
    private MockSecurityHelper mockSecurityHelper;

    [TestInitialize]
    public void Init()
    {
        // Create the service locator that will be used by the tests.
        ActivatingServiceLocator serviceLocator = new ActivatingServiceLocator();

        // Populate it with a mock object for the ISecurityHelper. Instantiate it
        // as a singleton, so each request to the service locator will return the
        // same object.
        serviceLocator.RegisterTypeMapping<ISecurityHelper,
            MockSecurityHelper>(InstantiationType.AsSingleton);

        // Replace the SharePointServiceLocator's Current property with the one
        // that was just created.
        SharePointServiceLocator.ReplaceCurrentServiceLocator(serviceLocator);

        // Get the mock object, so you can set some values on it in each unit test.
        this.mockSecurityHelper =
            SharePointServiceLocator.Current.GetInstance<ISecurityHelper>()
                as MockSecurityHelper;
    }

    [TestMethod]
    public void GetPriceForUnknownPartnerReturnsBasePrice()
    {
        // In the actual test, you can either set up your mocks to do what you want
        // or at the end of your tests, make sure they have been called.
        mockSecurityHelper.UserToReturn = "999";

        Pricing target = new Pricing();
        // ... Call some method on the Pricing object that needs
        // the mockSecurityHelper.
    }

    [TestCleanup]
    public void Cleanup()
    {
        // After each test, make sure to reset the ServiceLocator, to ensure you
        // do not interfere with other unit tests.
        SharePointServiceLocator.Reset();
    }
}
```

Usage Notes

It is recommended that you use the **Reset** method to return the service locator to its original state as part of the unit test's cleanup step. This prevents tests from interfering with each other.

Passing the **InstantiationType.AsSingleton** enumerated value as an argument to the **RegisterTypeMapping** method allows you to specify that a single instance of the implementation class is used for all service location requests. This is important in the unit-testing scenario because you typically want to set test-specific properties of the mock object that your test environment provides. This technique allows the test environment to isolate the implementation under test. By default, the **ActivatingServiceLocator** class creates a new instance with each service location request.

Note:

In the unit testing scenario, it is important to replace the current SharePoint service locator before you attempt to get the value of the **SharePointServiceLocator.Current** static property. The reason for this is that instantiating the default SharePoint service locator requires you to run in a SharePoint context, which may not be the case when performing a unit test. By replacing the current service locator before evaluating the **SharePointServiceLocator.Current** property, you can avoid a run-time dependency on the SharePoint environment.

Creating a Custom Service Locator

Typical Goals

In this scenario, you want to configure your application to use a service locator implementation other than the one provided by the SharePoint Guidance Library.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**, and to **Microsoft.Practices.ServiceLocation.dll**. Create a class that implements the **IServiceLocatorFactory** interface and creates the custom service locator instance. Register your new class factory using the **IServiceLocatorConfig** interface.

Using a Custom Service Locator

The following procedure describes how to use a custom service locator.

To use a custom service locator

1. Create a **ServiceLocatorFactory** class that will instantiate the custom service locator and populate it with type mappings.
2. Register the **ServiceLocatorFactory** implementation in the **ServiceLocatorConfig** class.

The following code example declares a new implementation of the **IServiceLocatorFactory** interface.

```
C#
class MyServiceLocatorFactory : IServiceLocatorFactory
{
    public IServiceLocator Create()
    {
        return new MyServiceLocator();
    }

    public void LoadTypeMappings(IServiceLocator serviceLocator,
                                IEnumerable<TypeMapping> typeMappings)
    {
        if (serviceLocator == null)
        {
            throw new ArgumentNullException("serviceLocator");
        }

        MyServiceLocator myServiceLocator = serviceLocator as MyServiceLocator;

        if (typeMappings == null)
        {
            return;
        }

        foreach (TypeMapping typeMapping in typeMappings)
        {
            // ... invoke custom methods of MyServiceLocator to establish type mappings
        }
    }
}
```

In this example, the **MyServiceLocator** class represents the custom service locator implementation you want to instantiate.

The following code example shows how to configure your application to use your custom service locator factory.

```
C#
IServiceLocator serviceLocator = SharePointServiceLocator.Current;
IServiceLocatorConfig typeMappings =
    serviceLocator.GetInstance<IServiceLocatorConfig>();
typeMappings.RegisterTypeMapping<IServiceLocatorFactory, MyServiceLocatorFactory>();
```

Usually, the preceding code runs inside a feature receiver.

Usage Notes

The **Create** method must return an instance of a class that adheres to the contract that is specified by the Common Service Locator's **IServiceLocator** interface.

The **LoadTypeMappings** method initializes the object that is returned by the **Create** method with application-specific type mappings that are provided as arguments. The **LoadTypeMapping** method must add each entry of the type-mapping enumeration to your service locator's table of type mappings.

It is possible that the **LoadTypeMappings** method can be invoked more than once. In the current version of the SharePoint Guidance Library, this method is invoked once for the default type mappings for the SharePoint Guidance Library and once for type mappings that are stored in configuration as SharePoint properties. Your implementation of the **LoadTypeMappings** method must be able to overwrite the previous type mappings and ensure that the most recent type mapping takes precedence.

Design of the SharePoint Service Locator

The SharePoint Guidance Library contains the **SharePointServiceLocator** class. With this class, you can do the following:

- Request instances of services by providing the interface for that particular service.
- Register a mapping between an interface and a class that implements that interface.
- Read and write the type mappings from the **SPFarm** property bag.
- Substitute a different service locator implementation, such as the Microsoft patterns & practices Unity dependency injection container.
- Use the service locator to inject mock objects into your unit tests.

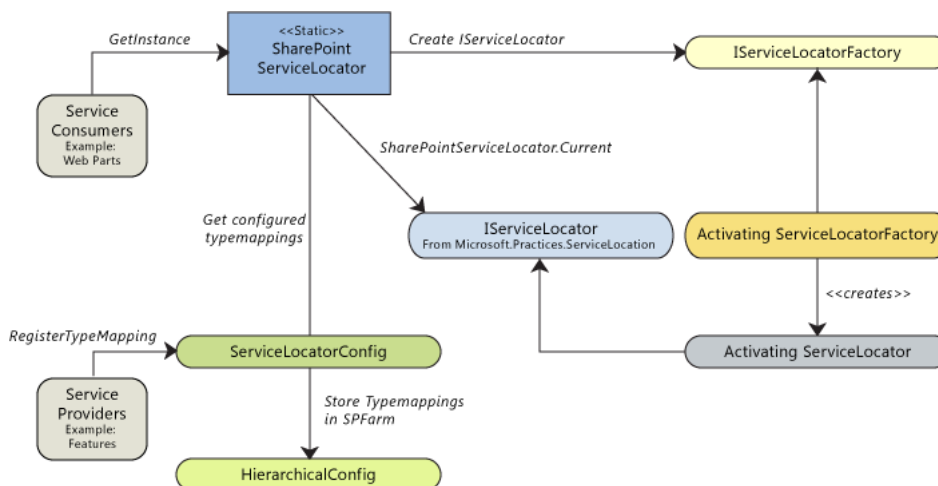
Implementation Overview

The SharePoint service locator defines the following classes:

- **SharePointServiceLocator**. This static class is the main entry point for the service locator and is responsible for managing the singleton **IServiceLocator** instance.
- **ActivatingServiceLocator**. This class provides an implementation of the **IServiceLocator** interface. This class will create instances of the requested services if required.
- **ServiceLocatorConfig**. This class allows you to save type mappings as configuration properties. The service locator creates instances at run time based on the type mappings that are saved using this class.
- **ActivatingServiceLocatorFactory**. This class implements the **IServiceLocatorFactory** interface. This class is responsible, by default, for instantiating the current service locator.

The following diagram shows the design of the SharePoint service locator.

Design of the SharePoint service locator



The main entry point to the service locator is the **SharePointServiceLocator** static class. This class creates, initializes, and exposes the singleton instance of the service locator. It uses the **Current** property to expose the instance. The singleton instance implements the **IServiceLocator** interface and has both generic and non-generic methods named **GetInstance** for retrieving services from a service locator.

Relationship to the Common Service Locator

This **IServiceLocator** interface is defined by the [Common Service Locator library](#), which is available on [CodePlex](#). The goal of the Common Service Locator is to provide a shared interface for inversion of control (IOC) containers and service locators. Because the SharePoint service locator relies on the Common Service Locator interface, it is possible to use service location without being dependent on a specific implementation. The SharePoint Guidance Library provides an **IServiceLocator** implementation with the **ActivatingServiceLocator** class, but the Common Service Locator Library provides several other **IServiceLocator** adapters, such as one for the [Microsoft patterns & practices Unity Application Block](#) and one for the [Spring.NET Framework](#).

Note:

The Common Service Locator project's **ServiceLocator** class cannot be used inside of SharePoint because it requires bootstrapping that is not possible in the SharePoint environment. The SharePoint Guidance Library ensures that calling this class throws a **NullReferenceException** or a **NotSupportedException**.

Creation of the Service Locator Instance

The **IServiceLocator** instance is created the first time that the **Current** property of the **SharePointServiceLocator** class is retrieved. The SharePoint service locator calls into an **IServiceLocatorFactory** instance that creates and configures an **IServiceLocator** instance. This process works as follows:

1. The **SharePointServiceLocator** calls the **ServiceLocatorConfig** class to retrieve the type mappings that are

stored in configuration using SharePoint property bags.

2. To create an **IServiceLocator** instance, the **SharePointServiceLocator** class looks to see if a custom **IServiceLocatorFactory** is configured.
3. If no custom **IServiceLocatorFactory** is configured, the **ActivatingServiceLocatorFactory** class is used as the default class factory responsible for producing the **IServiceLocator** instance.
4. If a custom **IServiceLocatorFactory** is configured, the **SharePointServiceLocator** will use the **IServiceLocatorFactory** to create a new **IServiceLocator** instance and populate it with the type mappings that it has read using the **ServiceLocatorConfig** class.

By default, the **Current** property returns an instance of the **ActivatingServiceLocator** class. This is a service locator that can create instances for services that have a constructor with no parameters.

The **ActivatingServiceLocator** class allows you to specify that a single instance of the implementation class is used for all service location requests. However, it is recommended that you not use a singleton instance unless you are comfortable with writing thread-safe services and if you have a real need for a single shared instance of your service.

Note:

The **ActivatingServiceLocator** class does not perform dependency injection. If you need this functionality, consider plugging in a dependency injection container such as the Microsoft patterns & practices Unity Application Block.

Configuring Type Mappings

You can add and remove custom type mappings by using the **RegisterTypeMapping** and **RemoveTypeMapping** methods of the **ServiceLocatorConfig** class. This class uses the **HierarchicalConfig** class to store the type mappings as properties of the current SharePoint farm's property bag.

Note:

The configuration information is stored as a farm-level property because the **SPFarm.Local** property is always present when code is executed on a SharePoint server. This means that the service locator can also be used from command line utilities and feature receivers. However, the SharePoint service locator needs to have read access to the **SPFarm.Local**. Typically, adding type mappings is done by a feature receiver in the **FeatureActivated** method. By default, type mappings are not removed when a feature is deactivated.

Passing the **InstantiationType.AsSingleton** enumerated value as an argument to the **RegisterTypeMapping** method allows you to specify that a single instance of the implementation class is used for all service location requests. The default value, **InstantiationType.NewInstanceForEachRequest**, creates new instances with each request.

The **SharePointServiceLocator** includes some default type mappings, so that it can work "out of the box." The following services are available without further configuration:

- **Logging.** By default, the **ILogger** interface is mapped to the **SharePointLogger** class. By default, this logging implementation logs to the event log and to the Unified Logging Service (ULS), but this behavior can also be overwritten by registering custom type mappings for the **IEventLogLogger** or **ITraceLogger** interfaces.
- **Configuration management.** By default, the **IConfigManager** and **IHierarchicalConfig** interfaces are mapped to the **HierarchicalConfig** class.

These default services can be overwritten by registering custom type mappings using the **ServiceLocatorConfig** class.

Using Service Location in Unit Testing

You can use the SharePoint service locator in your unit tests. To do this, you have to replace the current service locator with a custom service locator that has the type mappings that you need to run your unit tests. You can do this by calling the **ReplaceCurrentServiceLocator** method of the **SharePointServiceLocator** class. Calling this method before using the **Current** property prevents the **SharePointServiceLocator** class from trying to read type mappings from configuration and creating the new service locator. You can replace the service locator with a mock object, or you can also create an instance of the **ActivatingServiceLocator** class and fill it with the type mappings that you need for your tests.

After your test completes, call the **Reset** method of the **SharePointServiceLocator** class. This ensures that the next call of the **SharePointServiceLocator.Current** property creates a new service locator instance.

The SharePoint service locator raises an exception of type **ActivationException** if an error occurs during the process of service location. It may also raise .NET Framework exceptions for assembly and class load errors and exceptions of type **NoSharePointContextException** when a SharePoint context is required but not present.

Note:

Each SharePoint Web application runs in its own application domain. Calling the **ReplaceCurrentServiceLocator** and **Reset** methods only affects the **SharePointServiceLocator.Current** instance in the current application domain. These methods do not have an effect in other application domains. Therefore, the **ReplaceCurrentServiceLocator** and **Reset** methods should only be used within unit tests. If you want to change the type of service locator for your application, you should register a custom **IServiceLocatorFactory** interface, as described earlier.

Versioning Type Registrations

When registering types with the ServiceLocator, the fully qualified assembly name is used. This name contains the

name of the class or interface, the name of the assembly that contains the class, the version, and a hash of the public key. You will not have versioning issues for the following reasons:

- You can register a new implementation of an interface without recompiling the clients.
- If you change the interface, you need to also recompile the consumers.

You are also able to install several versions of interfaces side by side. However, having side by side versions of assemblies is an advanced .NET Framework topic that goes beyond the scope of this guidance.

Please consider the following recommendations when designing your features:

- It is recommended that the feature that contains an implementation of an interface also registers the type mapping. When the feature is installed, you should register the type mapping.
- Consider unregistering the type mapping when the feature is uninstalled. Although this is not required, it does keep the registration database clean.
- It is not recommended to override type mappings that have been made by a different feature. If feature A installs a type mapping for interface 1 and feature B overrides that type mapping, there is no built-in way to get the original type mapping back if feature B is uninstalled.

Development How-to Topics

This topic describes how to use the SharePoint Guidance Library's SharePoint service locator. The SharePoint service locator decouples the consumers of an interface from the implementations of that interface. It includes the following procedures:

- [How to: Register Services with the SharePoint Service Locator](#). This topic describes how to register a service with the service locator so that a client can use it.
- [How to: Get Services from the SharePoint Service Locator](#). This topic describes how to use a service that is registered with the service locator.

How to: Register Services with the SharePoint Service Locator

The following procedure describes how to register a service with the SharePoint service locator.

To register a service

1. In Visual Studio, add a reference to the SharePoint Guidance Library **Microsoft.Practices.SPG.Common.dll** and **Microsoft.Practices.ServiceLocation.dll**. If you are writing a feature receiver, item event receiver, workflow or anything else that needs to be in the global assembly cache, the **Microsoft.Practices.SPG.Common** and **Microsoft.Practices.ServiceLocation** assemblies also need to be in the global assembly cache.
 2. Create a feature receiver class and add code to the **FeatureInstalled** method that does the following:
 - Uses the **Microsoft.Practices.SPG.Common.ServiceLocation.SharePointServiceLocator** static class's **Current** property to retrieve the current SharePoint service locator
 - Uses the current SharePoint service locator to get an instance of the **IServiceLocatorConfig** interface
 - Uses the instance of the **IServiceLocatorConfig** interface to register the service type and interface type
- The following code demonstrates how to perform this step.

```
C#
using Microsoft.SharePoint;
using Microsoft.Practices.ServiceLocation;
using Microsoft.Practices.SPG.Common.ServiceLocation;

[CLSCompliant(false)]
[Guid("<Your feature GUID>")]
public class MyFeatureReceiver : SPFeatureReceiver
{
    /// ...

    [SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
    public override void FeatureInstalled(SPFeatureReceiverProperties properties)
    {
        // Get the ServiceLocatorConfig service from the service locator.
        IServiceLocator serviceLocator = SharePointServiceLocator.Current;
        IServiceLocatorConfig serviceLocatorConfig =
            serviceLocator.GetInstance<IServiceLocatorConfig>();

        serviceLocatorConfig.RegisterTypeMapping<IMyService, MyService>();
    }
}
```

The **FeatureInstalled** method is called when the feature is installed on a Web front-end server. The service registration is stored in the **SPFarm** object's property bag and the application pool is recycled. This forces the **SharePointServiceLocator** instance to reload its type mappings from the **ServiceLocatorConfig** instance.

How to: Get Services from the SharePoint Service Locator

This topic describes how to use a service that is managed by the SharePoint service locator. You must retrieve an instance of the implementation class that is registered for the interface.

The following services that are provided in the **Microsoft.Practices.SPG.Common** assembly are already registered with the **SharePointServiceLocator** class:

- **SharePointLogger**. This is registered as an implementation of the **ILogger** interface.
- **TraceLogger**. This is registered as an implementation of the **ITraceLogger** interface.
- **EventLogLogger**. This is registered as an implementation of the **IEventLogLogger** interface.
- **HierarchicalConfig**. This is registered as an implementation of the **IHierarchicalConfig** and **IConfigManager** interfaces.
- **ServiceLocatorConfig**. This is registered as an implementation of the **IServiceLocatorConfig** interface.

To get an instance of a service

1. In Visual Studio, add a reference to the SharePoint Guidance Library **Microsoft.Practices.SPG.Common.dll** and **Microsoft.Practices.ServiceLocation.dll**. If you are writing a feature receiver, item event receiver, workflow or anything else that needs to be in the global assembly cache, the **Microsoft.Practices.SPG.Common** and **Microsoft.Practices.ServiceLocation** assemblies also need to be in the global assembly cache.
2. Retrieve the current SharePoint service locator using the **Microsoft.Practices.SPG.Common.ServiceLocation.SharePointServiceLocator** static class's **Current** property.
3. Invoke the current service locator's **GetInstance** method with the interface as the type parameter.

The following code demonstrates how to get an instance of the service.

```
C#
using Microsoft.Practices.ServiceLocation;
using Microsoft.Practices.SPG.Common.ServiceLocation;

IServiceLocator serviceLocator = SharePointServiceLocator.Current;

// Get an instance of ILogger.
ILogger logger = serviceLocator.GetInstance<ILogger>();

// Get an instance of a custom service registered to the
// SharePoint service locator.
ICustomService customService = serviceLocator.GetInstance<ICustomService>();
```

The Configuration Manager

Frequently, SharePoint applications retrieve configuration settings at run time that were set when the application was installed or configured. To do this, the applications must access the configuration information, which can be stored in several locations. For example, configuration information can be stored in the Web.config file, the SharePoint configuration database (for example, values in the **SPFarm** property bag are stored in this database), or in a content database (for example, in an **SPList** object are stored in the content database). There are many challenges associated with these techniques. For example, improper use of the **SPFarm** property bag can corrupt the SharePoint configuration database. Also, although you can easily change the Web.config file with a feature installer, it is much harder to make changes to the Web.config file after a feature is deployed.

The SharePoint Guidance Library includes a hierarchical configuration manager that can safely store and retrieve configuration settings at the following levels:

- Farm (**SPFarm** class)
- Web application (**SPWebApplication** class)
- Site collection (**SPSite** class)
- Site (**SPWeb** class)

The configuration manager is a reusable component that you can include in your own SharePoint applications. It can store simple types, such as integers or strings, as well as any type that can be serialized to XML.

The configuration manager is easier to use than the SharePoint API because the configuration manager offers a uniform and type-safe way to read and write configuration settings. Also, the configuration manager does not allow you to store non-serializable values. This prevents you from accidentally corrupting the content or configuration database.

The configuration manager exposes two interfaces that are named **IHierarchicalConfig** and **IConfigManager**. The **IHierarchicalConfig** interface is intended for *readers* of configuration information, such as Web Parts. The **IConfigManager** interface is intended for *writers* of configuration information, such as feature receivers.

The **IHierarchicalConfig** interface hierarchically reads the configuration settings in your current context, through the **SPContext.Current** property. If a setting is not in the property bag of the current **SPWeb** object, the configuration manager next looks in the current **SPSite** object, and then it looks in the current **SPWebApplication** object, and finally, it looks in the current **SPFarm** object. Calls to the **IHierarchicalConfig** interface throw an exception if the caller of the **IHierarchicalConfig** interface is not running in the context of a SharePoint Web application.

The **IConfigManager** interface reads and writes configuration settings at specific locations in the hierarchy. Unlike the **IHierarchicalConfig** interface, the methods in this interface do not traverse the hierarchy. In addition, they do not require a SharePoint execution environment. You can pass in an arbitrary **SPFarm**, **SPWebApplication**, **SPSite** or **SPWeb** object in which to read or write the configuration settings. This means that the **IConfigManager** interface can be consumed from code, such as command line applications and feature receivers that do not run in a SharePoint context and do not have access to the **SPContext.Current** property.

Note:

The SharePoint Guidance Library's Configuration Manager provides an API to read and write configuration settings. It does not provide a user interface (UI) to read and write these configuration settings at run time. To do this, you either can create a custom UI for the configuration manager or you can use a general purpose property bag editor. For example, on [CodePlex](#), there is a community-driven effort to create a property bag editor that allows you to change the raw property bag values.

This section includes the following topics to help you use and understand the configuration manager:

- [Key Scenarios](#)
- [Design of the Configuration Manager](#)
- [Development How-to Topics](#)

Key Scenarios

The following topics explain the key scenarios for the SharePoint Guidance Library's configuration manager:

- [Adding and Updating Configuration Settings](#)
- [Removing Configuration Settings](#)
- [Retrieving Configuration Settings](#)

Adding and Updating Configuration Settings

Typical Goals

In this scenario, you want to save a piece of configuration information at one of the following four levels of the SharePoint context:

- Site (**SPWeb** class)
- Site collection (**SPSite** class)
- Web application (**SPWebApplication** class)
- Farm (**SPFarm** class)

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**, and to **Microsoft.Practices.ServiceLocation.dll**. Use the SharePoint service locator to obtain an instance that provides the **Microsoft.Practices.SPG.Common.Configuration.IConfigManager** interface and invoke the **SetInPropertyBag** method. For more information about the SharePoint service locator, see [The SharePoint Service Locator](#).

Adding and Updating Configuration Settings

The following code shows how to use the **SetInPropertyBag** method to add or update a configuration setting. There are overloaded versions of this method with parameter types that are specialized for the SharePoint classes **SPWeb**, **SPSite**, **SPWebApplication**, and **SPFarm**.

```
C#
// For IServiceLocator.
using Microsoft.Practices.ServiceLocation;
using Microsoft.Practices.SPG.Common.Configuration;
// For SharePointServiceLocator.
using Microsoft.Practices.SPG.Common.ServiceLocation;
// For SPFarm.
using Microsoft.SharePoint.Administration;

IServiceLocator serviceLocator = SharePointServiceLocator.Current;

IConfigManager configManager = serviceLocator.GetInstance<IConfigManager>();

// Open a specific SPWeb and add a new property or update an existing property.
// Note that if you open a specific SPWeb/SPSite (and do not use SPContext) you MUST
// dispose of it after you are done.

using (SPSite site = new SPSite(siteUrl))
{
    using (SPWeb web = site.OpenWeb()){
        configManager.SetInPropertyBag("Contoso.Applications.Workgroupname",
                                      "Customer Service",
                                      web);
    }
}

// Add a new site collection property or update an existing property.
// Note: If you use an SPWeb/SPSite from SPContext, you must NOT dispose of it!
configManager.SetInPropertyBag("Contoso.Applications.DivisionName",
                              "Pharmaceuticals",
                              SPContext.Current.Site);

// Add a new Web application property or update an existing property.
configManager.SetInPropertyBag("Contoso.Applications.CompanyName",
                              "Contoso",
                              SPContext.Current.Site.WebApplication);

// Add a new farm property or update an existing property.
configManager.SetInPropertyBag("Contoso.Applications.FarmLocation",
                              "Redmond",
                              SPFarm.Local);
```

Usage Notes

The first argument to the **SetInPropertyBag** method is the key that defines the configuration setting. If this is a null string, an exception is thrown. Because there might be name collisions with properties that were set by other applications or by SharePoint itself, it is recommended that you fully qualify the key for each configuration setting with the namespace of the code that defines the setting.

The second argument to the **SetInPropertyBag** method is the new value of the configuration setting. The value

must be an object that is serializable to XML.

Note:

In the **SPWeb** property bag, setting a property value to **null** also removes the key from the property bag.

There are several overloaded versions of the **SetInPropertyBag** method, depending on which level of the SharePoint hierarchy you want to update. For more information, see [The SharePoint Guidance Library](#).

A typical scenario is to configure a component that is being installed by invoking the **SetInPropertyBag** method in a feature receiver. The following code shows how to use the configuration manager's **SetInPropertyBag** method from within a feature receiver.

C#

```
public override void FeatureActivated(SPFeatureReceiverProperties properties)
{
    IServiceLocator serviceLocator = SharePointServiceLocator.Current;

    IConfigManager configManager = serviceLocator.GetInstance<IConfigManager>();
    SPWebApplication webApp = (SPWebApplication)properties.Feature.Parent;

    // ...

    configManager.SetInPropertyBag(
        "Contoso.Pharmaceuticals.Applications.DefaultLocation",
        "Redmond",
        webApp);
}
```

Note:

The **Parent** property of a feature depends on the scope of the feature that is being activated. In this example, the feature is scoped at the Web application level.

The following table shows SharePoint groups and the default permission levels that apply when adding or updating configuration settings.

Group (default permission level)	Can set site configuration	Can set site collection configuration	Can set Web application configuration	Can set farm configuration
<i>Site name</i> Visitors (Read)	No	No	No	No
<i>Site name</i> Members (Contribute)	No	No	No	No
<i>Site name</i> Owners (Full Control)	Yes	Yes	No	No
Farm Administrators (Full Control)	Yes	Yes	Yes	Yes

Note:

Site name Owners is a SharePoint group that is created by default with the site collection. The *Site name* Owners group has Full Control permissions. The site administrator is a member of this group. For more information about SharePoint permissions, see [Permission Levels and Permissions](#) on the Microsoft Office Online Web site.

Removing Configuration Settings

Typical Goals

In this scenario, you want to delete a property that you previously set using the **SetInPropertyBag** method of the **IConfigManager** interface.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**, and to **Microsoft.Practices.ServiceLocation.dll**. Use the SharePoint service locator to obtain an instance of a class that provides the **Microsoft.Practices.SPG.Common.Configuration.IConfigManager** interface and invoke the **RemoveKeyFromPropertyBag** method. For more information about the SharePoint service locator, see [The SharePoint Service Locator](#).

Removing Configuration Settings

The following code shows how to remove a configuration setting using the **RemoveKeyFromPropertyBag** method.

```
C#
// For IServiceLocator.
using Microsoft.Practices.ServiceLocation;
// For IConfigManager.
using Microsoft.Practices.SPG.Common.Configuration;
// For SharePointServiceLocator.
using Microsoft.Practices.SPG.Common.ServiceLocation;
// For SPFarm.
using Microsoft.SharePoint.Administration;

IServiceLocator serviceLocator = SharePointServiceLocator.Current;

IConfigManager configManager = serviceLocator.GetInstance<IConfigManager>();

// Remove a site property.
configManager.RemoveKeyFromPropertyBag("Contoso.Applications.Workgroupname",
    SPContext.Current.Web);

// Remove a site collection property.
configManager.RemoveKeyFromPropertyBag("Contoso.Applications.DivisionName",
    SPContext.Current.Site);

// Remove a Web application property.
configManager.RemoveKeyFromPropertyBag("Contoso.Applications.CompanyName",
    SPContext.Current.Site.WebApplication);

// Remove a farm property.
configManager.RemoveKeyFromPropertyBag("Contoso.Applications.FarmLocation",
    SPFarm.Local);
```

Usage Notes

The first argument to the **RemoveKeyFromPropertyBag** method is the key of the configuration setting to be deleted. This should be a string that was previously used in a corresponding invocation of the **SetInPropertyBag** method. The **RemoveKeyFromPropertyBag** method does not fail if the key cannot be found, so this method can be safely called to ensure that the key is removed.

Note:

In the **SPWeb** property bag, setting a property value to **null** has the same effect as removing the key from the property bag.

The second argument to the **RemoveKeyFromPropertyBag** method is an instance of one of the following classes that are provided by SharePoint: **SPWeb**, **SPSite**, **SPWebApplication**, and **SPFarm**. The property that is located in the property bag of the specified object will be removed.

The following table shows the group names and default permission levels that apply when removing configuration settings.

Group (default permission level)	Can remove site configuration	Can remove site collection configuration	Can remove Web application configuration	Can remove farm configuration
Site name Visitors (Read)	No	No	No	No
Site name Members (Contribute)	No	No	No	No

<i>Site name</i> Owners (Full Control)	Yes	Yes	No	No
Farm Administrators (Full Control)	Yes	Yes	Yes	Yes

Retrieving Configuration Settings

Typical Goals

In this scenario, you want to retrieve the value of a property that you previously set using the **SetInPropertyBag** method of the **IConfigManager** interface.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**, and to **Microsoft.Practices.ServiceLocation.dll**. Use the SharePoint service locator to obtain an instance of a class that provides the **Microsoft.Practices.SPG.Common.Configuration.IHierarchicalConfig** interface and invoke the **ContainsKey** method. If the **ContainsKey** method returns **true**, invoke the **GetByKey** method to retrieve the stored value. For more information about the SharePoint service locator, see [The SharePoint Service Locator](#).

Retrieving Configuration Settings

The following code shows how to retrieve a configuration setting using the **ContainsKey** and **GetByKey** methods.

```
C#
// For IServiceLocator
using Microsoft.Practices.ServiceLocation;
using Microsoft.Practices.SPG.Common.Configuration;
// For SharePointServiceLocator.
using Microsoft.Practices.SPG.Common.ServiceLocation;
// For SPFarm.
using Microsoft.SharePoint.Administration;

IServiceLocator serviceLocator = SharePointServiceLocator.Current;

IHierarchicalConfig config = serviceLocator.GetInstance<IHierarchicalConfig>();

string v1;

// Retrieve a string-valued config setting from the current site (SPWeb), or if not
// found from the current SPSite, SPWebApplication or SPFarm.
if (config.ContainsKey("Contoso.Applications.Workgroupname"))
    v1 = config.GetByKey<string>("Contoso.Applications.Workgroupname");

bool v2;

// Retrieve a Boolean-valued config setting from the current site collection
// (SPSite), or if not found, from the current SPWebApplication or SPFarm.
if (config.ContainsKey("Contoso.Applications.DivisionFlag",
                      ConfigLevel.CurrentSPSite))
    v2 = config.GetByKey<bool>("Contoso.Applications.DivisionFlag",
                             ConfigLevel.CurrentSPSite);

MyClass v3;

// Retrieve an instance of a serializable class from the current Web application
// (SPWebApplication) or if not found, from the current SPFarm.
if (config.ContainsKey("Contoso.Applications.CompanyProfile",
                      ConfigLevel.CurrentSPWebApplication))
    v3 = config.GetByKey<MyClass>("Contoso.Applications.CompanyProfile",
                                 ConfigLevel.CurrentSPWebApplication);

string v4;

// Retrieve a string-valued config setting from the current SPFarm.
if (config.ContainsKey("Contoso.Applications.FarmLocation",
                      ConfigLevel.CurrentSPFarm))
    v4 = config.GetByKey<string>("Contoso.Applications.FarmLocation",
                                 ConfigLevel.CurrentSPFarm);
```

Usage Notes

The first argument to the **ContainsKey** method and **GetByKey** method is the name of the property to be retrieved. This should be a string that was previously used in a corresponding invocation of the **SetInPropertyBag** method, as described in [Adding and Updating Configuration Settings](#).

An optional second argument to the **ContainsKey** method and **GetByKey** method is one of the following enumerated values:

- **ConfigLevel.CurrentSPWeb**. This value indicates that properties of the current site, site collection, Web application, and farm are searched.
- **ConfigLevel.CurrentSPSite**. This value indicates that properties of the current site collection, Web application, and farm are searched.
- **ConfigLevel.CurrentSPWebApplication**. This value indicates that properties of the current Web application

and farm are searched.

- **ConfigLevel.CurrentSPFarm.** This value indicates that properties of the current farm are searched.

If you do not provide the second argument to the **ContainsKey** method or the **GetByKey** method, the **ConfigLevel.CurrentSPWeb** value is used.

The **GetByKey** method takes a type parameter that specifies the type of the value that will be returned.

 Note:

If you do not want to search up the hierarchy, you can use the **ContainsKeyInPropertyBag** and **GetFromPropertyBag** methods that are provided by the configuration manager's **IConfigManager** interface. These methods search only the specified scope.

 Note:

Items stored as configuration settings can be read by all users. You should not store sensitive or personal information as configuration settings.

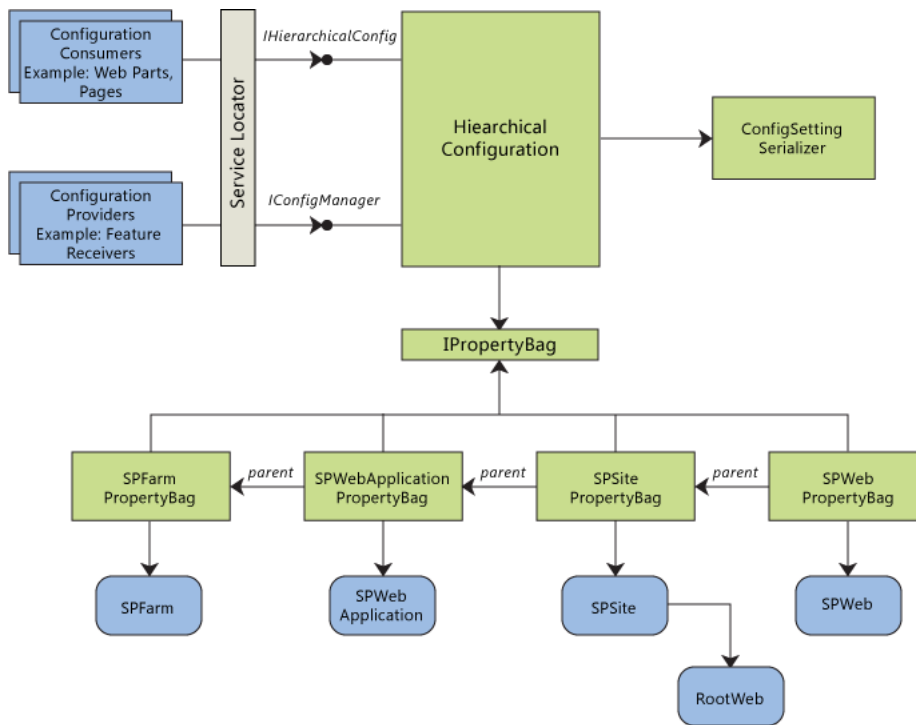
Design of the Configuration Manager

The configuration manager provides the following:

- It provides a uniform interface for reading and writing configuration settings into the property bags of **SPFarm**, **SPWebApplication**, **SPSite**, and **SPWeb** objects.
- It provides a type-safe way to read configuration settings.
- It provides automatic serialization of complex data types.
- It provides the ability to read configuration settings in a hierarchical way. Settings defined at a lower or more specific level can override settings at a higher or more general level.

The following diagram shows the design of the configuration manager.

Design of the configuration manager



The configuration manager is implemented by the **Microsoft.Practices.SPG.Common.Configuration.HierarchicalConfig** class. It is located in the **Microsoft.Practices.SPG.Common** assembly. It exposes the following two interfaces:

- **IHierarchicalConfig**. This interface consumes configuration settings.
- **IConfigManager**. This interface manages configuration settings.

These interfaces should be accessed through the service locator.

Type Safety

By using the **HierarchicalConfig** class, you can read configuration settings in a type-safe way. Type safety means that if a configuration setting was stored as a certain type, it can only be retrieved as that type. The use of generics in the interface enforces this rule. You will receive an error message at run time if you attempt to read a configuration setting with the wrong type. For example, if you read a configuration setting as type **integer** but it was set as type **datetime**, you will receive an error.

Note:

If the design of the **HierarchicalConfig** class was not based on strongly typed configuration settings, the code that consumes the configuration setting would be responsible for converting configuration settings to the correct type at run time. Strong types are less prone to error and need no extra conversion step.

Internals

The functionality of the **HierarchicalConfig** class is exposed through the **IHierarchicalConfig** and the **IConfigManager** interfaces.

Internally, the **HierarchicalConfig** class uses the **ConfigSettingSerializer** class to convert the configuration settings to and from XML.

The **HierarchicalConfig** class stores its configuration settings in the SharePoint **SPFarm**, **SPWebApplication**, **SPSite**, and **SPWeb** objects. Conceptually, all these objects have property bags, but each has a different implementation and mechanism for reading, writing, and removing data. For example:

- The property bags of **SPFarm** and **SPWebApplication** objects allow you to store objects, but the property bag

of the **SPWeb** object only supports strings.

- The **SPSite** object does not have a property bag, so the configuration manager uses the property bag of the **RootWeb** object in order to store properties at the site collection level. In order to differentiate between configuration settings at the site collection level and Web level in all cases, the configuration manager adds the prefix Site_ to the configuration settings of the **SPSite** object. An exception occurs if you attempt to set a property that has the Site_ prefix in its property name.
- Removing properties from **SPWeb** object works differently than with the other levels. The value of the property must be set to null instead of using the **Remove** method on the property bag.
- To address these differences, the **IPropertyBag** interface provides the **HierarchicalConfig** class with a uniform way to access the property bags of the individual levels. For each of the SharePoint objects (**SPFarm**, **SPWebApplication**, **SPSite**, and **SPWeb**), a class that implements the **IPropertyBag** interface is created. For example, the **SPFarmPropertyBag** class is created for an **SPFarm** object. Each of these **IPropertyBag** implementations contains the appropriate way to read and write configuration settings for that type of property bag.

Development How-to Topics

This topic describes how to use the SharePoint Guidance Library's configuration manager. It includes the following procedures:

- [How to: Add and Update Configuration Settings Using Configuration Manager](#). This topic describes how to save configuration data to any of the four levels that make up the SharePoint context.
- [How to: Remove a Configuration Setting Using Configuration Manager](#). This topic describes how to remove configuration data from any of the four levels that make up the SharePoint context.
- [How to: Retrieve Configuration Settings Using Configuration Manager](#). This topic describes how to retrieve configuration data from any of the four levels that make up the SharePoint context.

For more information about the configuration library, see [The SharePoint Guidance Library](#).

How to: Add and Update Configuration Settings Using Configuration Manager

The following procedure describes how to save configuration data at the **SPWeb**, **SPSite**, **SPWebApplication**, or **SPFarm** level.

To save configuration data

1. In Visual Studio, add a reference to the SharePoint Guidance Library **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow or anything else that needs to be in the global assembly cache, the Microsoft.Practices.SPG.Common assembly also needs to be in the global assembly cache.
2. Create an instance of the **HierarchicalConfig** class and use the **Microsoft.Practices.SPG.Common.Configuration.IConfigManager** interface.
3. Invoke the **SetInPropertyBag** method, and then pass in the key, the value, and the context.

The following code shows an example of steps 2 and 3.

```
C#
using Microsoft.Practices.SPG.Common.Configuration;
// For SPFarm.
using Microsoft.SharePoint.Administration;

IConfigManager configManager = new HierarchicalConfig();

// Add a new site property or update an existing site property.
configManager.SetInPropertyBag("MyApplications.WorkgroupName",
                               "Customer Service",
                               SPContext.Current.Web);

// Add a new site collection property or update an existing property.
configManager.SetInPropertyBag("MyApplications.DivisionName",
                               "Pharmaceuticals",
                               SPContext.Current.Site);

// Add a new Web application property or update an existing property.
configManager.SetInPropertyBag("MyApplications.CompanyName",
                               "Contoso",
                               SPContext.Current.Site.WebApplication);

// Add a new farm property or update an existing property.
configManager.SetInPropertyBag("MyApplications.FarmLocation",
                               "Redmond",
                               SPFarm.Local);
```

The **IConfigManager** interface defines how to manage configuration settings at a specific level of the SharePoint context. The **IHierarchicalConfig** interface defines how to retrieve a configuration value by traversing the SharePoint context hierarchy, beginning with the **SPWeb** level.

Note:

It is a best practice to use the service locator to retrieve instances of the **IConfigManager** and **IHierarchicalConfig** interfaces.

How to: Remove a Configuration Setting Using Configuration Manager

The following procedure describes how to remove configuration data at the **SPWeb**, **SPSite**, **SPWebApplication**, or **SPFarm** level.

To remove configuration data

1. In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow or anything else that needs to be in the global assembly cache, the Microsoft.Practices.SPG.Common assembly also needs to be in the global assembly cache.
2. Create an instance of the **HierarchicalConfig** class and use the **Microsoft.Practices.SPG.Common.Configuration.IConfigManager** interface.
3. Invoke the **RemoveKeyFromPropertyBag** method, and then pass in the key and the context.

The following code shows an example of steps 2 and 3.

```
C#
using Microsoft.Practices.SPG.Common.Configuration;
// For SPFarm.
using Microsoft.SharePoint.Administration;

IConfigManager configManager = new HierarchicalConfig();

configManager.RemoveKeyFromPropertyBag("MyApplications.Workgroupname",
                                       SPContext.Current.Web);

configManager.RemoveKeyFromPropertyBag("MyApplications.DivisionName",
                                       SPContext.Current.Site);

configManager.RemoveKeyFromPropertyBag("MyApplications.CompanyName",
                                       SPContext.Current.Site.WebApplication);

configManager.RemoveKeyFromPropertyBag("MyApplications.FarmLocation",
                                       SPFarm.Local);
```

The **IConfigManager** interface defines how to manage configuration settings at a specific level of the SharePoint context. The **IHierarchicalConfig** interface defines how to retrieve a configuration value by traversing the SharePoint context hierarchy, beginning with the **SPWeb** level.

How to: Retrieve Configuration Settings Using Configuration Manager

This procedure demonstrates how to retrieve configuration data at the **SPWeb**, **SPSite**, **SPWebApplication**, or **SPFarm** level. The configuration manager traverses the context hierarchy, starting at the current **SPWeb** level, until it finds a setting with a matching key.

1. In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow or anything else that needs to be in the global assembly cache, the **Microsoft.Practices.SPG.Common** assembly also needs to be in the global assembly cache.
2. Create an instance of **HierarchicalConfig** and use the **Microsoft.Practices.SPG.Common.Configuration.IHierarchicalConfig** interface.
3. Invoke the **GetByKey<T>** method and pass in the key.

The following code shows an example of steps 2 and 3.

```
C#  
using Microsoft.Practices.SPG.Common.Configuration;  
  
IHierarchicalConfig hierarchicalConfig = new HierarchicalConfig();  
  
string workGroupName =  
hierarchicalConfig.GetByKey<string>("MyApplications.WorkgroupName");
```

The **IConfigManager** interface defines how to manage configuration settings at a specific level of the SharePoint context. The **IHierarchicalConfig** interface defines how to retrieve a configuration value by traversing the SharePoint context hierarchy, beginning with the **SPWeb** level.

List-Based Repositories

A repository is a design pattern that separates a data source from its associated business logic by encapsulating the data access details. Repositories translate the underlying data representation into an entity model that fits the problem domain. For example, a repository can translate a SharePoint list item into a business entity such as a product detail record. For more information about the repository pattern, see [The Repository Pattern](#).

The SharePoint Guidance Library includes classes that help you to implement repositories for SharePoint lists. Accessing SharePoint lists with a repository makes it easier to reuse, maintain, and test your code. The following are some of the benefits of using repositories with SharePoint lists.

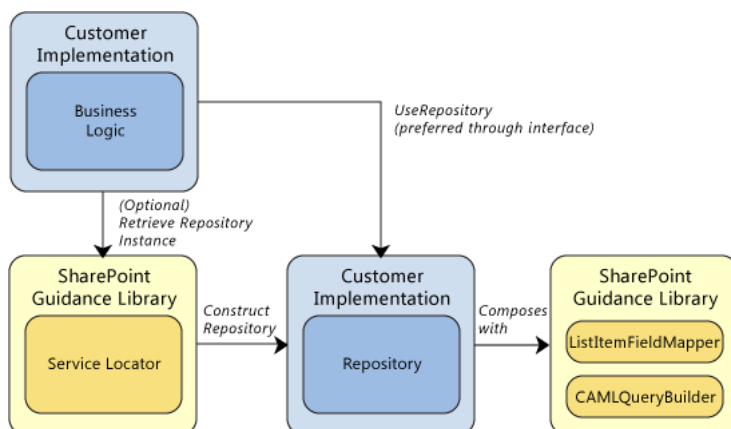
- Centralized logic for accessing the data store eliminates data dependencies in your business logic. The application has a single implementation for common list-access operations instead of having this code repeated throughout the business logic. This saves development effort.
- Encapsulating the SharePoint list access details reduces the likelihood that developers will write code that has poor performance or unacceptable security characteristics. It is recommended that an experienced infrastructure team write repositories that can be used by the rest of the development team. Accessing lists often requires SharePoint object model methods that must be used very carefully. For more information about working with the SharePoint object model, see [Best Practices: Common Coding Issues When Using the SharePoint Object Model](#) on MSDN.
- Repository implementations are replaceable for purposes such as unit testing and sustained engineering. For example, a unit test framework can substitute the implementation of the repository class with a mock object or stub in order to test the application's business logic in isolation. Also, if you decide later to reengineer your application to use a different data source, you only need to replace the repository. It is also recommended that you use the service locator pattern to make the repository implementation easier to replace.

Note:

The SharePoint Guidance Library uses composition instead of class inheritance for list repositories. The library has helper classes that you invoke from within your own repository implementation. This gives more flexibility in how you build repositories because you do not need to derive from a particular base class that is provided by the library. Your repository can use any base class you choose or no base class at all.

The following diagram shows a typical repository that is based on the SharePoint Guidance Library.

Repository based on the SharePoint Guidance Library



In this model, business logic retrieves a repository implementation through the service locator. The service locator maps the requested repository interface to a concrete type and creates an instance of that type.

The business logic uses an interface to invoke the operations that are provided by the repository. These operations retrieve and store strongly-typed entities that correspond to SharePoint list items. The repository can also include methods that retrieve a set of business entities based on user-provided selection criteria.

The **ListItemFieldMapper** and **CAMLQueryBuilder** helper classes that are provided by the SharePoint Guidance Library simplify the repository implementation. These classes help to translate between the list item and the business entity and to query the underlying list for information. The following are the classes:

- **CAMLQueryBuilder**. This class has methods for the most commonly used filter expressions. You can either filter by content type or use field-matching filters. This eliminates the need to write Collaborative Application Markup Language (CAML) for these conditions. For expressions that are more complex than "is equal to" and "is not equal to," you can add additional filters by using CAML to specify query constraints. Multiple filter expressions are always applied as an AND condition by the **CAMLQueryBuilder** class.

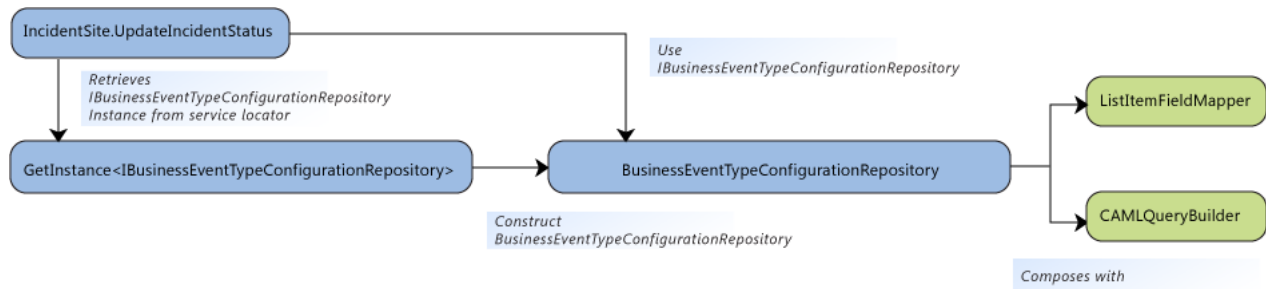
Note:

The **CAMLQueryBuilder** class is a simple implementation that works well for many cases. If you need more complex query support and do not want to write CAML, there are open source implementations such as the .NET Framework class library CAML.NET or U2U CAML Query Builder, with which you can create the CAML queries interactively. For more information, see [Useful Development Tools](#).

- **ListItemFieldMapper**. This class converts strongly-typed business entities to and from **SPListItem** objects. It specifies the fields in the **SPListItem** object that map to properties in the business entity. The **ListItemFieldMapper** class then uses this mapping to create and populate business entity values from the **SPListItem** object or to update an **SPListItem** object from values in a business entity.

The following diagram demonstrates the structure of a SharePoint list repository in the Partner Portal application.

Structure of a SharePoint list repository



The repository implementation uses an **SPListItem** object from the SharePoint object model to access data. This object is an untyped representation of an item in a list. With this approach, an **SPListItem** can represent any type of tabular information that is similar to a data row in a database. The repository translates this untyped representation into a strongly-typed class such as the Partner Portal's **Incident** object. The internal logic of the repository maps the untyped **SPListItem** objects to the strongly-typed business entities and back again.

The following topics describe the helper classes and show how to use them:

- [Key Scenarios](#). This section explains common situations for using the repository helper classes.
- [List Repository Design](#). This section explains the design and implementation of the repository helper classes.
- [Development How-to Topics](#). This section gives step-by-step examples of how to use the repository helper classes to implement a SharePoint list repository class of your own.

The implementation of the classes is located in the **Microsoft.Practices.SPG.Common** assembly and the **Microsoft.Practices.SPG.Common.ListRepository** namespace.

Key Scenarios

The SharePoint Guidance Library provides classes that assist you in building repositories for SharePoint lists. Every SharePoint list repository must perform the following two tasks:

- Build CAML queries to query the SharePoint list.
- Map to and from the fields of a SharePoint list item and the public properties of a business entity class that you define.

The following sections describe the scenarios:

- [Creating a Repository](#). This scenario demonstrates how to create a repository class and the corresponding business entity class.
- [Submitting a Query to a List](#). This scenario demonstrates how to use a CAML query to retrieve data from a list.
- [Populating a Business Entity from a List Item](#). This scenario demonstrates how to convert the results of a CAML query into strongly-typed business entities.
- [Saving a Business Entity to a List](#). This scenario demonstrates how to update a list item with a strongly-typed business entity.

Creating a Repository

Typical Goals

In this scenario, you want to create a repository class for a specific content or data type in a SharePoint list.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**.

Define a repository class and a class for the business entity that corresponds to the content type fields that are found in each of your list items. Optionally, you can define an interface that is implemented by your repository class.

Note:

Even though it is optional to define an interface for the repository, it is recommended that you do so. This decouples consumers from the implementation and allows you to use the service location pattern.

Establish mappings between the public properties of the business entity and the fields of the SharePoint list items that you will access using the repository. First, build the field mappings with an instance method of the **ListItemFieldMapper<T>** class. This class is instantiated with the business entity class as the type parameter T. Next, call the list item field mapper's **AddMapping** method for each field of the content type that corresponds to a public property of the business entity.

Creating a Repository

The Partner Portal application's **BusinessEventTypeConfigurationRepository** class demonstrates how to create a repository. The repository uses the **BusinessEventTypeConfiguration** class as the business entity. Here is the definition of the **BusinessEventTypeConfiguration** business entity class.

C#

```
public class BusinessEventTypeConfiguration
{
    public string SiteTemplate { get; set; }
    public string BusinessEventIdentifierKey { get; set; }
    public string TopLevelSiteRelativeUrl { get; set; }
}
```

Generally, it is recommended that you create a static class for each list or content type that contains the field GUIDs as read-only members. This improves the readability of the code and eliminates errors that can result from specifying field GUIDs in multiple places. It is also recommended to use GUIDs over internal names as internal names are much more likely to be reworded and updated than GUIDs.

The following code shows how the repository uses the list item field mapper to establish field mappings between the Business Event Type Configuration list in SharePoint and the **BusinessEventTypeConfiguration** business entity class.

C#

```
public class BusinessEventTypeConfigurationRepository :
    IBusinessEventTypeConfigurationRepository
{
    ListItemFieldMapper<BusinessEventTypeConfiguration> listItemFieldMapper =
        new ListItemFieldMapper<BusinessEventTypeConfiguration>();

    public BusinessEventTypeConfigurationRepository()
    {
        listItemFieldMapper.AddMapping(
            FieldIds.SiteTemplateFieldId,
            "SiteTemplate");
        listItemFieldMapper.AddMapping(
            FieldIds.BusinessEventIdentifierKeyFieldId,
            "BusinessEventIdentifierKey");
        listItemFieldMapper.AddMapping(
            FieldIds.TopLevelSiteRelativeUrlFieldId,
            "TopLevelSiteRelativeUrl");
    }
    // ...
}
```

The repository composes the **ListItemFieldMapper<T>** by creating a private instance field of the **ListItemFieldMapper<T>** class. The mapper uses the **BusinessEventTypeConfiguration** type. This represents the business entity that contains the data from the SharePoint list item.

The **AddMapping** method is invoked with the list field GUID identifier and the entity property name for each value that is mapped between the list item and the business entity. This step tells the **ListItemFieldMapper** object how to translate between them.

For implementation details about this example, see the **BusinessEventTypeConfigurationRepository.cs** and **BusinessEventTypeConfiguration.cs** files of the

Microsoft.Practices.SPG.SubSiteCreation\BusinessEventTypeConfiguration directory in the reference implementation.

Usage Notes

For predictable results when calling the **AddMapping** method, make sure that each SharePoint field ID and each business entity property name is unique. You should also make sure there is type consistency between the fields of list and the properties of your business entity. The **ListItemFieldMapper**'s **AddMapping** method does not throw exceptions for duplicate names or IDs, or for type inconsistencies between the SharePoint list and the business entity class. If these mappings are not correct, the **ListItemFieldMapper** class throws a **ListItemFieldMappingException** when an entity instance is created.

Submitting a Query to a List

Typical Goals

In this scenario, you want to retrieve list items from a SharePoint list based on a set of filter conditions.

Solution

SharePoint provides **SPQuery** objects to retrieve data from SharePoint lists. An **SPQuery** object uses the [CAML](#) query language to execute data queries. The **CAMLQueryBuilder** class simplifies, and in many cases, eliminates the need to write CAML queries in XML. The **CAMLQueryBuilder** class uses filter criteria to create an **SPQuery** object to query a list.

You build a query by invoking query builder methods such as **AddEqual**, **AddNotEqual**, and **FilterByContentType**. You can add an arbitrary CAML filter expression with the **AddFilter** method. Then, you use the **Build** method to retrieve an **SPQuery** object that contains your query. Finally, you call **SPList.GetItem**s method of the list you want to query. You provide the query object as an argument. This invocation returns an **SPListItemCollection** object that contains the list items that satisfy the query.

Submitting a Query to a List

The **GetBusinessEventTypeConfiguration** method of the **BusinessEventTypeConfigurationRepository** class demonstrates how to query a list. This is shown in the following code.

```
C#
public BusinessEventTypeConfiguration GetBusinessEventTypeConfiguration(
    string businessEvent)
{
    // ...

    string adminWebUrl = // ...

    using (SPSite site = new SPSite(adminWebUrl))
    {
        using (SPWeb adminWeb = site.OpenWeb())
        {
            SPList businessEventSiteTemplateList =
                adminWeb.Lists[Constants.BusinessEventTypeConfigListName];
            CAMLQueryBuilder camlQueryBuilder = new CAMLQueryBuilder();
            camlQueryBuilder.AddEqual(FieldIds.BusinessEventFieldId, businessEvent);

            SPListItemCollection items =
                businessEventSiteTemplateList.GetItem(camlQueryBuilder.Build());

            // ...
        }
    }
}
```

In this example, a **CAMLQueryBuilder** object is constructed and the filter condition is defined. The query matches the **businessEvent** parameter to the **BusinessEventFieldId** in the list. This query is then submitted to the list through the **GetItems** method to retrieve all **SPListItem** objects that match the filter criteria.

For implementation details, see `BusinessEventTypeConfigurationRepository.cs` located in the `Microsoft.Practices.SPG.SubSiteCreation\BusinessEventTypeConfiguration` directory.

Populating a Business Entity from a List Item

Typical Goals

In this scenario, you want to convert the results of queried list items into one or more strongly-typed business entity objects. This scenario is possible after you create a repository and establish field mappings between a SharePoint list and a business entity class.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. Use the **CreateEntity** method of the **ListItemFieldMapper<T>** class to create and populate the entity from a list item.

Populating a Business Entity from a List Item

The **GetBusinessEventTypeConfiguration** method demonstrates how to use the **CreateEntity** method. For more information about the **GetBusinessEventTypeConfiguration** method, see [Creating a Repository](#).

C#

```
public BusinessEventTypeConfiguration GetBusinessEventTypeConfiguration(
    string businessEvent)
{
    // ...

    SPListItemCollection items =
        businessEventSiteTemplateList.GetItems(camlQueryBuilder.Build());

    if (items.Count > 0)
    {
        return this.listItemFieldMapper.CreateEntity(items[0]);
    }
    else
    {
        throw new SubSiteCreationException(string.Format(CultureInfo.CurrentCulture,
            ConfigDataNotFoundMessage, businessEvent));
    }
}
```

Usage Notes

To see how the mapper is configured, see [Creating a Repository](#). The result is a new business entity instance whose properties have been set to contain the values of the mapped fields from the list item. The list item is the argument to the **CreateEntity** method.

Saving a Business Entity to a List

Typical Goals

In this scenario, you want to use a repository to update a list item. A business entity is the source of the update.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. Use the **FillSPLListItemFromEntity** method of the **ListItemFieldMapper<T>** class to perform the conversion.

Saving a Business Entity to a List

When developing your repository, remember that you may need to save information from the business entity back to the SharePoint list as users update and add business information. The following code demonstrates how the **SubSiteCreationRequestsRepository** class implements an **AddSubSiteCreationRequest** method. This method adds a new list item that is based on the business entity **SubSiteCreationRequest**.

C#

```
public void AddSubSiteCreationRequest(SubSiteCreationRequest request)
{
    // ...

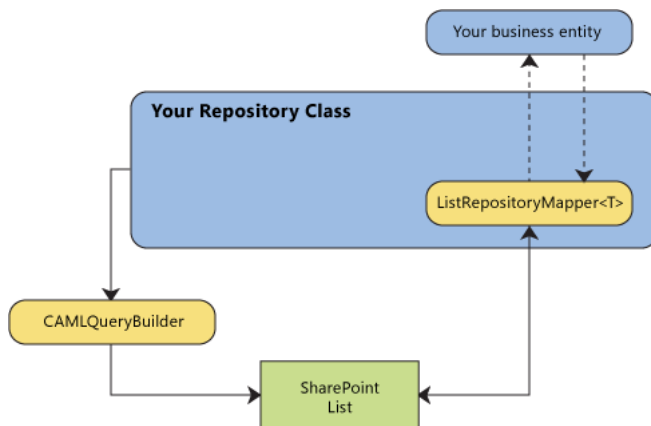
    string adminWebUrl = // ...
    using (SPSite site = new SPSite(adminWebUrl))
    {
        using (SPWeb adminWeb = site.OpenWeb())
        {
            SPList subSiteCreationRequests =
                adminWeb.Lists[Constants.SubSiteRequestsListName];
            SPLListItem subSiteCreationRequest = subSiteCreationRequests.Items.Add();
            this.listItemFieldMapper.FillSPLListItemFromEntity(
                subSiteCreationRequest, request);
            subSiteCreationRequest.Update();
        }
    }
}
```

For implementation details, see `SubSiteCreationRequestsRepository.cs` in the `Microsoft.Practices.SPG.SubSiteCreation\SubSiteCreationRequests` directory.

List Repository Design

The repository's functionality is based on the principle of composition instead of on inheritance. In this model, instead of deriving from a base class, the SharePoint Guidance library provides components that make it easier for you to build your own repository implementation. The following diagram describes how these components can simplify the development of a repository.

Design of the list repository



The implementation contains an instance of a **ListItemFieldMapper** class. This class is not used directly by client code. Instead, you typically use it within methods of your repository to transfer information between the SharePoint list and the business entity. The implementation is a variation of a common pattern that is named **Data Mapper**. For more information, see [Data Mapper](#) on Martin Fowler's Web site.

Because you typically do not retain an instance member for the **CAMLQueryBuilder** class, it is not strictly composed with your repository. It is closer to a utility class that simplifies the retrieval of information from a list. The **CAMLQueryBuilder** class, or an alternative such as [CAML.NET](#), provides more readable code, compile-time validation, and error reduction compared to writing your own CAML queries.

Role of Service Locator and Repository Interface in Implementation

Using a repository improves an application's design by centralizing the business access logic, providing strong types for compile-time type checking and encapsulation, and decoupling business logic from SharePoint-specific dependencies. These are all significant benefits for code maintenance, correctness, and flexibility.

Although implementing the repository pattern is a recommended practice, even without service location, these patterns are typically used in conjunction with one another. Service location provides an additional layer of abstraction that hides the concrete implementation of the repository from the consuming business logic. When using service location, you retrieve an instance of the repository based on the defined interface. This means that, to use service location, you must define an interface and implement the interface for your repository. The Partner Portal application follows this pattern for all of its repositories.

For more information about the service locator, see [The Service Locator Pattern](#).

Note:

It is important to validate the data that is being persisted. When using SharePoint lists as a persistence mechanism, it is recommended to use list item event receivers such as **ItemAdding** and **ItemUpdating** to perform validation. These event receivers can cancel the add/update operation if the list item data is invalid.

Development How-to Topics

This topic describes how to use the SharePoint Guidance Library's helper classes for list repositories. It includes the following procedures:

- [How to: Create a List Repository](#). This topic describes how to create a repository class for a SharePoint list.
- [How to: Submit a Query to a List](#). This topic describes how to use the **CAMLQueryBuilder** class to retrieve a list item from a SharePoint list.
- [How to: Populate a Business Entity from a List Item](#). This topic demonstrates how to convert a list item into a strongly-typed business entity.
- [How to: Save a Business Entity to a List](#). This topic demonstrates how to convert a business entity into a list item and then add it to a SharePoint list.

For more information about the SharePoint Guidance Library, see [The SharePoint Guidance Library](#).

How to: Create a List Repository

The following procedure demonstrates how to create a list repository.

To create a list repository

1. In Visual Studio, add a reference to the SharePoint Guidance Library **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow or anything else that must be in the global assembly cache, then the Microsoft.Practices.SPG.Common assembly also needs to be in the global assembly cache.
2. Define a class for the business entity that corresponds to the content type fields. For information on how to create a content type and associate the content type to a list instance, see [Content Types](#).

The following code shows an example of this step.

C#

```
public class Customer
{
    public string CustomerId { get; set; }
    public string Name { get; set; }
    public int SalesTerritoryZone { get; set; }
}
```

3. Define a static class that contains the SharePoint field IDs for the list you want to use.

The following code shows an example of this step.

C#

```
public static class CustomerFieldIds
{
    public static readonly Guid IdFieldId =
        new Guid("3c9699e9-2f9b-4cd6-845e-76c7e58d129a");

    public static readonly Guid NameFieldId =
        new Guid("84267e40-7f47-4f40-b3be-4004312eb467");

    public static readonly Guid TerritoryZoneFieldId =
        new Guid("7106DC30-31D9-420d-969F-24A22B7AB7CD");
}
```

4. Define a repository class and create an instance of the **ListItemFieldMapper** class as a member. In the repository's constructor, add mappings that associate the field IDs of the content type to the corresponding property of the business entity. Use a string to represent the name of the business entity property. The following code shows an example.

C#

```
public class CustomerRepository: ICustomerRepository
{
    ListItemFieldMapper<Customer> listItemFieldMapper =
        new ListItemFieldMapper<Customer>();

    public CustomerRepository()
    {
        listItemFieldMapper.AddMapping(
            CustomerFieldIds.IdFieldId,
            "CustomerId");

        listItemFieldMapper.AddMapping(
            CustomerFieldIds.NameFieldId,
            "Name");

        listItemFieldMapper.AddMapping(
            CustomerFieldIds.TerritoryZoneFieldId,
            "SalesTerritoryZone");
    }
}
```

How to: Submit a Query to a List

The following procedure demonstrates how to use the **CAMLQueryBuilder** class to submit a query to a list.

To submit a query to a list

1. In Visual Studio, add a reference to the SharePoint Guidance Library **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow or anything else that needs to be in the global assembly cache, the Microsoft.Practices.SPG.Common assembly also needs to be in the global assembly cache.
2. Define the query that your repository will provide.
3. Open the **SPSite** and **SPWeb** objects that contain the list. Remember to properly dispose of these objects when you finish.
4. Use an instance of the **CAMLQueryBuilder** class to create an **SPQuery** object. Pass the **SPQuery** object to the list's **GetItems** method.

The following code shows an example of the preceding steps.

C#

```
public IList<Customer> GetAllCustomers()
{
    // ...

    using (SPSite site = new SPSite(customerWebUrl))
    {
        using (SPWeb customerWeb = site.OpenWeb())
        {
            SPList customerList =
                customerWeb.Lists[Constants.CustomerListName];
            CAMLQueryBuilder camlQueryBuilder = new CAMLQueryBuilder();
            camlQueryBuilder.FilterByContentType(CustomerContentTypeName);
            SPListItemCollection items =
                customerList.GetItems(camlQueryBuilder.Build());

            // ...
        }
    }
}
```


How to: Populate a Business Entity from a List Item

The following procedure demonstrates how to convert the results of a query into strongly-typed business entities. To perform this procedure, you must first create a repository and establish field mappings between a SharePoint list and a business entity class.

To populate a business entity

1. In Visual Studio, add a reference to the SharePoint Guidance Library **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow or anything else that must be in the global assembly cache, then the Microsoft.Practices.SPG.Common assembly also needs to be in the global assembly cache.
2. Use an instance of the **ListItemFieldMapper** class that is populated with the list item Field IDs.
3. Convert the **SPListItem** instances into business entities with the **ListItemFieldMapper.CreateEntity** method. The following code shows an example.

C#

```
ICollection<Customer> customers = new List<Customer>();
foreach(SPListItem item in items)
{
    Customer customer = listItemFieldMapper.CreateEntity(item);
    customers.Add(customer);
}
```

The following code shows an expanded version of the query method.

C#

```
public ICollection<Customer> GetAllCustomers()
{
    // ...

    using (SPSite site = new SPSite(customerWebUrl))
    {
        using (SPWeb customerWeb = site.OpenWeb())
        {
            SPList customerList =
                customerWeb.Lists[Constants.CustomerListName];
            CAMLQueryBuilder camlQueryBuilder = new CAMLQueryBuilder();
            camlQueryBuilder.FilterByContentType(CustomerContentTypeName);
            SPListItemCollection items =
                customerList.GetItems(camlQueryBuilder.Build());

            ICollection<Customer> customers = new List<Customer>();
            foreach(SPListItem item in items)
            {
                Customer customer = listItemFieldMapper.CreateEntity(item);
                customers.Add(customer);
            }
            return customers;
        }
    }
}
```

How to: Save a Business Entity to a List

The following procedure demonstrates how to convert a business entity to a list item and save it.

To add a business entity to a list

1. In Visual Studio, add a reference to the SharePoint Guidance Library **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow or anything else that needs to live in the GAC, the **Microsoft.Practices.SPG.Common** assembly also needs to be in the GAC.
2. Define a method that takes a business entity instance as a parameter.
3. Open the **SPSite** and **SPWeb** objects that contain the list. Remember to properly dispose of the **SPSite** and **SPWeb** objects when you finish.
4. Call the **Add** method of the list's **Items** collection to get an empty **SPListItem** instance.
5. Populate the blank **SPListItem** instance with the values in the business entity.
6. Update the list.

The following code demonstrates the preceding steps.

C#

```
public void AddCustomer(Customer customer)
{
    // ...

    using (SPSite site = new SPSite(customerWebUrl))
    {
        using (SPWeb customerWeb = site.OpenWeb())
        {
            SPList customerList =
                customerWeb.Lists[Constants.CustomerListName];
            SPListItem customerListItem = customerList.Items.Add();
            this.listItemFieldMapper.FillSPListItemFromEntity(
                customerListItem, customer);
            customerList.Update();
        }
    }
}
```

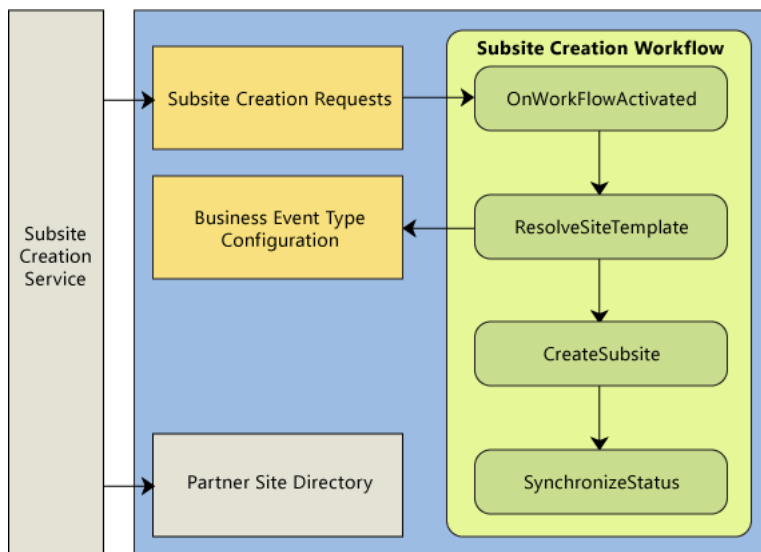
Workflow-Driven Site Creation

A common SharePoint task is to create collaboration Web sites. In many cases, these sites share similar characteristics. Typically, they are created programmatically, in response to a business event or as part of a business process or workflow.

The SharePoint Guidance Library provides a subsite creation component. The subsite creation component creates subsites that are based on site templates and site definitions. This is a reusable component that you can include in your own applications. The component includes a SharePoint sequential workflow and configuration data that is stored in a SharePoint list. For more information about the SharePoint Guidance Library's subsite creation component, see [Event-Driven Site Creation](#).

The subsite creation component implements the subsite pattern. This pattern is a series of steps that make up the entire creation process. The pattern can be adapted to fit a business's particular needs. For example, one or more of the steps can be customized or additional steps can be added. The process of creating subsites may require interaction with other systems or with users. This is why a workflow governs the creation process. The following figure illustrates the subsite creation process that is used for the Partner Portal application.

Subsite creation process



Notice that all the creation steps are included within the workflow.

Note:

Many businesses want to implement security restrictions that prevent users from creating subsites independently of the subsite creation workflow. For example, the Partner Portal application requires that incident collaboration sites can only be created by using a particular Web service. Because the site creation process relies on a workflow, administrators can restrict access to the Sub Site Creation Requests list to specific users. If you are using a service interface for the Sub Site Creation Requests list, you can restrict access to the list to only the service's service account.

The SharePoint Guidance Library provides a Web Solution Package (WSP) for deploying and installing all the components that are required for the subsite creation component. For more information about installing the SharePoint Guidance Library, see [SharePoint Guidance Library](#) in this guidance.

The MicrosoftSPGSubSiteCreationSite feature installs and activates the subsite creation workflow. The MicrosoftSPGSubSiteCreationWeb feature installs the Business Event Type Configuration and the Sub Site Creation Requests lists. It also associates the subsite creation workflow with the Sub Site Creation Requests list.

This section includes the following topics:

- [Key Scenarios](#)
- [Design of the Subsite Creation Feature](#)
- [How to: Create a Custom Subsite Creation Workflow](#)

Key Scenarios

The following topics explain the key scenarios for the SharePoint Guidance Library's subsite creation component:

- [Configuring a Business Event for Subsite Creation](#)
- [Customizing Subsite Creation Workflow](#)

Configuring a Business Event for Subsite Creation

Typical Goals

In this scenario, you want to programmatically create a new SharePoint subsite in response to a business event.

Solution

Install the SharePoint **MicrosoftSPGSubSiteCreation** Web solution package (WSP) and activate the two features that it contains. It contains the following features:

- Microsoft patterns & practices Subsite Creation Workflow (Site)
- Microsoft patterns & practices Subsite Creation Infrastructure (Web)

The Subsite Creation Workflow (Site) feature installs the subsite creation workflow. The Subsite Creation Infrastructure (Web) feature creates instances of the Business Event Type Configuration and the Sub Site Creation Requests lists. It automatically associates the subsite creation workflow with the Sub Site Creation Requests list. The following procedure shows how to configure the subsite creation workflow with a new business event.

To add a new business event to the subsite creation workflow

1. Navigate to the site where the Microsoft patterns & practices Subsite Creation Infrastructure (Web) feature was activated.
2. Using the Quick Launch, navigate to the Business Event Type Configuration list.
3. Create a new Business Event Type Configuration list item. To do this, do the following:
 - a. Provide any value for the business event. This value will be used later.
 - b. Provide a valid site template name or site definition configuration. For example, "incidentsubsite.stp" or "STS#0".
 - c. Provide a value for the Business Event Identifier Key.
 - d. Provide a value for the top-level site where the subsite will be created. Do not include a slash mark at the beginning of the path name.
4. Validate that the subsite creation workflow is associated with the Sub Site Creation Requests lists. To do this, do the following:
 - a. Using the Quick Launch, navigate to the Sub Site Creation Requests list.
 - b. On the **Settings** menu, click **List Settings**.
 - c. Click the **Workflow Settings** link.
 - d. Verify that subsite creation workflow is in the list. If it is not in the list, use the **Add a workflow** link to associate the subsite creation workflow to the Sub Site Creation Requests list.
5. Test by queuing a new subsite creation request (Option 1). To do this, do the following:
 - a. Navigate back to the Sub Site Creation Requests list. Create a new list item.
 - b. Provide the name of the business event from step 3a.
 - c. Provide any value for the Event ID.
 - d. Provide a valid URL of the site collection where the subsite should be created.
6. Test by queuing a new subsite creation request (Option 2). To do this, do the following:
 - a. Use the following code to programmatically queue a new subsite creation request.

C#

```
SubSiteCreationRequest request = new SubSiteCreationRequest();
// Use the business event from step 3a.
request.BusinessEvent = businessEvent;
request.EventId = eventIdentifier;
// Provide a valid site collection where the subsite should be created.
request.SiteCollectionUrl = siteCollectionUrl;

SubSiteCreationRequestsRepository repository = new SubSiteCreationRequestsRepository();
repository.AddSubSiteCreationRequest(request);
```

Customizing Subsite Creation Workflow

Typical Goals

In this scenario, you want to customize the workflow that is associated with a subsite creation process. For example, you want to add, update, or remove activities that will occur when a business event occurs.

Solution

You can either modify the subsite creation workflow or create a new custom workflow that reuses some of the activities provided by the SharePoint Guidance Library.

Modifying the Subsite Creation Workflow

Use the Visual Studio 2008 Workflow designer surface to customize the subsite creation workflow. The designer surface supports dragging activities from the Toolbox. Add any new activities you need for your workflow or remove any of the activities that are already included in the subsite creation workflow.

Reusing Activities in a New Custom Workflow

Use the Visual Studio 2008 Workflow designer surface to create a custom workflow.

To reuse the activities in the SharePoint Guidance Library, you must either add a reference to the **Microsoft.Practices.SPG.SubSiteCreation** assembly or add the **Microsoft.Practices.SPG.SubSiteCreation** project to your Visual Studio solution. If you add the reference to the assembly, you must perform the following additional steps to see the reusable activities in the Visual Studio Toolbox.

To see the reusable activities

1. Right-click any tab in the Toolbox, and then click **Choose items**.
2. Click the **Activities** tab.
3. Click the **Browse** button.
4. Select the **Microsoft.Practices.SPG.SubSiteCreation.dll** file, and then click **Open**.
5. Click **OK**.

Next, use the Toolbox to add activities as needed from the SharePoint Guidance Library. For each activity that you add, provide input values for the dependency properties defined for that activity. For example, the **ResolveSiteTemplateActivity** class has a dependency property named **BusinessEventName** that must have a valid value at run time. You can provide a literal value or bind to another property or field using the Properties window in Visual Studio.

Note:

If you are using a 64-bit development server, it is important that you do not use either the SharePoint 2007 Sequential Workflow or SharePoint 2007 State Machine Workflow project templates. There are known incompatibilities with these project templates and 64-bit SharePoint assemblies. However, you can create a standard sequential workflow or state machine workflow. When using the standard workflow project templates, remember to add the **OnWorkflowActivated** activity at the beginning of a new custom workflow. You must also manually bind the **CorrelationToken** and **WorkflowProperties** dependency properties.

For more detailed step-by-step instructions for creating your custom workflow for creating a subsite, see [How to: Create a Custom Subsite Creation Workflow](#).

Design of the Subsite Creation Feature

The Business Event Type Configuration List

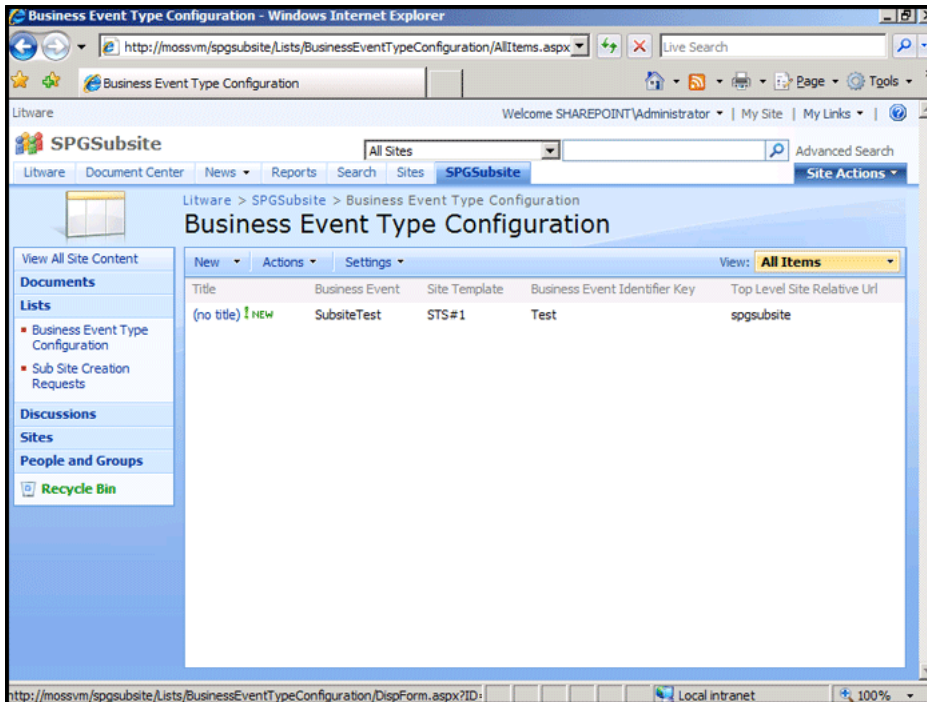
Creating sites in SharePoint requires the following two pieces of information:

- It requires a site template or site definition.
- It requires the address where the site should be created.

The subsite creation component includes a list that contains this information and other configuration data that is used by the subsite creation workflow at run time. The list maps a business event to the site template or definition and to the relative URL of the parent site.

The following figure illustrates an example of the configuration data.

Example of configuration data



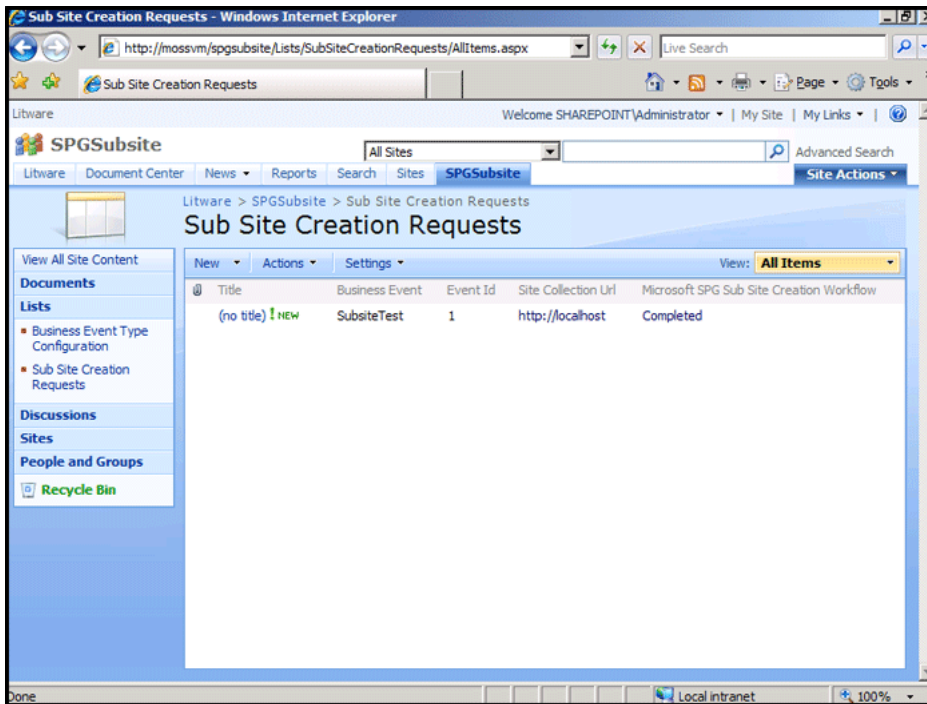
The Sub Site Creation Requests List

All workflows in SharePoint must be associated with a list and initiated when a list item is added or updated. The Sub Site Creation Requests list acts as a queue for processing subsite creation requests. When a new SharePoint subsite is needed, an external business process adds a new item to the list. The following three pieces of information are contained in this list:

- The business event
- The identifier of the external event
- The URL of the site collection where the subsite will be created

The following figure illustrates an example of an item in the Sub Site Creation Requests list.

Example of item in the Sub Site Creation Requests list



Subsite Creation Workflow

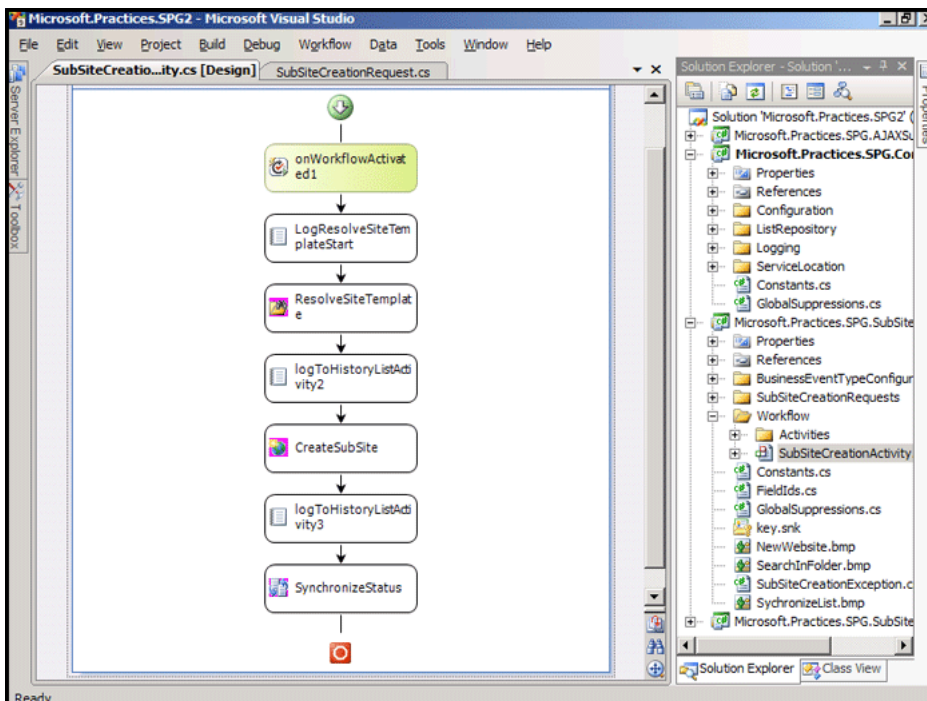
The subsite creation workflow is based on the sequential workflow activity. The subsite creation workflow is composed of the following three custom activities:

- It resolves the name of the site template or site definition.
- It creates the subsite.
- It synchronizes the status of the site.

Inputs and outputs to the individual activities are bound by dependency properties. The activities provided in the workflow can be replaced with other custom activities. The workflow also contains a fault handler that writes an error message to the workflow history list. This allows users to see if any errors occurred during any of the workflow activities.

The following figure is the designer view of the subsite creation workflow.

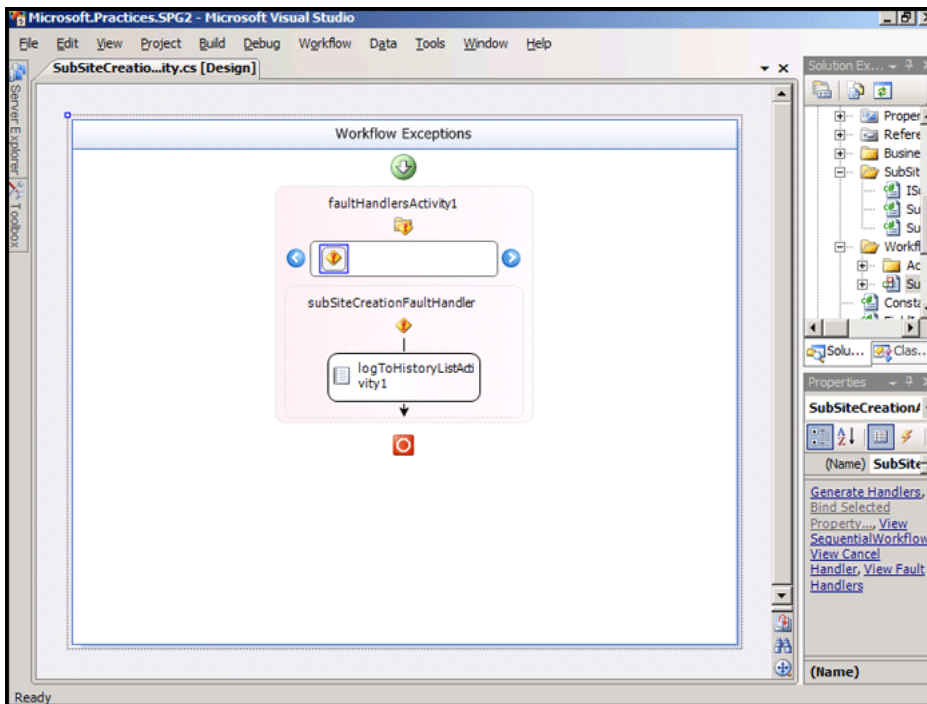
Designer view of subsite creation workflow



The context menu of the workflow start node allows you to view the fault handlers.

The following illustration shows the designer view of the fault handler.

Designer view of fault handler



ResolveSiteTemplateActivity

The **ResolveSiteTemplateActivity** resolves three pieces of information:

- It resolves the name of the site template or site definition to use when creating the subsite.
- It resolves the relative URL of the top-level site in the site collection where the subsite will be created.
- It resolves the business event key to be used to associate the subsite with a specific business event.

The only required input is the name of the business event. The **ResolveSiteTemplateActivity** queries the business event type configuration list for the business event.

CreateSubSiteActivity

The **CreateSubSiteActivity** creates the subsite. It takes the output from the **ResolveSiteTemplateActivity** to create a new subsite. In addition to creating the subsite, the **CreateSubSiteActivity** also stores the identifier of the business event in the newly created site's property bag.

SynchronizeStatusActivity

In most scenarios, the status of a business event is managed by an external system. For example, the status of an incident can be maintained in a Customer Relationship Management (CRM) system. As the status of the incident is updated in the CRM system, it should be reflected in the SharePoint subsite. The **SynchronizeStatusActivity** sets the initial status of the business event in the SharePoint subsite. As the status of the business event changes, the external system updates the status.

Note:

The Partner Portal application includes two use cases that demonstrate how to create subsites. One example is creating subsites for resolving incidents and one example is creating subsites for order exceptions.

How to: Create a Custom Subsite Creation Workflow

The SharePoint Guidance Library's site creation workflow component performs most of the standard tasks that are required to create a site. However, you may need a site workflow with custom business logic. The following procedure demonstrates how to create a custom workflow by reusing some of the activities that are included in the SharePoint Guidance Library.

To create a site creation workflow

1. In Visual Studio, point to **New** on the **File** menu, and then click **Project**. Under **Project types**, click **Workflow**. Under **Templates**, click **Empty Workflow Project**. Enter the project name, and then click **OK**.

Note:

An alternative is to choose the SharePoint 2007 Sequential Workflow project template. This option presents you with a wizard that requires an existing SharePoint site. This procedure creates its own deployment project for the workflow and does not require a wizard. Additionally, the wizard will not load on a 64-bit development server because of an incompatibility between Visual Studio and 64-bit SharePoint.

2. In Solution Explorer, right-click the project name, point to **Add**, and then click **Sequential Workflow**. Under **Templates**, click **Sequential Workflow (code)**. Name the workflow CustomSubSiteCreation.cs, and then click **Add**.
3. In Solution Explorer, right-click **References**, click **Add Reference**, and then add the following references:
 - Windows SharePoint Services
 - Windows SharePoint Services Workflow Actions
 - Microsoft.Practices.SPG.SubSiteCreation.dll (alternatively, you can add the Microsoft.Practices.SPG.SubSiteCreation.csproj to the current solution.) This project is included in the SharePoint Guidance Library which is part of the patterns & practices SharePoint Guidance.
4. Add the **OnWorkflowActivated** activity. To do this, do the following:
 - Drag the **OnWorkflowActivated** activity that is located on the **SharePoint Workflow** tab in the Toolbox to the designer surface.
 - Open the CustomSubSiteCreation.designer.cs file.
 - Locate the following line of code in the **InitializeComponent** method that is in the **CustomSubSiteCreation** class.

C#

```
this.onWorkflowActivated1 = new Microsoft.SharePoint.WorkflowActions.OnWorkflowActivated();
```

- Paste the following code in the **InitializeComponent** method that is in the **CustomSubSiteCreation** class.

C#

```
CorrelationToken customSubSiteCreationToken = new CorrelationToken();  
customSubSiteCreationToken.Name = "workflowToken";  
customSubSiteCreationToken.OwnerActivityName = "CustomSubSiteCreation";  
this.onWorkflowActivated1.CorrelationToken = customSubSiteCreationToken;
```

5. Drag a **Code** activity from the Toolbox to the designer surface. This example procedure uses a standard code activity. Your business might require complex business logic for determining the correct site template to use. If this is the case, you can create a library of custom activities.
6. Add activities from the SharePoint Guidance Library to the Toolbox. To do this, do the following:
 - In the Toolbox, right-click the **General** tab, and then click **Choose Items**.
 - Click the **Activities** tab.
 - Click **Browse**, and then navigate to **Microsoft.Practices.SPG.SubSiteCreation.dll**.
 - Click **OK**.
7. Drag the **CreateSubSite** activity to the designer surface. It should be under the **Code** activity.
8. Bind the Dependency properties for the **CreateSubSiteActivity**. To do this, do the following:
 - In the designer, right-click **CreateSubSiteActivity**, and then click **Properties**.
 - In the Properties pane, click the **BusinessEvent** property, and then click the **ellipsis** button (...).
 - In the **Bind "BusinessEvent" to an activity property** dialog box, click the **Bind to a new member** tab, and then click **OK**.
9. Perform steps 8a through 8c for the **BusinessEventId**, **BusinessEventIdKey**, **SiteCollectionUrl**, **SiteTemplateName**, **SubSiteUrl**, and **TopLevelSiteRelativeUrl** properties.
10. Implement the **ExecuteCode** event handler for the code activity. To do this, do the following:
 - In the Designer, right-click the code activity, and then click **Properties**.
 - Click the **Events** button.
 - Click **ExecuteCode**, and then click the drop-down box. Enter the method name **codeActivity1_ExecuteCode**.
 - Paste the following code into the **codeActivity1_ExecuteCode** method.

C#

```
this.CreateSubSite_BusinessEvent1 = "SubSiteCreationHowTo";  
this.CreateSubSite_BusinessEventId1 = Guid.NewGuid().ToString();  
this.CreateSubSite_BusinessEventIdKey1 = "SubSiteCreationHowToId";  
this.CreateSubSite_SiteCollectionUrl1 = "http://localhost:9001";//must be a valid  
site
```

```
this.CreateSubSite_SiteTemplateName1 = "STS#0";
this.CreateSubSite_TopLevelSiteRelativeUrl1 = "subsitecreationhowtosites";
```

11. Sign the assembly with a new .snk file. To do this, do the following:

- Right-click the project, and then click **Properties**.
- Click **Signing**.
- Select **Sign the assembly**.
- Click the **Choose a strong name key file** drop-down box, and then click **New**.
- Specify the key file name. Do not select the **Protect my key file with a password** option. Click **OK**.

The following procedure demonstrates how to create the deployment project.

To create a deployment project

1. In Visual Studio, point to **New** on the **File** menu, and then click **Project**. Under **Project types**, click **Visual C#**. Under **SharePoint**, click **Empty**. Enter the project name. In the **Solution** drop-down box, click **Add to solution**, and then click **OK**.
2. In the **Select Trust Level** dialog box, select **Full Trust(Deploy to GAC)**, and then click **OK**.
3. In Solution Explorer, right-click **References**, click **Add Reference**, click the **Projects** tab, and then add a reference to the custom site creation workflow project that you created in the previous procedure.
4. In the Properties window, make sure that the **Copy Local** option is set to **True**.
5. Add a new XML file to the project that contains the workflow definition. To do this, do the following:
 - a. Right-click the project name, and then click **Add**. Click **New Item**. In the Categories pane, click **Data**, which is listed under **Visual C#**. Click **XML File**. Name the XML file. Click **Add**.
 - b. Paste the following XML code into the XML file.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<Elements Id="57AED6C5-5CEB-4068-9617-0FE2BD6191AD"
xmlns="http://schemas.microsoft.com/sharepoint/">
  <Workflow Name="Custom Sub Site Creation Workflow"
    Description="Workflow for creating sub sites"
    Id="56880CD1-EB35-4f59-952E-E2DDE275D17E"
    CodeBesideAssembly="CustomSubSiteCreationHowTo, Culture=neutral, Version=1.0.0.0,
PublicKeyToken=fb379bdd60f32127"
    CodeBesideClass="CustomSubSiteCreationHowTo.CustomSubSiteCreation"
    TaskListContentTypeId="0x0108">
    <Categories/>
    <MetaData>
      <StatusPageUrl>_layouts/WrkStat.aspx</StatusPageUrl>
    </MetaData>
  </Workflow>
</Elements>
```

- c. Click **Tools**, click **CreateGUID**, click **Registry Format**, click **Copy**, and then click **Exit**. In the XML file, delete the GUID that is assigned to the **Id** attribute of the **Elements** element. Paste in the new GUID. Remove the surrounding braces.
 - d. In the XML file, update the **Name** attribute in the **Workflow** element to the name you want to use.
 - e. In the XML file, update the **Description** attribute in the **Workflow** element to a description you want to use.
 - f. In the XML file, repeat step c and assign the new GUID to the **Id** attribute in the **Workflow** element.
 - g. Build your project. To do this, right-click the workflow project, and then click **Build**.
 - h. Update the **CodeBesideAssembly** and **CodeBesideClass** attributes with the correct assembly information of the custom subsite creation workflow project. To do this, do the following:
 - a. Open a Visual Studio command prompt, and then change the directory to the bin\Debug directory of the custom site creation workflow project.
 - b. Use the **SN.EXE -T** command to discover the public key token. The argument is the name of the site creation workflow assembly (including the .dll file name extension).
 - c. Copy the public key token and paste it as the value of the **PublicKeyToken** attribute. Make sure that the **CodeBesideAssembly** name is the name of the site creation workflow assembly.
 - d. Make sure that the **CodeBesideClass** attribute has the namespace name and class name of the site creation workflow class.
6. Provide a valid debug start URL. To do this, do the following:
 - a. Right-click the deployment project, and then click **Properties**.
 - b. Click **Debug**.
 - c. Select **Start browser with URL**. Enter the URL of a SharePoint site.
7. Open the WSP View Window in Visual Studio, and then click the **Refresh** button.
8. Open the Feature.xml file for Untitled 1 Feature.
9. Make sure the **Scope** attribute is set to **Site**.

You now have a custom workflow and a project that can deploy and install your workflow. After the workflow is deployed and activated, you can associate the workflow with a list from which you can initiate the workflow.

The Safe Script Manager

The ASP.NET script manager is a component of the .NET Framework that manages ASP.NET AJAX script libraries and script files, partial-page rendering, and client proxy class generation for Web and application services. For more information about the **ScriptManager** control, see [ScriptManager Class](#) and [ScriptManager Control Overview](#) on MSDN. For information about the AJAX script libraries and script files, see [Adding AJAX and Client Capabilities Roadmap](#) on MSDN.

The script manager has a requirement that only one instance of the **ScriptManager** control can be added to a Web page. If you add two script managers, the second script manager throws an exception.

There are two ways to enforce this requirement:

- Ensure that the **ScriptManager** control is present on each page by adding it to all master pages. This is a good approach if almost all pages use AJAX functionality. However, adding the **ScriptManager** control to each page increases overhead because it adds the AJAX script references to the page.
- Check to see if a **ScriptManager** control is already present before adding a new **ScriptManager** control. This must be done in code, not in the ASPX or ASCX markup. Developers must remember to perform this check each time they want to use the control.

The SharePoint Guidance Library's **SafeScriptManager** class solves this issue. The **SafeScriptManager** class is an ASP.NET custom control that loads the .NET Framework's **ScriptManager** control if it is not already present on the current Web page. The safe script manager is a reusable component that you can include in your own SharePoint applications. It also includes SharePoint-specific support for AJAX update panels.

This section includes the following topics:

- [Key Scenarios](#)
- [Design](#)
- [Development How-to Topics](#)

Key Scenarios

The following section discusses how to use AJAX with a SharePoint Web part:

- [Using AJAX Capabilities with a Web Part](#)
- [Using the AJAX Update Panel in SharePoint](#)

Using AJAX Capabilities with a Web Part

Typical Goals

You want to use the capabilities of the ASP.NET AJAX library within a SharePoint Web Part. For example, this scenario occurs if your Web Part contains a Silverlight control for displaying media. Silverlight controls require AJAX.

Solution

In Visual Studio, add a reference to **Microsoft.Practices.SPG.AJAXSupport.dll**. Insert the **Microsoft.Practices.SPG.AjaxSupport.Controls.SafeScriptManager** control into the ASCX file that declares your Web Part.

Using AJAX Capabilities with a Web Part

The following XML shows how to add the safe script manager to a Web Part ASCX page.

XML

```
<%@ Register
    Assembly="Microsoft.Practices.SPG.AJAXSupport, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=8768CCAE1C3C9EB2"
    TagPrefix="spg"
    Namespace="Microsoft.Practices.SPG.AJAXSupport.Controls" %>

<spg:SafeScriptManager ID="SafeScriptManager"
                        runat="server"
/>
```

Usage Notes

There are no usage notes for the safe script manager.

Using the AJAX Update Panel in SharePoint

Typical Goals

You want to use the AJAX **UpdatePanel** control on a SharePoint page or in a Web Part even though, by default, this control does not work with these standard SharePoint components because of differences in the postback mechanisms that are used by SharePoint and ASP.NET.

Note:

The ASP.NET AJAX **UpdatePanel** control must change the page's postback behavior to enable support for asynchronous postbacks and partial rendering. However, in a SharePoint application, these modifications cause a JavaScript error. This is because SharePoint attempts to make a similar change. The **SafeScriptManager** control corrects the JavaScript problem.

Solution

In Visual Studio, add a reference to **Microsoft.Practices.SPG.AJAXSupport.dll**. Add a **SafeScriptManager** control to your page and set the **EnableUpdatePanelSupport** flag to **true**. This changes the postback script that is needed for the **UpdatePanel** control to work in a SharePoint environment. The following code demonstrates how to add a **SafeScriptManager** control to a page.

XML

```
<%@ Register
    Assembly="Microsoft.Practices.SPG.AJAXSupport, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=8768CCAE1C3C9EB2"
    TagPrefix="spg"
    Namespace="Microsoft.Practices.SPG.AJAXSupport.Controls" %>

<spg:SafeScriptManager ID="SafeScriptManager"
    EnableUpdatePanelSupport = "True"
    runat="server"
/>

<asp:UpdatePanel runat="Server" id="MyUpdatePanel">
    ...
</asp:UpdatePanel>
```

Usage Notes

You may place several **SafeScriptManager** controls on your page, and it is allowed to set the **EnableUpdatePanelSupport** flag on each **SafeScriptManager** control on your page. However, when there are multiple **SafeScriptManager** controls on a page, only one of them must set the **EnableUpdatePanelSupport** flag to **true** in order to enable the correct postback behavior for the whole page.

Design

The safe script manager is implemented as an ASP.NET control. Its start up logic checks for the presence of the **ScriptManager** control and adds the control if it is not already present on the page. This is shown in the following code.

C#

```
protected override void CreateChildControls()
{
    this.scriptManager = ScriptManager.GetCurrent(this.Page);
    if (this.ScriptManager == null)
    {
        this.scriptManager = CreateScriptManager();
        this.Controls.Add(this.scriptManager);
    }

    base.CreateChildControls();
}
```

If the **EnableUpdatePanelSupport** property is set to **true**, this control emits some JavaScript code that allows you to use the ASP.NET **UpdatePanel** control in SharePoint.

Development How-to Topics

The [How to: Use the Safe Script Manager to Provide AJAX Support to Web Parts](#) topic describes how to use the SharePoint Guidance Library's safe script manager. The **SafeScriptManager** class is an ASP.NET custom control that loads the .NET Framework's **ScriptManager** control only if it is not already present on the current Web page. For more information, see [The SharePoint Guidance Library](#).

How to: Use the Safe Script Manager to Provide AJAX Support to Web Parts

The following procedure demonstrates how to add the safe script manager. You can either do this in code or include it on a custom master page.

To add the safe script manager

1. In Visual Studio, add a reference to **Microsoft.Practices.SPG.AJAXSupport.dll**.
2. Create an ASP.NET Web Part. Override the **CreateChildControls** method and add an instance of the **SafeScriptManager** class to the Web Part's **Controls** collection. The following code shows how to do this.

C#

```
protected override void CreateChildControls()
{
    base.CreateChildControls();
    if (this.ControlMode == SPControlMode.Display)
    {
        this.Controls.Add(new SafeScriptManager());

        // Add controls that require ScriptManager instance to
        // be on page.
    }
}
```

3. An alternative to step 2 is to insert the **Microsoft.Practices.SPG.AjaxSupport.Controls.SafeScriptManager** control into an ASCX file that is included in your Web Part. The following XML shows how to do this.

XML

```
<%@ Register
    Assembly="Microsoft.Practices.SPG.AJAXSupport, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=8768CCAE1C3C9EB2"
    TagPrefix="spg"
    Namespace="Microsoft.Practices.SPG.AJAXSupport.Controls" %>

<spg:SafeScriptManager ID="SafeScriptManager"
    runat="server"
/>
```

The SharePoint Logger

The SharePoint Guidance Library includes a basic logging and tracing component named the SharePoint logger that is tailored to the SharePoint environment. It is a reusable component that you can include in your own SharePoint applications.

Logging and tracing are different activities that produce output that is intended for two audiences. Logging is directed toward system administrators who typically rely on the Windows event logs to monitor deployed applications. They often use automated tools such as the System Center Operations Manager (SCOM) to monitor the event logs. Administrative centers are also referred to as *operations*.

Tracing is directed toward developers. In particular, SharePoint developers often use the Unified Logging Service (ULS) trace log. This log is used by Windows SharePoint Services itself. Logging your custom application traces to the ULS trace log allows you to view them in the larger context of Windows SharePoint Services operations without having to correlate multiple trace logs.

The **SharePointLogger** class provides a simple interface named **ILogger**. The interface includes methods that log messages based on whether they are intended for system administrators or developers. The output of the **LogToOperations** method is written to the event log, where it is read by system administrators. Messages for system administrators should give a clear indication of the problem and possible action that the administrator can take. These messages are also written to the trace log to help developers who may need to debug the application.

The output of the **TraceToDeveloper** method is only written to the trace log. Messages to the trace log are meant for developers who are familiar with the application's implementation.

To help with debugging, both methods also log additional contextual information if it is available, such as the name of the current user, the URL that was requested, and some other fields.

The **SharePointLogger** class should not be directly instantiated. Instead, you should use the SharePoint service locator to request it. This allows you to plug in different logging implementations, such as the patterns & practices Enterprise Library Logging Application Block. For more information about the SharePoint service locator, see [The SharePoint Service Locator](#).

Note:

The ULS trace logs can be uploaded to Microsoft as part of customer technical service and support. Do not write personal data or proprietary information to the trace log. You should not write sensitive information, such as passwords or confidential data, to any log.

This section includes the following topics.

- [Key Scenarios](#)
- [Design of the SharePoint Logger](#)
- [Development How-to Topics](#)

For general guidance about logging and tracing, see [Providing Application Diagnostics](#). For more information about the ULS trace log, see [Trace Logs](#) on MSDN.

Key Scenarios

The following sections discuss the scenarios for the SharePoint logger:

- [Logging an Event to Operations](#). This scenario is applicable to deployed applications and applications that are in development.
- [Logging an Exception to Operations](#). This scenario is applicable to deployed applications and applications that are in development.
- [Creating a Trace Message for Developers](#). This scenario is applicable during development. It does not apply to deployed applications.
- [Customizing the Logger for Unit Testing](#). This scenario is applicable to unit testing.
- [Customizing the Logger in an Application](#). This scenario shows how to configure your application to use custom logging.
- [Selecting a Custom Event Source Name](#). This scenario allows you to customize the event source name in the ULS trace log.

Logging an Event to Operations

Typical Goals

In this scenario, you want to record an event to both the Windows event log and the ULS trace log. The record of the event is meant to be used by system administrators.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. Use the SharePoint service locator to get a reference to a **Microsoft.Practices.SPG.Common.Logging.ILogger** interface and invoke the **LogToOperations** method. Pass an error string and an optional integer event ID to the **LogToOperations** method.

Logging an Event

The following code demonstrates how to log an event.

```
C#
ILogger logger = SharePointServiceLocator.Current.GetInstance<ILogger>();
logger.LogToOperations("The current user doesn't have a PartnerID specified.",
    (int) EventLogEventId.PartnerNotFound);
```

The **LogToOperations** method writes to both the Windows event log and to the ULS trace log. The **EventLogEventId** enumeration is an example of a user-defined set of event IDs. It is a recommended practice to use enumerated values or constants for your application's event IDs.

Usage Notes

You should use the **LogToOperations** method to log information that is meaningful to system administrators. If possible, messages should indicate an action that the administrator can take. For example, compare the message "A timeout occurred" to "Could not connect to database X, because the connection timed out". The first message does not provide any information to system administrators that they can respond to. The second message gives them an idea of where to look for the problem.

If the event is caused by an exception, consider using the overloaded version of the **LogToOperations** method that takes an exception as an argument. This method formats and logs the exception details.

The **LogToOperations** method has overloaded versions that allow you to specify optional information for the event you are recording. You can include the following:

- An event-log entry type that distinguishes warnings from errors
- A string that denotes a category

The following are examples of event logging that use the different overloads of the **LogToOperations** method.

```
C#
// ILogger logger = ...
// string msg = ...
// string category = ...

// Log an event.
logger.LogToOperations(msg, (int) EventLogEventId.SkuNotFound);

// Log an event without specifying an event ID.
logger.LogToOperations(msg);

// Log an event giving an event-log entry type (without specifying an event ID).
logger.LogToOperations(msg, EventLogEntryType.Error);

// Log an event with an event-log entry type.
logger.LogToOperations(msg, (int) EventLogEventId.PartnerNotFound,
    EventLogEntryType.Error);

// Log an event with an event-log entry type and category.
logger.LogToOperations(msg, (int) EventLogEventId.PartnerNotFound,
    EventLogEntryType.Error, category);
```

You can also log exception information with your event. To do this, see [Logging an Exception to Operations](#).

For more information about the **EventLogEntryType** enumeration, see [EventLogEntryType Enumeration](#) on MSDN.

Logging an Exception to Operations

Typical Goals

In this scenario, you want to log an exception to the Windows event log and the ULS trace log. Typically, this is an unhandled exception that the system administrator needs to know about.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. Use the SharePoint service locator to get a reference to a **Microsoft.Practices.SPG.Common.Logging.ILogger** interface and invoke the **LogToOperations** method. Pass the exception that you want to log as an argument to the **LogToOperations** method.

Logging an Exception

The following code demonstrates how to log an exception.

```
C#
try
{
    /// ... a SharePoint operation ...
}
catch (SPException sharePointException)
{
    ILogger logger = SharePointServiceLocator.Current.GetInstance<ILogger>();
    logger.LogToOperations(sharePointException);
}
```

The **LogToOperations** method records the exception in the Windows event log and the ULS trace log.

Usage Notes

Typically, logging exceptions to operations is done for unhandled exceptions. Although the exception message rarely provides a clear course of action, it is valuable for a system administrator to know that there are problems with a certain component and that a developer might have to look at it. Be selective about the exceptions that are logged.

You can provide additional text when logging exceptions. Try to provide information that can help identify what happened when the exception occurred. For example, the exception "An unknown exception occurred while trying to retrieve product information from the product service. The exception message was: A timeout occurred" is much more helpful to operations than "A timeout occurred".

The **LogToOperations** method has overloaded versions that allow you to specify optional information for the exception you are logging. You can include the following:

- An application-specific error string of your choice
- An integer event ID
- An event-log entry type that distinguishes warnings from errors
- A string that denotes a category

The following are examples of exception logging that use the different overloads of the **LogToOperations** method.

```
C#
// ILogger logger = ...
// Exception ex = ...
// string msg = ...
// string category = ...

// Log an exception with an additional error message.
logger.LogToOperations(ex, msg);

// Log an exception with an additional error message and
// an application-defined event ID.
logger.LogToOperations(ex, msg, (int) EventLogEventId.SkuNotFound);

// Log an exception with an additional error message, a default event ID,
// an event-log entry type, and a category string.
logger.LogToOperations(ex, msg, 0, EventLogEntryType.Error, category);

// Log an exception with a default event ID, an event-log entry type, and
// a category string.
logger.LogToOperations(ex, 0, EventLogEntryType.Warning, category);
```

For more information about the **EventLogEntryType** enumeration, see [EntryLogEntryType Enumeration](#) on MSDN.

Creating a Trace Message for Developers

Typical Goals

In this scenario, you want to write debugging information to the ULS trace log.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. Use the SharePoint service locator to get a reference to a **Microsoft.Practices.SPG.Common.Logging.ILogger** interface and invoke the **TraceToDeveloper** method. You can include descriptive strings, event IDs, exceptions, trace severity levels, and category strings as arguments to the **TraceToDeveloper** method.

Creating a Trace Message During Development

The following code demonstrates how to create a trace message.

```
C#
ILogger logger = SharePointServiceLocator.Current.GetInstance<ILogger>();
logger.TraceToDeveloper("Unexpected condition");
```

The **TraceToDeveloper** method writes to the ULS trace log.

Usage Notes

The **TraceToDeveloper** method has 11 overloaded versions that allow you to specify optional information such as trace severity level, event ID, exception data, and category string. The following are some examples of how to use the overloads.

```
C#
// ILogger logger = ...
// Exception ex = ...
// string msg = ...
// string category = ...

// Here is a simple trace.
logger.TraceToDeveloper(msg);

// trace an event
logger.TraceToDeveloper(msg, (int) EventLogEventId.SkuNotFound);

// trace with a trace severity
logger.TraceToDeveloper(msg, TraceSeverity.High);

// trace with a category string
logger.TraceToDeveloper(msg, category);

// trace with an event ID and a trace severity level
logger.TraceToDeveloper(msg, (int) EventLogEventId.PartnerNotFound,
    TraceSeverity.Verbose);

// trace with an event ID, a trace severity level and a category string
logger.TraceToDeveloper(msg, (int) EventLogEventId.PartnerNotFound,
    TraceSeverity.Unexpected, category);

// trace an exception
logger.TraceToDeveloper(ex);

// trace an exception with an additional error message
logger.TraceToDeveloper(ex, msg);

// trace an exception with an additional error message and
// an application-defined event id
logger.TraceToDeveloper(ex, msg, (int) EventLogEventId.SkuNotFound);

// trace an exception with an additional error message, a default event id,
// a trace severity level and a category string
logger.TraceToDeveloper(ex, msg, 0, TraceSeverity.High, category);

// trace an exception with a default event id, trace severity string and
// a category string
logger.TraceToDeveloper(ex, 0, TraceSeverity.Verbose, category);
```

The **EventLogEventId** enumeration is user defined. It is a recommended practice to encode your application's event types as enumerated values.

The **TraceSeverity** enumeration is provided by SharePoint. It is found in the **Microsoft.SharePoint.Administration** namespace. For more information, see [TraceSeverity Enumeration](#) on MSDN.

Customizing the Logger for Unit Testing

Typical Goals

In this scenario, you want to bypass the default logging and tracing behavior of the SharePoint logger and write all logging and tracing information to a mock logger that is specified by your unit test.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. Create a mock implementation of the **ILogger** interface. This mock logger can write to the standard output or to a test-provided file stream, or simply set a public property that shows that the logger was invoked as expected. When you initialize your unit test, replace the current SharePoint service locator with a test-provided service locator that includes a type mapping to your mock logger. Run your unit test. For information about replacing the service locator, see [The SharePoint Service Locator](#).

Customizing the Logger for Unit Testing

```
C#
public class MyFixture
{
    private MockLogger logger;

    [TestInitialize]
    public void TestInitialize()
    {
        ActivatingServiceLocator locator = new ActivatingServiceLocator();

        locator.RegisterTypeMapping<ILogger, MockLogger>
            (InstantiationType.AsSingleton);

        SharePointServiceLocator.ReplaceCurrentServiceLocator(locator);

        this.logger =
            SharePointServiceLocator.Current.GetInstance<ILogger>()

                                   as MockLogger;
    }

    // ...
}
```

In this test initialization method, the current service locator instance is configured so that an instance of the **MockLogger** class is returned when the SharePoint service locator asks for an **ILogger** object. After this class is registered with the service locator, the logging output for any objects that use **ILogger** are directed to the test-provided logging implementation. Notice that you did not need to alter any code in the component that is being tested.

Usage Notes

To implement this scenario, you must provide a mock implementation of **ILogger**. The following code shows an example implementation.

```
C#
public class MockLogger : ILogger
{
    // Define a public property for the message.
    // The unit tests can use this to validate that the method
    // was called.
    public string LogToOperationsCalledWithMessage {get; set;};

    public void LogToOperations (string message)
    {
        LogToOperationsCalledWithMessage = message;  }
}
```

Customizing the Logger in an Application

Typical Goals

In this scenario, you want to configure your application to use a custom logging component. This scenario applies to deployed applications.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**, and to **Microsoft.Practices.ServiceLocation.dll**. Create a new class that implements the **IEventLogLogger** or **ITraceLogger** interface. For example, your new class might encapsulate an existing logging mechanism that you want to integrate into your application. Use the SharePoint service locator to get a reference to an **IServiceLocatorConfig** interface. Use the **RegisterTypeMapping** method of the **IServiceLocatorConfig** object to configure the service locator with type mappings to your new logging classes. For information about adding the service locator, see [Getting Services from the SharePoint Service Locator](#).

Customizing the Logger in an Application

The following code demonstrates how to customize the logger.

```
C#
using Microsoft.Practices.ServiceLocation;
using Microsoft.Practices.SPG.Common.ServiceLocation;
using Microsoft.Practices.SPG.Common.Logging;

[CLSCompliant(false)]
[Guid("8b0f085e-72a0-4d9f-ac74-0038dc0f6dd5")]
public class MyFeatureReceiver : SPFeatureReceiver
{
    /// ...

    [SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
    public override void FeatureInstalled(SPFeatureReceiverProperties properties)
    {
        IServiceLocator serviceLocator = SharePointServiceLocator.Current;
        IServiceLocatorConfig typeMappings =
            serviceLocator.GetInstance<IServiceLocatorConfig>();

        typeMappings.RegisterTypeMapping<IEventLogLogger, MyEventLogLogger>();
        typeMappings.RegisterTypeMapping<ITraceLogger, MyTraceLogger>();
    }
}
```

This code establishes service locator type mappings for the **IEventLogLogger** interface and the **ITraceLogger** interface. The implementation class **MyEventLogLogger** is mapped to the **IEventLogLogger** interface. The implementation class **MyTraceLogger** is mapped to the **ITraceLogger** interface. These mappings mean that the output for any objects that use the SharePoint logger use these logging implementations. Notice that you did not need to alter any code in the component that is being tested.

Usage Notes

To implement this scenario, you must provide event log logger and trace logger implementations. The following code shows examples of these implementations.

```
C#
public class MyTraceLogger : ITraceLogger
{
    [SharePointPermissionAttribute(SecurityAction.InheritanceDemand,
                                   ObjectModel = true)]
    [SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
    public void Trace(string message, int eventId, TraceSeverity severity,
                     string category)
    {
        /// ... custom code to handle tracing request ...
    }
}

public class MyEventLogLogger : IEventLogLogger
{
    [SharePointPermissionAttribute(SecurityAction.InheritanceDemand,
                                   ObjectModel = true)]
    [SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
    public void Log(string message, int eventId, EventLogEntryType severity,
                   string category)
    {
        /// ... custom code to handle event logging request ...
    }
}
```


Selecting a Custom Event Source Name

Typical Goals

By default, the SharePoint logger logs messages to the ULS trace log that is under the Office SharePoint Server event source. You can change this by selecting a custom event source name.

Solution

In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. Use the SharePoint service locator to get a reference to the SharePoint configuration manager using the **IConfigManager** interface. Invoke the **SetInPropertyBag** method and pass the custom event source name that you want to use for your ULS trace log as an argument. Use the library-provided constant **Microsoft.Practices.SPG.Common.Constants.EventSourceNameConfigKey** as the configuration key argument.

Selecting a Custom Event Source Name

The following code demonstrates how to select a custom event source name for the ULS trace log.

```
C#
string eventSourceKey =
    Microsoft.Practices.SPG.Common.Constants.EventSourceNameConfigKey;

IConfigManager configManager =

SharePointServiceLocator.Current.GetInstance<IConfigManager>();

configManager.SetInPropertyBag(eventSourceKey,
    "My Custom Event Source",
    SPFarm.Local);
```

The invocation of the **SetInPropertyBag** method sets a property of the current farm to the custom event source named "My Custom Event Source". The SharePoint logger will use this event source when logging to the ULS trace log.

The **eventSourceKey** variable has the value **Microsoft.Practices.SPG.Common.EventSourceName**.

Usage Notes

The event source used in combination with the event ID is often used by system administrators to identify an event that has occurred in the system.

It is recommended that you use your own event source name. The SharePoint logger will write an additional warning message if the default event source is used for trace messages.

Design of the SharePoint Logger

The ILogger Interface

The **ILogger** interface is designed to accommodate most logging scenarios and to help developers target their messages to either system administrators or developers.

The implementation of the **ILogger** interface should be retrieved through the SharePoint service locator. The default implementation is the **SharePointLogger** class, but it is possible to register different logging implementations. For example, you can register an implementation that is based on the Microsoft patterns & practices Enterprise Library.

The SharePointLogger Class

The **SharePointLogger** class is a straightforward implementation of the **ILogger** interface. The class is composed of two replaceable components that implement the **IEventLogLogger** interface and the **ITraceLogger** interface. The interfaces do the following:

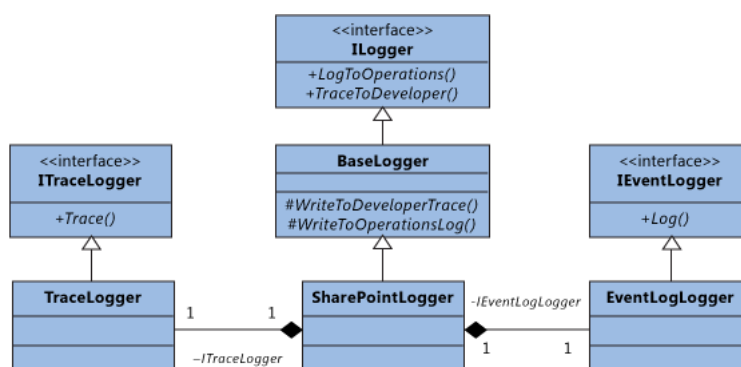
- The **IEventLogLogger** interface records events that are meant for system administrators. By default, this interface is implemented by the **EventLogLogger** class. This class writes log messages to the Windows event log.
- The **ITraceLogger** interface records the trace information that is used by application developers. By default, this interface is implemented by the **TraceLogger** class. This class writes trace messages to the SharePoint Unified Logging System (ULS). If you prefer to write to a different trace log, such as the ASP.NET trace log, you can create and register a custom **ITraceLogger** interface.

The **SharePointLogger** class has the following responsibilities:

- **Forward the log messages to the appropriate logger.** Messages targeted at operations are sent to both the class that implements the **IEventLogLogger** interface and the class that implements the **ITraceLogger** interface. Messages targeted solely at developers are only sent to the class that implements the **ITraceLogger**.
- **Enrich the log message with contextual information.** Messages with contextual information, such as the current URL and the name of the currently logged on user, save troubleshooting time.
- **Format exceptions into a human readable message.** Messages in the log files must be that are readable by humans.
- **Provide a high level of robustness in case the logging fails.** If a message cannot be written to the event logger implementation, a **LoggingException** is thrown that contains both the original log message and the reason for the logging failure. No exception is thrown if a message cannot be written to the trace logger implementation. Instead, the **SharePointLogger** class will attempt to write a message to the event logger implementation to indicate that the trace has failed.

The following diagram shows the class hierarchy of the SharePoint logger implementation.

Class hierarchy of the SharePoint logger implementation



Note:

In the diagram, the arrows link classes to the interfaces that they implement. The diamonds show containment of components. The design choice of containing functionality instead of inheriting it through the type hierarchy is referred to as *composition*. Composition produces flexible designs and increases the opportunity for component reuse.

The event log and trace logger implementations perform the details of writing administrative and diagnostic messages.

If you need additional functionality or flexibility, you can extend the **SharePointLogger** by inheriting from it or from the **BaseLogger** class, by creating a custom **ILogger** implementation or by building on another logging framework, such as the Microsoft patterns & practices Enterprise Library Logging Application Block.

The EventLogLogger Class

The **EventLogLogger** class writes entries to the Windows event log. This class is configured as the default implementation of the **IEventLogLogger** interface that is used by the SharePoint logger.

Because the user account may not have sufficient privileges to write to the global event log, the **Log** method of the **EventLogLogger** class executes with elevated privileges. This means that it uses the configured application pool service account privileges.

The **EventLogLogger** class uses the value **Office SharePoint Server** as the default ULS trace log event source name. This event source name is already used by the Office SharePoint Server. You should configure a custom event source name before issuing trace messages. If you do not set a custom event source name, a warning will also be logged that a custom event source name should be configured.

You can configure your application to use a custom event source name by using the **IConfigManager** interface. With the **IConfigManager** interface you can set the value of the **Microsoft.Practices.SPG.Common.EventSourceName** in the current farm's property bag to the event source you want to use. For more information on how to set a property using the **IConfigManager** interface, see [Adding and Updating Configuration Settings](#).

Note:

It is recommended that you set the **Microsoft.Practices.SPG.Common.EventSourceName** property of the current **SPFarm** object. This allows the logger to find the property in all SharePoint contexts, including the case when feature receivers are called during a command-line installation operation. In the command-line installation, there is no current site, site collection or Web application.

The TraceLogger Class

The **TraceLogger** class writes tracing data to the ULS trace log. This class is configured as the default implementation of the **ITraceLogger** interface that is used by the SharePoint logger.

The **TraceLogger** class uses an auxiliary static class that is named **UlsTraceProvider**. This class encapsulates the details of writing to the ULS trace log. The **TraceLogger** class is based on an implementation that is discussed in [Trace Log Example](#) on MSDN.

The SharePoint logger uses the ULS trace log because it allows developers to see their own diagnostic messages in the context of the trace messages that are generated by SharePoint. The ULS logs to a file in a structured format. It is possible to open this file in Notepad; however, ULS log files are usually large. It is often easier to use a ULS Viewer to read these log files. For example, the [ULS Log Viewer](#) on CodePlex is a community driven effort to create an ULS Log Viewer for SharePoint.

For more information about the ULS trace log, see [Trace Logs](#) on MSDN.

Customizing the SharePoint Logger

The SharePoint logger implementation uses the SharePoint service locator to access the **EventLogLogger** class and the **TraceLogger** class. With service location, you can replace the default logging and tracing components if you need more flexibility or want to reuse a logging component that you already have. For example, during unit testing you can customize logging so that the logging information becomes part of the unit test's output.

For more information about the SharePoint service locator, see [The SharePoint Service Locator](#).

Service location occurs at three points of the logger's design.

- The **IEventLogLogger** interface is mapped by default to the **EventLogLogger** class.
- The **ITraceLogger** interface is mapped by default to the **TraceLogger** class.
- The **ILogger** interface is mapped by default to the **SharePointLogger** class.

You can customize logging and tracing by reconfiguring any of these type mappings.

- If you want to change the way that event logging is handled, replace the **EventLogLogger** class with a new class that implements the **IEventLogLogger** interface.
- If you want to change the way that trace logging is handled, replace the **TraceLogger** class with a new class that implements the **ITraceLogger** interface. For example, you can log to the ASP.Net trace log.
- If you want to slightly modify the way the **SharePointLogger** class handles logging, you can create a class that derives from it. It has virtual methods for most operations, so you can override most of its behavior.
- If you want full control over how logging and tracing are handled, replace the **SharePointLogger** class with a new class that derives from the **BaseLogger** class or that directly implements the **ILogger** interface.

Use the **IServiceLocatorConfig** interface to configure your application to use your new custom logger. For an example of how to replace a service using configuration, see [Getting Services from the SharePoint Service Locator](#).

Note:

Callers should invoke the logging interface types (such as **ILogger**) and not the logging implementation types (such as the **SharePointLogger** class) directly.

To customize the event source name for the SharePoint logger, see the discussion of the [EventLogLogger](#) class earlier in this topic.

Development How-to Topics

This topic describes how to use the SharePoint Guidance Library logging and tracing component. It includes the following procedures:

- [How to: Log an Event for Operations](#). This topic describes how to log messages that are meant for system administrators.
- [How to: Log an Exception to Operations](#). This topic describes how to log unhandled exceptions.
- [How to: Create a Trace Message During Development](#). This topic describes how to write debugging information to the Unified Logging Service (ULS) trace log.
- [How to: Customize the Logger for Unit Testing](#). This topic describes how to create a mock logger for unit testing.
- [How to: Customize the Logger in an Application](#). This topic describes how to use a custom logger.
- [How to: Select a Custom Event Source Name](#). This topic describes how to use an event source other than the default.

For more information about the SharePoint Guidance Library, see [The SharePoint Guidance Library](#).

How to: Log an Event for Operations

The following procedure demonstrates how to log messages that are meant for system administrators to the Windows event log and the ULS trace log.

To log an event for operations

1. In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow, or anything else that needs to be in the global assembly cache, the **Microsoft.Practices.SPG.Common** assembly also needs to be in the global assembly cache.
2. Create an instance of **SharePointLogger** and invoke the **LogToOperations** method. Pass an error string and an optional integer event ID to the **LogToOperations** method. The following code shows how to do this.

C#

```
ILogger logger = new SharePointLogger();  
logger.LogToOperations("The current user doesn't have a PartnerID specified.",  
    (int) EventLogEventId.PartnerNotFound);
```


How to: Log an Exception to Operations

The following procedure demonstrates how to log an exception to the Windows event log and the ULS trace log. Typically, this is an unhandled exception that the system administrator needs to know about.

To log an exception

1. In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow, or anything else that needs to be in the global assembly cache, the **Microsoft.Practices.SPG.Common** assembly also needs to be in the global assembly cache.
2. Create an instance of **SharePointLogger** and invoke the **LogToOperations** method. Pass the exception that you want to log as an argument to the **LogToOperations** method. The following code shows how to do this.

C#

```
try
{
    /// ... a SharePoint operation ...
}
catch (SPException sharePointException)
{
    ILogger logger = new SharePointLogger();
    logger.LogToOperations(sharePointException);
}
```

How to: Create a Trace Message During Development

The following procedure demonstrates how to write debugging information to the ULS trace log.

To write debugging information

1. In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow, or anything else that needs to be in the global assembly cache, the **Microsoft.Practices.SPG.Common** assembly also needs to be in the global assembly cache.
2. Create an instance of **SharePointLogger** and invoke the **TraceToDeveloper** method. You can include descriptive strings, event IDs, exceptions, trace severity levels, and category strings as arguments to the **TraceToDeveloper** method. The following code shows how to do this.

C#

```
ILogger logger = new SharePointLogger();  
logger.TraceToDeveloper("Unexpected condition");
```

How to: Customize the Logger for Unit Testing

The following procedure demonstrates how to bypass the default logging and tracing behavior of the SharePoint logger and write all logging and tracing information to a mock logger. This is typically done for unit testing.

To create a mock logger

1. In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow, or anything else that needs to be in the global assembly cache, the **Microsoft.Practices.SPG.Common** assembly also needs to be in the global assembly cache.
2. Create a mock implementation of the **ILogger** interface. This mock logger can write to the standard output, to a test-provided file stream, or simply set a public property.
3. When you initialize your unit test, replace the current SharePoint service locator with a test-provided service locator that includes type mappings to your mock logger. (For information about replacing the service locator, see [The SharePoint Service Locator](#).) The following code shows how to do this.

```
C#
public class MyFixture
{
    private MockLogger mockLogger;

    [TestInitialize]
    public void TestInitialize()
    {
        ActivatingServiceLocator locator = new ActivatingServiceLocator();

        locator.RegisterTypeMapping<ILogger, MockLogger>
            (InstantiationType.AsSingleton);

        SharePointServiceLocator.ReplaceCurrentServiceLocator(locator);

        this.mockLogger =
            SharePointServiceLocator.Current.GetInstance<ILogger>()

                                   as MockLogger;
    }

    // ...
}
```

4. Run the unit test and validate that the business logic interacts with the mock logger as expected. The following code shows how to do this.

```
C#
[TestMethod]
public void OrderProcessorLogs()
{
    // Arrange
    this.mockLogger.LogToOperationsCalledWithMessage = null;
    Order testOrder = new Order("TestValue");

    // Act
    OrderProcessor target = new OrderProcessor();
    target.ProcessOrder(testOrder);

    // Assert
    Assert.AreEqual("Processing Order: TestValue",
        this.mockLogger.LogToOperationsCalledWithMessage);
}
```

How to: Customize the Logger in an Application

The **SharePointLogger** class internally uses two separate sinks. The **IEventLogLogger** interface, which is implemented by the **EventLogLogger** class, is intended for system administrators. The **ITraceLogger** interface, which is implemented by the **TraceLogger** class, is intended for developers. You can replace either a sink or the **SharePointLogger** class with another implementation of the **ILogger** class. The following procedure demonstrates how to replace both sinks with custom implementations.

To use a custom implementation

1. In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**, and to **Microsoft.Practices.ServiceLocation.dll**. If you are writing a feature receiver, item event receiver, workflow, or anything else that needs to be in the global assembly cache, the **Microsoft.Practices.SPG.Common** and **Microsoft.Practices.ServiceLocation** assemblies also need to be in the global assembly cache.
2. Create new implementations of **IEventLogLogger** and **ITraceLogger** interfaces. For example, your new classes might encapsulate an existing logging mechanism that you want to integrate into your application. The following code shows how to do this.

C#

```
public class MyTraceLogger : ITraceLogger
{
    [SharePointPermissionAttribute(SecurityAction.InheritanceDemand,
                                   ObjectModel = true)]
    [SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
    public void Trace(string message, int eventId, TraceSeverity severity,
                     string category)
    {
        /// ... custom code to handle tracing request ...
    }
}

public class MyEventLogLogger : IEventLogLogger
{
    [SharePointPermissionAttribute(SecurityAction.InheritanceDemand,
                                   ObjectModel = true)]
    [SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
    public void Log(string message, int eventId, EventLogEntryType severity,
                   string category)
    {
        /// ... custom code to handle event logging request ...
    }
}
```

3. Create a farm-scoped feature receiver class. This is where you will register the new implementations with the service locator. Create an instance of the **ServiceLocatorConfig** class. Use the **RegisterTypeMapping** method to configure the service locator with type mappings to the new logging classes. For information about adding services to the service locator, see [Registering a Service with the SharePoint Service Locator](#).

C#

```
using Microsoft.Practices.ServiceLocation;
using Microsoft.Practices.SPG.Common.ServiceLocation;
using Microsoft.Practices.SPG.Common.Logging;

[CLSCompliant(false)]
[Guid("<Your Farm Scoped Feature Guid>")]
public class MyFarmScopedFeatureReceiver : SPFeatureReceiver
{
    /// ...

    [SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
    public override void FeatureActivated(SPFeatureReceiverProperties properties)
    {
        IServiceLocatorConfig serviceLocatorConfig =
            new ServiceLocatorConfig();

        serviceLocatorConfig.RegisterTypeMapping<IEventLogLogger, MyEventLogLogger>();
        serviceLocatorConfig.RegisterTypeMapping<ITraceLogger, MyTraceLogger>();
    }
}
```

How to: Select a Custom Event Source Name

By default, the SharePoint logger logs messages to the ULS trace log that is under the Office SharePoint Server event source. The following procedure shows how to select a custom event source name.

To select a custom event source name

1. In Visual Studio, add a reference to the SharePoint Guidance Library, **Microsoft.Practices.SPG.Common.dll**. If you are writing a feature receiver, item event receiver, workflow, or anything else that needs to be in the global assembly cache, the **Microsoft.Practices.SPG.Common** assembly also needs to be in the global assembly cache.
2. Use the **IConfigManager** interface to create an instance of the **HierarchicalConfig** class.
3. Invoke the **SetInPropertyBag** method and pass in the custom event source name that you want to use for your ULS trace log as an argument. Use the library-provided constant **Microsoft.Practices.SPG.Common.Constants.EventSourceNameConfigKey** as the configuration key argument. The following code shows how to perform steps 2 and 3.

```
C#
string eventSourceKey =
    Microsoft.Practices.SPG.Common.Constants.EventSourceNameConfigKey;

IConfigManager configManager = new HierarchicalConfig();

configManager.SetInPropertyBag(eventSourceKey,
    "My Custom Event Source",
    SPFarm.Local);
```

The **SetInPropertyBag** method sets a property of the current farm to the custom event source that is named My Custom Event Source. The SharePoint logger will use this event source when logging to the ULS trace log.

Integrating Line-of-Business Systems

SharePoint applications allow users to access information that is stored on back-end, line-of-business (LOB) systems. SharePoint applications can integrate data from different systems and present a unified view of it to end users.

This section discusses how to integrate LOB systems with SharePoint. It contains the following subsections:

- [Consuming Services in SharePoint](#)
- [Exposing Services from SharePoint](#)
- [Security Considerations for LOB Integration](#)

Consuming Services in SharePoint

A SharePoint application can use a service layer or another external component when it retrieves data from an external system. The following topics discuss two ways to use services to connect your application to LOB system:

- [Consuming Web Services with the Business Data Catalog \(BDC\)](#)
- [Using WCF Services](#)

Consuming Web Services with the Business Data Catalog (BDC)

Line-of-business (LOB) systems often contain an organization's most valuable data, but accessing that data can require specialized training. This can make the data accessible to only a few knowledgeable users. One way to make the data more accessible is with the SharePoint platform. You can create SharePoint applications that collect data from LOB systems, merge it, and present it to users as a single view. Users can access this data with SharePoint's familiar user interface (UI). The Partner Portal is an example of such an application. The Business Data Catalog (BDC), which is included in Microsoft Office SharePoint 2007, is a convenient way to integrate data from back-end, LOB server applications. The Partner Portal application shows how to use the BDC with Web services to access LOB data.

Note:

The Business Data Catalog can connect to databases and to Web services. The Partner Portal application does not include any examples of this; however, using the BDC with a database is similar in many ways to using it with a Web service. For examples and references for how to use the BDC to expose data from a database, see [Modeling Database Systems](#) on MSDN.

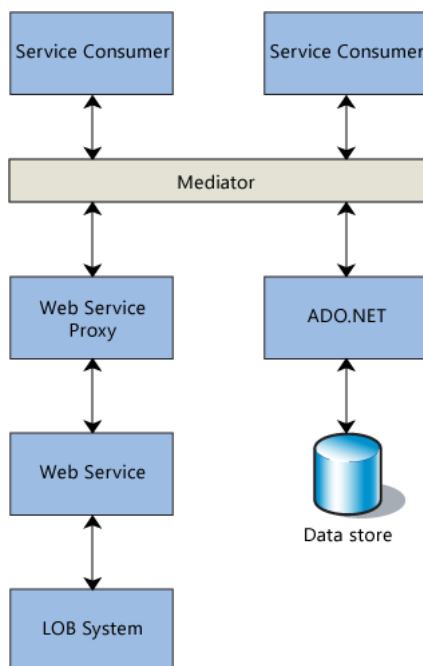
BDC Mediated Integration

The BDC mediates between services and consumers. Instead of directly accessing the LOB systems, a SharePoint application uses the BDC to mediate interactions between the systems and the application's Web Parts.

The BDC integrates business data that is available either through Web services or from databases. A feature of the BDC is that you can construct read-only operations with XML. You do not need to write any code. If you need write operations, you can either use custom-coded Web Parts to access the BDC programmatically or InfoPath Forms Services. For an example of a custom Web part that uses the BDC, see [Product Catalog](#). For more information about integrating the BDC with InfoPath Forms Services, see [Editing Business Data Using Business Data Catalog Actions and InfoPath Forms Services](#) on MSDN.

The following illustration shows how the Partner Portal uses the mediated integration pattern.

Partner Portal's use of the mediated integration pattern



Because they use the BDC as a mediator, the Web Parts can communicate with the LOB systems without requiring any information about a specific service implementation.

Understanding the BDC Metadata Model

To mediate between the LOB systems and the SharePoint Web Parts, the BDC uses a metadata model that specifies the systems, entities, and operations of the LOB systems that will be exposed through SharePoint. These elements are defined in an application definition file. This file must follow the format of the BdcMetadata.xsd schema file. It is located in the *install\directory:\Program Files\ Microsoft Office Servers\12.0\Bin* folder.

To create an application definition file, you should understand the following metadata objects:

- **LOBSystem**
- **LOBSystemInstance**

- **Entity**
- **Method**
- **Association**
- **Action**

For detailed information about these and other metadata objects, see [Business Data Catalog: Metadata Model](#) on MSDN.

The Finder and SpecificFinder Methods

Two methods with special significance are the **Finder** and the **SpecificFinder** methods. The **Finder** method returns all instances that represent the business entity and those instances that meet the criteria specified in the filters. The **SpecificFinder** method is invoked whenever there is a key-based lookup. This means it returns only one instance or row of data. The methods in an entity can be marked as **Finder** or **SpecificFinder**. There can be only a single **Finder** method and **SpecificFinder** method for each entity, and a single method can act as both a **Finder** and a **SpecificFinder** method.

Using the BDC Definition Editor

Defining the metadata model in the application definition file can be time consuming and error prone. The Business Data Catalog Definition Editor, which is provided in the [SharePoint Server 2007 SDK](#), helps you to write this file. It uses a graphical user interface and automatically creates entities and methods from the database metadata and WSDLs. For a demonstration of how to use the editor, see [Creating a Web Service Connection by Using the Business Data Catalog Definition Editor](#).

Use the BDC Definition Editor to create the initial application definition file. Typically, you must then manually edit the file to customize it for your particular scenarios. For information about how to manually write BDC metadata, see [Authoring Metadata](#) on MSDN. Some of the situations that require you to manually edit the application definition file include the following:

- You want to change the **LOBSYSTEMINSTANCE** object properties to provide different connection strings between the development, User Acceptance Test, and production environments.
- You want to create user-friendly names for the **LOBSYSTEM**, **LOBSYSTEMINSTANCE**, and **ENTITY** objects to help end users understand what these objects represent when they see them on the Central Administration page or in the BDC Web Parts.
- You want to make the queries that are defined for the **Finder** and **SpecificFinder** methods more precise.

There are also commercially available tools, such as [BDC Meta Man](#), that eliminate the need to manually edit the files and also allow you to reopen the files in the editor if you have manually edited them.

After you define the services, you can use standard BDC Web Parts to access them without writing any additional code. You can also access the data programmatically.

BDC Security Options

The BDC provides different security options to use when you access back-end systems. These options include the following:

- **Trusted subsystem (RevertToSelf)**. The BDC runtime uses its own service account credentials instead of the current user's credentials to access the back-end service.
- **Impersonation and delegation (WindowsCredentials)**. The Web service uses the Windows identity of the user to access the back-end service.
- **Single sign-on (Credentials)**. The BDC runtime uses the single sign-on infrastructure to map the user's identity to other credentials to access the back-end service.

For more information about authentication with the BDC, see [Business Data Catalog Authentication](#) on MSDN. For more information about the trusted subsystem approach, see [Trusted Subsystem](#) in the Windows Communication Foundation (WCF) documentation and the white paper, [Trusted Subsystem Design](#), on MSDN. For more information about single sign-on, see [Single Sign-on](#) on MSDN. You should consider using the WCF if you require more sophisticated security mechanisms. For more information about integration with the WCF service, see [Using WCF Services](#).

Caching

The Business Data Catalog (BDC) caches the metadata information it uses such as service definitions, operations, and associations. However, in the case of the retrieved back-end data, the BDC Web Parts and the underlying BDC data infrastructure do not provide a built-in caching mechanism. Calls are made to the back-end data source each time the data is accessed. The frequency of these calls may negatively affect an application's overall performance. The only way to provide caching and to use the BDC is to either to implement your own Web Parts or to code your business components. You can do this for frequently accessed or computationally intensive views while relying on the standard Web Parts for the rest of the user interface.

The Partner Portal application uses custom Web Parts in the product catalog to cache navigation, product, and category information. The site navigation menu uses the **BusinessDataCatalogSiteMapProvider** Web Part. The product details page uses the **ProductDetails** Web Part and the **RelatedParts** Web Part.

Cached information is kept in an instance of the ASP.NET [System.Web.Caching.Cache](#) class. This is a local in-memory cache, which could be different in different in a Web front end server. Using an in-memory cache is an

acceptable technique for information that changes infrequently or at predictable intervals. There is a risk that a user will see outdated information if the back-end data changes before the cached data expires on a server within a farm.

The Contoso.LOB.Services.Client\Repositories\CachedBdcProductCatalogRepository.cs file demonstrates the implementation of the caching mechanism. For more information, see [Caching in SharePoint](#).

How to: Expose BDC Profile Pages

After the Business Data Catalog (BDC) imports an application definition, the BDC automatically generates a *profile page* for each entity that has a [SpecificFinder](#) method. A profile page can display all the information from a record in the BDC. For example, a profile page can display all the fields in a record for a specific customer. The default profile page contains a BDC Item Web Part for that entity. Users can navigate to the profile page of an entity through a View Profile action. Clicking the **View Profile** link in a Business Data column or in a Business Data Web Part lets the user see the details of an item on that item's profile page.

By default, the profile page is hosted on the Shared Services Administration site collection where the BDC administration site is located. Developers can create their own profile pages and host them in any site that is not part of the Shared Services Provider site collection. One reason to create your own profile pages is for security. You may not want to give end users access to the Shared Services Administration site collection. You can direct users to custom profile pages by updating the View Profile action.

To update the "View Profile" action created during the BDC application definition import process

1. Open the SharePoint Central Administration site.
2. In the Quick Launch, click **Shared Services Administration**.
3. Do one of the following:
 - If you installed the Partner Portal application by running the batch file ContosoSetup.bat, click **ContosoSSP**.
 - If the preceding does not apply to you, click **SharedServices1**.
4. Under **Business Data Catalog**, click **View Entities**.
5. Click the entity that you want to update. If you installed the Partner Portal application, you should see three entities: **Category**, **Part**, and **Product**. In this case, click **Category**.
6. Click the **View Profile** action.
7. Update the URL. For the **Category** entity, the URL is already updated by the installation process, so you do not need to update it.
8. If your URL contains query string parameters, use the URL parameters section to specify the business entity property to pass as a value to the query string parameters. For example, if your URL is `"/sites/productcatalog/Category.aspx?CategoryId={0}"`, you must specify a URL parameter with the **CategoryId** property.

You also can create custom actions for entities in the BDC. Custom actions allow users to navigate to any custom page to display business data. To define a BDC action, follow the steps in the [Step 3 \(Optional\): Define Actions](#) article on MSDN.

In the Partner Portal application, the product and category profile pages are hosted in the Product Catalog site collection. For more information, see [Product Catalog](#).

Using WCF Services

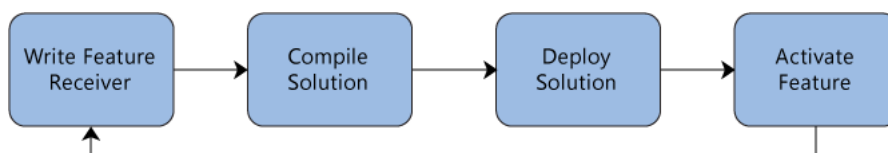
[Windows Communication Foundation](#) (WCF) is a programming model from Microsoft that allows developers to build service-oriented applications. WCF provides security, reliability, transactional semantics, and interoperability. Programming with WCF is beyond the scope of this guidance. For more information, see the [Getting Started Tutorial](#) on MSDN.

You can access WCF services that are consumed in SharePoint applications with standard .NET Framework programming techniques. The main challenges of consuming WCF services in a SharePoint application are determining how to store and maintain the configuration information. Consumers of a WCF service must be properly configured to access the services. There are two ways to configure the clients of a WCF service. You can either use a configuration file or you can configure the client programmatically. Configuration data includes a valid service endpoint, a binding, and a behavior.

Because SharePoint is based on the ASP.NET platform, configuration data can be included in the Web.config file of a SharePoint Web application. SharePoint has a built-in deployment mechanism for making Web.config modifications. However, if the deployment process includes moving the application to more than one environment, providing the correct WCF endpoint information to the SharePoint object model is difficult. For example, during user acceptance testing (UAT), the endpoint of the Partner Portal application's pricing system points to a location on Server A. When the application is moved into the production environment, the endpoint of the pricing system points to a location on Server B. If the Web.config modifications are encoded as literals in the application code, developers must recompile the code with new endpoint information for each environment.

The following illustration shows the steps for deploying modifications to multiple environments by recompiling the code in case the Web.config modifications are encoded as literals in the application code.

Deployment to multiple environments with recompilation

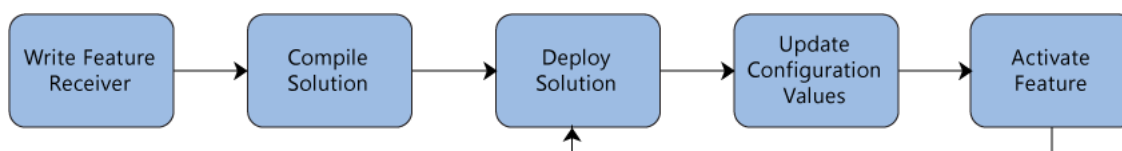


You can avoid this situation by reading configuration values from an XML file deployed to the 12 hive. For example, the Partner Portal application reads the configuration values from an XML file in a known location and uses SharePoint's **SPWebConfigModification** API to update the Web.config in all Web front-end servers. For detailed information, see the Partner Portal application's Contoso.LOB.Services.Client project.

When there are multiple Web front-end servers, the XML file with the correct WCF configuration values must be deployed to the Web front-end server where the feature receiver will execute. If the feature receiver is executed on a Web front-end server that does not have the correct WCF configuration XML file, that environment will be updated with the wrong WCF endpoints. The best practice is to update the XML file in all Web front-end servers in that environment.

The following illustration shows the steps for deploying modifications to multiple environments without recompiling the code.

Deploying to multiple environments without recompilation



To ensure that all the Web front-end servers have the same WCF configuration XML file, you can update the XML file in the Web solution package before deployment. This requires a tool for opening and repackaging CAB files.

If you do not want to use the SharePoint object model methods to update the Web.config file, you can configure the binding and behavior of a WCF client with the WCF API. WCF allows you to programmatically set the binding and behavior for clients that connect to a WCF service. The advantage to this approach is that less configuration data must be maintained in the SharePoint application. The drawback is that it requires a greater development effort to integrate the client with the WCF services. For more information, see [Managing Web.config Modifications](#).

Exposing Services from SharePoint

SharePoint provides Web services for basic administrative functions and for access to SharePoint lists. In cases where the built-in services are insufficient, you can implement custom service layers to expose the content. For example, you can provide better document management or more secure list access.

This section contains the following subsections:

- [Exposing WCF Services from SharePoint](#)
- [Exposing ASMX Services from SharePoint](#)
- [Choosing Between WCF and ASMX](#)

Exposing WCF Services from SharePoint

Windows Communication Foundation (WCF) is appropriate whenever a SharePoint application must use a transport other than the basic HTTP binding to integrate with a line-of-business (LOB) system. Examples of other transports are TCP and Message Queuing (also known as MSMQ). If you can use the basic HTTP binding, you should consider using ASMX Web services. A difference between WCF and ASMX Web services is that WCF provides a more complete security layer than ASMX Web services.

Although WCF services support many transport options, they are not fully integrated into SharePoint. For example, SharePoint cannot natively host WCF services as it does ASMX Web services. Because of this limitation, WCF services must be hosted in a Web site that does not host a SharePoint Web application. The WCF Web site must be created in Internet Information Services (IIS) with a different port and virtual directory than the SharePoint application. Giving the WCF Web site its own port and virtual directory isolates the service into its own application domain. The WCF service implementations must be hosted on the same server that runs the SharePoint application.

To infer the context of the request to SharePoint, do one of the following:

- Bind the WCF service to a specific SharePoint site or site collection.
- Pass the Uniform Resource Identifier (URI) of the SharePoint site or site collection with each request.

Exposing ASMX Services from SharePoint

ASP.NET-hosted Web services are natively supported by SharePoint. SharePoint ships with a set of ASMX [services](#) for functions such as administration and accessing lists. When these services do not provide enough functionality, developers can write custom ASMX services that can be hosted within SharePoint. Developing an ASMX service for SharePoint is a well-documented procedure. For a step-by-step walkthrough, see [Walkthrough: Creating a Custom Web Service](#) on MSDN.

Choosing Between WCF and ASMX

The following table lists some issues to consider when you must choose between WCF and ASMX to implement a service.

Characteristic	ASMX	WCF
Development effort	ASMX requires a lower level of development skills than WCF.	WCF can require a higher level of development skills because it has a larger, more flexible programming model.
Flexibility	ASMX is accessible only through the basic HTTP transport.	With configuration, WCF service implementations can support many transport options.
Security	The ASMX security layer has fewer security features than the WCF security layer.	The WCF security layer has more security features than the ASMX security layer.
Performance	ASMX has lower performance characteristics than WCF (see the note that follows this table).	WCF has higher performance characteristics than ASMX (see the note that follows this table).
Can be hosted in SharePoint	Yes.	No.

Note:

For more information about the performance characteristics of ASMX and WCF services, see [A Performance Comparison of Windows Communication Foundation \(WCF\) with Existing Distributed Communication Technologies](#) on MSDN.

Security Considerations for LOB Integration

Identity and access control are common issues faced by developers of Web-based applications. When you integrate a SharePoint application with a back-end line-of-business (LOB) system, you must consider how to convey the user's identity, or other required information, to the Web service. When a SharePoint application issues a request, the LOB system does not have direct access to the identity of the end user of the SharePoint application. The following are three approaches to this challenge:

- **Trusted subsystem.** The Web service authenticates the identity of the SharePoint application and then delegates access control decisions to the application.
- **Single sign-on.** The identity of the SharePoint user is mapped to account credentials that are understood by the back-end system. Access control decisions are the responsibility of the Web service or back-end LOB system.
- **Trusted façade.** The Web service relies on the SharePoint application to forward the identity of the authenticated user to the back-end system.

For more information about how the Partner Portal application gains access to the LOB systems, see [Security Decisions](#).

The Trusted Subsystem

In the trusted subsystem model, the identity of the calling application is trusted by the service. Typically, the calling application's identity is a process identity. For example, it might be the process identity of the SharePoint Web application's Internet Information Services (IIS) application pool. With a trusted subsystem, the Web service never sees the identity of the user who accesses the SharePoint site. Instead, the service trusts the SharePoint application to protect the information and operations that it provides to SharePoint. The trusted subsystem is supported by the Business Data Catalog (BDC) and by Windows Communication Foundation (WCF)-based services.

The advantage of the trusted subsystem is simplicity. It is best suited to situations where no complex authorization decisions that are based on the user's identity must be made. For more information about the trusted subsystem, see [Trusted Subsystem](#).

Trusted Façade

The trusted façade is similar to the trusted subsystem in that the service trusts the identity of the SharePoint application pool. However, the trusted façade also provides the service with an identity that is related to the user who views the information retrieved from the Web service. SharePoint, in this case, acts as a security façade for the Web service. SharePoint may provide the actual identity of the authenticated user, or it may provide a role that is associated with the authenticated user. The service uses this information to apply more specific authorization rules than are possible with the trusted subsystem. In the Partner Portal application, the pricing service receives the identity of the partner company that employs the user. It applies pricing rules that are based on a contract that the partner has established with Contoso. The trusted façade does not store any additional user passwords in the SharePoint tier. For more information, see [The Trusted Façade Pattern](#).

Single Sign-On

The term single sign-on (SSO) has several possible meanings. In SharePoint, SSO is defined as credential mapping. SharePoint SSO maps authenticated SharePoint users to credentials used to access a back-end system. Microsoft Office SharePoint Server 2007 includes an SSO capability that is integrated with the BDC. This SSO capability uses the Windows identity of the authenticated user accessing SharePoint when performing the credential mapping. When using forms-based authentication, a Windows identity is not established for the user accessing SharePoint; therefore, the SSO capability of Microsoft Office SharePoint Server 2007 cannot be used. For more information, see [Plan for single sign-on](#) on TechNet and [Single Sign-on](#) on MSDN.

Note:

The Partner Portal application uses a trusted façade instead of SSO. Although it is possible to include a custom SSO provider, it requires a level of complexity that is inappropriate for a reference implementation.

Considerations for Content-Driven Applications

A content-driven application's primary purpose is to deliver information such as documents, videos, and images. This is in contrast to other types of applications such as transactional applications for banks. In a content-driven application, one group of people either authors or provides some information for another group of people. An example of a simple content-driven application is one where employees upload expense reports and managers approve them. Content-driven applications also often provide collaborations spaces where users can work together and share the information.

SharePoint is an ideal platform for developing enterprise-scale, content-driven applications. The rest of this topic discusses how to structure content-driven information and who develops content-driven information. There are links to other pertinent guidance at the end of the topic.

Information Architecture and Site Topology

One of the most important aspects of a content driven-application is how the information is structured. Generally, a logical information architecture enables users to easily find the content they need. It also helps make the application scalable and maintainable. In terms of SharePoint, the information structure has many implications. It affects how your content is authored, secured, and aggregated.

Content can be stored at one of four levels. These levels are the farm, the Web application, the site collection, and the site. Each level has its own advantages and constraints. The Partner Portal application provides an example of the decisions you must make when you organize your information.

Contoso offers special promotions to all of its partners, but one partner does not necessarily receive the same promotions as another partner. The Partner Portal application provides the promotions to the partners. Several options for how to store the promotions were considered when the application was designed.

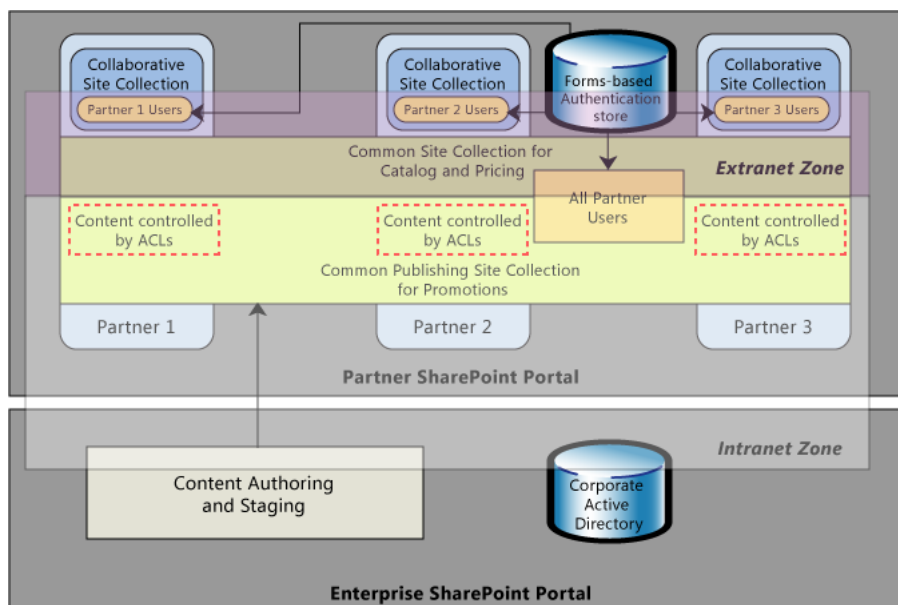
One approach was to author and store promotions in each partner's site collection. This makes the content easy to secure but difficult to maintain because there can be duplicate promotional information on each site.

A second approach was to create a central publishing site collection to manage all promotions and to use item-level access control lists (ACL) to secure the promotions. This makes the content easy to maintain but difficult to secure because all the individual ACLs must be maintained. Using individual ACLs can also have performance implications.

A third option was to place the partners into groups with different privileges. The appropriate promotions would be stored in a site collection that was dedicated to a group. This approach is a compromise between the other two approaches. It is harder to secure than the first option but easier than the second option. It is easier to maintain than the first option but harder to maintain than the second option.

The following diagram demonstrates the Partner Portal application's information architecture. It uses the second approach.

Partner Portal application's information architecture



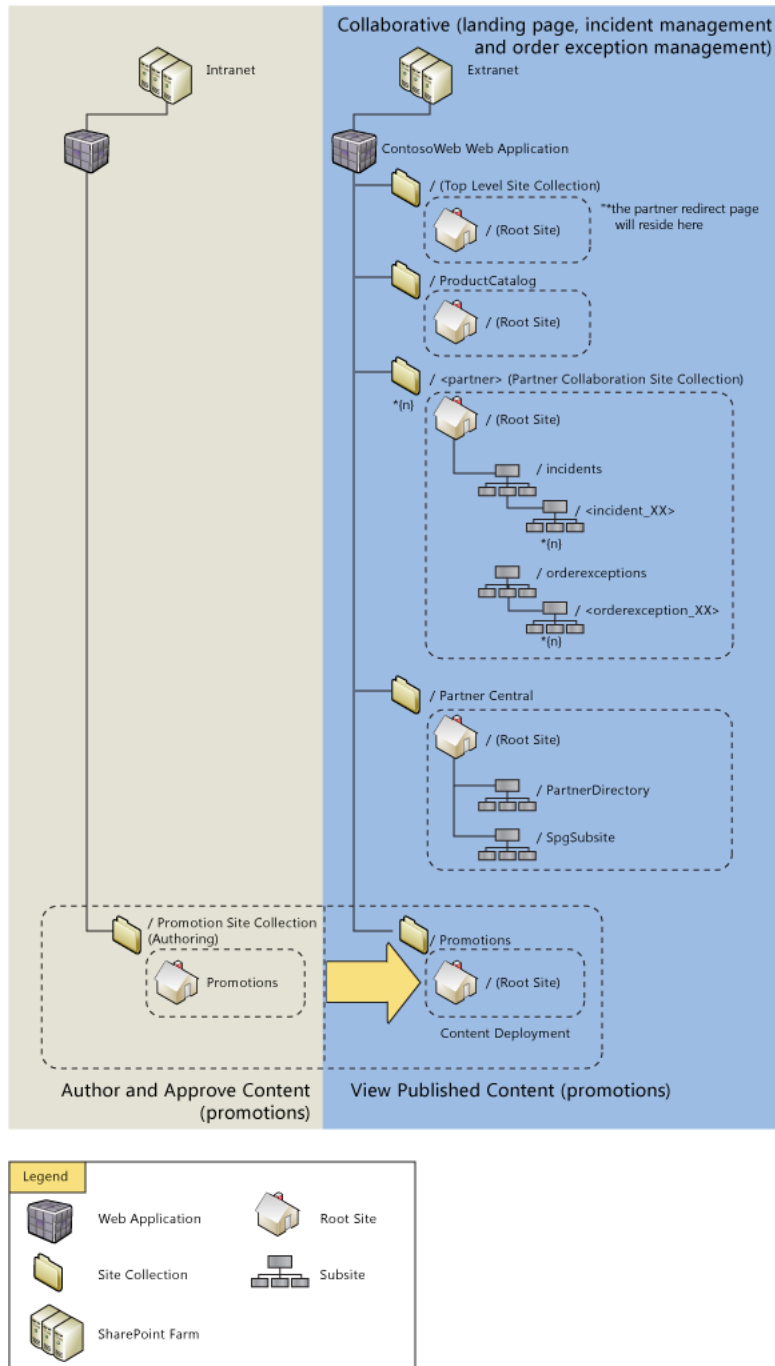
Partners are authenticated in the extranet with forms-based authentication ("FBA" in the preceding figure). Each partner has a separate site collection. These partner-specific site collections include collaboration spaces to address incidents and order exceptions.

The Partner Portal also includes a site collection for the product catalog and for pricing information. All partners can view the shared content in the Catalog/Pricing site collection, but the Web Part content is personalized for each partner. The product catalog is shared content. All partners see the same catalog. Pricing information is specific to the partner.

All partners can also view the shared content in the publishing site collection for promotions, but the promotion page instances have ACLs. The ACL guarantees that only authorized partners can see that promotion page. Promotion pages are authored in the intranet and deployed to the Promotions site collection in the extranet. SharePoint provides publishing features that allow you to create content within your corporate network and deploy it to the extranet.

The following diagram shows the site topology of the Partner Portal.

Topology of the Partner Portal



Creating a Content-Driven Application

Typically, custom SharePoint applications are created by developers. For example, a developer can write a custom Web Part that retrieves data from a persistent store. In content-driven applications, a great deal of work is also done by non-technical content authors such as business analysts and Web designers. While developers use Visual Studio, content authors use tools such as Microsoft Office SharePoint Designer and the SharePoint Web user interface (UI). Traditional developers develop components such as custom controls and Web Parts. Content authors are responsible for activities such as setting up workflows, constructing page layouts, and authoring content for a page.

In other words, creating a content-driven application can involve many people with skills in many different areas. Each area has its own challenges. This guidance includes the following topics:

- [Techniques for Aggregating List and Site Information](#)
- [Cross-Site Collection Navigation](#)
- [Understanding Custom Site Navigation](#)
- [Developing Custom Publishing Page Layouts](#)
- [Options for Site Branding](#)
- [Understanding Publishing and Content Deployment](#)
- [Using the Central Template Gallery](#)
- [Extracting Artifacts Created with SharePoint Designer and the Browser](#)

Integrating Publishing Sites and Collaboration Sites

Two distinct facets of Microsoft Office SharePoint Server 2007 are its collaborative capabilities and its publishing capabilities.

SharePoint provides collaboration sites, which are workspaces where users can contribute information and work together. For example, collaboration sites can host discussion threads, allow you to share documents, and coordinate tasks. When you create a collaboration site, it is important to structure it in a logical and straightforward way. Users are active contributors who must be able to add and find information easily. Typically, collaboration sites have more contributors than readers.

SharePoint's publishing capabilities provide the ability to author content that is meant for Internet Web sites. In the case of publishing sites, the information is primarily meant to be seen. The aesthetic appeal and branding of that information is more important than how the information is used. Publishing sites should make it possible to easily author new content. Typically, publishing sites have many more readers than contributors.

Combining a collaboration site with a publishing site provides compelling opportunities for creating innovative enterprise-scale applications. The collaboration portal site definition that is provided by Microsoft Office SharePoint Server 2007 is an example of how SharePoint's collaboration features work together with its publishing features.

When you create a site collection that is based on the collaboration portal site definition, SharePoint generates subsites for you. Some subsites are meant for collaboration and some are meant for publishing. The News subsite is an example of a publishing site. A publishing site allows a small number of content developers or authors to contribute news articles that many more users will read. The content can be authored with rich HTML, images, and media. The Document Center subsite is an example of a collaborative site. Many users can upload documents that can then be shared and worked on by other users. The Document Center also includes lists such as the Tasks list that coordinates the efforts around the documents.

The Partner Portal application that is included with this guidance integrates SharePoint's publishing features with its collaboration features. The Partner Portal is a centralized extranet site where Contoso partners can collaborate with Contoso employees to resolve incidents and order exceptions. The Partner Portal uses tasks and document libraries for collaboration. Each partner can contribute to the Tasks list and to the document library.

The Partner Portal also provides a central location where partners can receive communications and information that is meant only for them. For example, partners see promotions, company news, and partner-specific news. The attractiveness and branding of the content on these sites is more important than how the information is structured. Partners are viewers, not contributors.

Techniques for Aggregating List and Site Information

Aggregating list items is one of the most common tasks that a SharePoint application performs. The list items can be distributed across many sites and site collections. This means that aggregating data can be a costly operation for systems with complex topologies. Consequently, it frequently causes performance issues.

Although SharePoint offers several approaches that minimize the impact of aggregating data, developers and architects should perform this task judiciously. Always consider the amount of information that is returned by a query. Instead of returning many rows and then using an Extensible Stylesheet Language Transformation (XSLT) transformation to filter out all but a few of them, consider optimizing the query. For more information about designing information architecture, see [Determine the information architecture of your site](#) and [Information architecture in Office SharePoint Server](#), on TechNet.

After you decide that you must aggregate information, determine which aggregation technique is appropriate. You should consider the following points:

- **Aggregation scope.** Do you need to aggregate information from a single site or from multiple sites? Will the query traverse sites within a site hierarchy or cross-site hierarchies? Does the query span site collections?
- **Information scope.** What is the type of information being queried? Does the query apply only to list items, or does it involve other objects like aggregating information stored in property bags? Do you need to derive information from the relevant objects?
- **Performance characteristics.** How much data will be retrieved? How much flexibility do you need in manipulating the retrieved data? For detailed information about list access performance, see [Working with Large Lists in Office SharePoint Server 2007](#).
- **Data staleness.** Does the retrieved data need to be real time or is the possibility of the data being stale acceptable?

Within site collections, SharePoint can use SQL to query the site collection database. However, if you want to aggregate information across site collections, you can no longer use SQL. This means that you must implement the in-memory data joins yourself. In most circumstances, a better approach is to use SharePoint Enterprise search and accept some data latency.

The following table characterizes the different aggregation techniques.

Aggregation technique	Characteristics	Appropriate use
Content Query Web Part	Aggregation scope: site collection. Information scope: aggregates data along the site hierarchy; only works with list data; works with multiple content types; does not provide direct access to SharePoint objects (see Portal site map provider). Performance: optimized for cross-list queries. Uses the object cache. Accuracy: Can control if results are cached. If data is cached, then may be stale.	This Web Part is only for querying list data. The SharePoint user interface only allows XSLT transformations. It is only available with Microsoft Office SharePoint Server. For an example of how to use this Web Part, see the Partner Portal application. The ActiveIncidentTasksWebPart is a Content Query Web Part that aggregates all of a partner's open incident tasks and displays them on the incident dashboard. Used for no-code solutions.
Portal site map provider	Aggregation scope: site collection. Information scope: aggregates data along the site hierarchy; works with multiple content types; works with sites, Webs and lists; provides direct access to SharePoint objects for data manipulation, such as calculating values. Performance: optimized for queries. Uses the object cache. Accuracy: Cached result sets may be stale.	It is more complex than the Content Query Web Part. It is only available with Microsoft Office SharePoint Server. For an example, see the Partner Portal application. The IncidentStatusListWebPart in the Contoso.PartnerPortal.Collaboration.Incident project uses a portal site map provider object in its presenter class (IncidentStatusListPresenter.cs) to aggregate all of a partner's incidents and to display them on the incident dashboard. Used for code based solutions.
Direct cross-site collection query	Aggregation scope: across a server farm. Information scope: provides	It cannot traverse a site hierarchy. It is best suited for targeted, cross-farm queries. For an example, see the Partner Portal application. The

	no data aggregation; works with multiple content types; works with sites, Webs and lists. Performance: Not optimized or cached; should only be used for targeted queries. Accuracy: no latency.	PartnerPromotionsWebPart that is on the partner home page uses a direct cross-site collection query to retrieve all the promotions that are intended for the currently logged-in user. To perform the query, it calls the partner promotion repository's GetAllMyPromos method.
Search	Aggregation scope: works across the server farm. Information scope: aggregates along and across arbitrary information hierarchies and content types; does not provide direct access to objects. Performance: Can query large data sets across sites and the server farm; can perform complex query and document searches; efficient for aggregating different types of data or data that is spread across a number of site collections. Accuracy: Data is as accurate as the last crawl.	The FullTextSqlQuery class is only available with the Microsoft Office SharePoint Server object model. Security information is collected at the time of the crawl. This requires specialized logic for non-native search data sources. It is best suited for aggregating information across site collections or a server farm. Data latency must be acceptable. See the Partner Portal application for an example. The PartnerRollupWebPart in the Contoso.PartnerPortal.PartnerCentral project uses the search API to aggregate all open tasks for all partners. It displays them on the Partner Central site for administrative purposes.

Using the Content Query Web Part

The **ContentByQueryWebPart**, commonly referred to as the Content Query Web Part, is a part of the Enterprise Content Management (ECM) functionality in Microsoft Office SharePoint Server. It aggregates and displays list items within a site hierarchy. In addition to hierarchical query capabilities, the Content Query Web Part provides caching and query optimization for the **SPSiteDataQuery** object that it contains. These optimizations have better performance than if you directly call methods of the [SPSiteDataQuery](#) object. You can also use SharePoint Designer to customize the aggregation and display capabilities of the Content Query Web Part with little or no coding. For more information, see [ContentByQueryWebPart Class](#) and the following procedures on MSDN:

- [How to: Customize the Content Query Web Part by using Custom Properties.](#)
- [How to: Customize XSL for the Content Query Web Part.](#)
- [How to: Display Custom Fields in a Content Query Web Part.](#)
- [How to: Customize RSS for the Content Query Web Part.](#)

The Content Query Web Part has the following limitations:

- You can only aggregate data within a single site collection.
- You can only aggregate list information.

Alternatively, you can use the Content Query Web Part's underlying [CrossListQueryInfo](#) and [CrossListQueryCache](#) classes to retrieve list information programmatically. However, the recommended approach for a programmatic query is to use the [PortalSiteMapProvider](#) class. This class offers additional features that the Content Query Web Part does not have. For more information, see [Using the PortalSiteMapProvider](#) on MSDN.

Note:

The Content Query Web Part is only supported in publishing sites. Also, the site collection that contains the site must have the publishing features enabled. In any other context, the results may be inconsistent. For example, RSS feeds will not be correctly queried.

Another option is to use SharePoint search. This is appropriate when you must retrieve large result sets and can accept some data latency. For more information, see [Using Search to Aggregate Data](#).

Using the PortalSiteMapProvider

The [PortalSiteMapProvider](#) class is the navigation site map provider class for Microsoft Office SharePoint Server 2007. It is useful for aggregating information because it provides an alternative to the Content Query Web Part for cached queries. It also provides access to cached object stores.

The **PortalSiteMapProvider** class allows you to access cached objects for sites, lists, and list items. It also provides methods for executing **SPQuery** and **SPSiteDataQuery** queries against cached SharePoint lists. You can use the **GetCachedListItemsByQuery** method to cache the query results. Executing a **SPSiteDataQuery** query with a **PortalSiteMapProvider** object is similar to using the **CrossListQueryCache** class.

A primary advantage of the **PortalMapSiteProvider** class is its use and management of the SharePoint object cache. The object cache provides highly efficient access to objects that are not part of a list. By default, the **PortalSiteMapProvider** class queries and manages the object cache. This feature does not require any extra code.

The following example from the Partner Portal application demonstrates how to use the **PortalSiteMapProvider** to retrieve all incident sites within a particular partner's site collection. It retrieves the information from the property bag for each site. Because this information does not reside in a list, neither the Content Query Web Part nor the **SPSiteDataQuery** class can easily retrieve the information.

C#

```
public void SetIncidentStatusListData()
{
    PortalSiteMapProvider provider = PortalSiteMapProvider.CombinedNavSiteMapProvider;
    if (provider.CurrentNode != null && provider.CurrentNode.HasChildNodes)
    {
        List<IncidentStatus> incidentStatuses = new List<IncidentStatus>();
        foreach (PortalWebSiteMapNode node in
            provider.GetChildNodes(provider.CurrentNode))
        {
            if (node.GetProperty(BusinessEventProperty) != null &&
                node.GetProperty(BusinessEventProperty).ToString().ToLower() == "incident")
            {
                IncidentStatus incidentStatus = new IncidentStatus();
                incidentStatus.Title = node.Title;
                ...
                incidentStatuses.Add(incidentStatus);
            }
        }

        view.Data = incidentStatuses;
    }
}
```

The **PortalSiteMapProvider** class shares the same site collection boundary as the Content Query Web Part. The **PortalSiteMapProvider** class is only available on sites where the Microsoft Office SharePoint Server 2007 Publishing feature is activated.

Using Search to Aggregate Data

The search feature efficiently queries large sets of distributed information. If you must aggregate information across site collection boundaries, it is preferable to use search instead of a query. This is because the data can be distributed across multiple query servers and cached as the search engine indexes the content.

Using Office SharePoint Server Search

An example of when to use search is in the Partner Portal application. It uses search to aggregate information about outstanding tasks. Contoso employees who work with partners to resolve issues must be able to view all the open tasks that are associated with the incident sites. These tasks are distributed across multiple site collections. The number of tasks and partner site collections varies as partners are added and removed. Potentially, this number can be very large.

The search feature aggregates data across an entire server farm. It includes two classes for different types of queries. The **KeywordQuery** class is appropriate for simple searches that are based on words, phrases, or prefixes. The **FullTextSqlQuery** class is for more complex queries that need to use advanced features of the search engine.

Using search does have implications for IT departments. The search services must be installed, set up, and maintained. However, many organizations rely on the search feature, and it is increasingly common for it to be part of the standard infrastructure. Although this guidance specifically discusses SharePoint search, any search engine that supports programmatic access is conceptually equivalent.

Using the FullTextSqlQuery Class

To perform queries that use the SQL syntax, use the **Microsoft.Office.Server.Search.Query.FullTextSqlQuery** class. The query text is very similar to the T-SQL queries that are used in SQL Server. Results from the queries are returned as **Microsoft.Office.Server.Search.Query.ResultTableCollection** objects. These objects can be loaded into an instance of **System.Data.DataTable**. For more information, see [FullTextSqlQuery](#).

You can use scopes to narrow a search query. Scopes restrict the search to a subset of indexed items and can be defined in SharePoint's Central Administration. You can also use content types as a way to constrain search results.

The Partner Portal application uses an Incident Task content type for all incident tasks. The search is limited to content type and has an **All Sites** scope. The following query finds all incident tasks of type Incident Task across all sites.

SQL

```
SELECT Path, Title, AssignedTo, Status, Priority, ContentType FROM SCOPE() WHERE "SCOPE"='All Sites' AND ContentType = 'Incident Task' ORDER BY Path ASC
```

For more information about defining search scopes, see [Define scopes for Searches](#) on TechNet.

More Information

To learn more about the Search feature, see [MOSS 2007 Search FAQ](#).

There are security issues to consider when you enable the Enterprise search feature. By default, sites that require forms-based authentication are not indexed. For more information, see [SharePoint 2007 Tool: Add/Edit Crawl Rules with Form/Cookie Credentials](#).

To learn more about the search query object model, see [Enterprise Search Query Object Model Overview](#) on MSDN.

To learn more about the **FullTextSqlQuery** class, see [SQL: FullTextSqlQuery Event Class](#) on MSDN.

To learn more about the **KeywordQuery** class, see [KeywordQuery Class \(Microsoft.SharePoint.Search.Query\)](#) on MSDN.

To learn how to configure the Enterprise Search, see [Walkthrough: Configuring Search for the AdventureWorks Business Data Application Sample](#) on MSDN.

Using Cross-Site Collection Queries

Systems that require accurate real-time data cannot rely on the SharePoint Search feature to provide it. When real-time data must be aggregated across site collections, you must iterate through site collections and sites to perform the queries. Querying for data across site collections or the server farm has significant performance implications and requires careful consideration.

It is a recommended practice to use this approach only when there are a small number of site collections whose locations are known. For example, in the Partner Portal application, partners view their promotions from their collaboration sites. Each partner's site collection is outside of the site collection that contains the promotions data. Because the location of the promotions data is known, it is reasonable to directly query against the promotions page library. The **PartnerPromotionRepository** class uses this technique to retrieve the promotions pages. The pages are located in a specific library that is located in a central publishing site collection. Typically, you can use SharePoint Web services to query for data across the server farm.

Cross-Site Collection Navigation

Cross-site collection navigation refers to a Web-based user interface that allows you to access sites that are located in different site collections. It is also named global navigation.

Like with ASP.NET applications, SharePoint uses site maps to implement navigation. Site maps describe the structure of a Web site to the navigational controls that provide the user interface. Microsoft Office SharePoint Server 2007 includes a site map provider that allows users to navigate the sites that are within the current site collection. If you want your application to support global navigation, you can customize it in one of two ways:

- You can extend SharePoint's site map provider.
- You can write a custom site map provider.

The following subtopics discuss these techniques:

- [Extending Navigation to Cross-Site Collection Boundaries](#)
- [Creating Custom Navigation That Crosses Site Collection Boundaries](#)

Extending Navigation to Cross-Site Collection Boundaries

You can extend one of SharePoint's site map providers to enable global navigation. One way to do this is to use XML to customize the [SPXmlContentMapProvider](#) class.

Using the SPXmlContentMapProvider Class

The **SPXmlContentMapProvider** class is an XML-based site map provider that is used by SharePoint. This provider can read a navigation structure from any standard site map. SharePoint uses the **SPXmlContentMapProvider** class to provide the navigation structures for application pages that are located in the `_layouts` folder. Because the **SPXmlContentMapProvider** class reads the navigation structures from an XML file, it is not limited to a site collection boundary. The XML file can include URIs that are outside of the current site collection.

There are two ways to use XML to customize the **SPXmlContentMapProvider** class:

- You can create a custom site map.
- You can create a site map that will be merged into SharePoint's site map.

The next sections describe each of these.

Creating a Custom Site Map

A site map contains a hierarchical structure of **sitemap** elements. You must create an XML file that incorporates each site collection into this common structure. The site map file must have the `.sitemap` file name extension. After you create the site map, copy it to the `_app_bin` folder of the SharePoint Web application. Finally, use the SharePoint Web application's `Web.config` file to configure an instance of the **SPXmlContentMapProvider** class. Copy the site map file to the Web front-end server. Use SharePoint's deployment infrastructure to do this.

Merging XML Fragments with the layouts.sitemap File

SharePoint allows developers to deploy files to the `_layouts` folder of a Web front-end server. It also allows developers to provide XML fragments of a site structure that can be merged into the `layouts.sitemap` file. The name of the XML file that contains the new site structure must begin with `layouts.sitemap`. For example, a valid file name is `layouts.sitemap.contosoglobalnavigation.xml`.

When SharePoint starts, it scans all the XML files that are in the `_layouts` folder whose names begin with `layouts.sitemap`. It merges the contents of these files with the `layouts.sitemap` file that is located in the Web application's `_app_bin` folder. Because SharePoint is already configured to read the site map from the `layouts.sitemap` file, you do not need to configure an instance of the **SPXmlContentMapProvider** class.

Considerations for Multiple Web Front-End Servers

Both creating a custom site map and merging XML fragments present challenges if you have multiple Web front-end servers. Because the **SPXmlContentMapProvider** class is a file-based site map provider, you must ensure that the XML file that defines your navigation structure is installed on every Web front-end server in the farm.

SharePoint does not provide a way to synchronize the files in the `_app_bin` folder with all of the Web front-end servers in a farm. If you are merging XML fragments, SharePoint's deployment infrastructure can deploy the XML file to those servers. However, the API that merges those fragments with the `layouts.sitemap` file only works on a single Web front-end server. Developers or system administrators must manually perform the merges on all the servers each time that the site map changes.

Another consideration for using the **SPXmlContentMapProvider** class is that SharePoint does not provide a user interface to manage the custom navigation structure. This can make it difficult to work with XML files that have a complex structure.

Creating Custom Navigation That Crosses Site Collection Boundaries

You can implement global navigation by creating a custom navigation solution that can cross site collections. One reason to do this is to store your site map in a centralized location. A central location reduces the costs of deploying and maintaining your navigation solution.

Implementing a Custom Site Map Provider

If you want a custom solution, you must implement a custom site map provider. This allows you to make your site map useable by any navigational control you choose, such as a menu, a breadcrumb or a tree. The ASP.NET framework includes base classes that you can use to create a custom site provider.

For example, the Partner Portal application extends the **StaticSiteMapProvider** class. This class translates a site map that is in persistent storage to one that is stored in memory. The contents of the site map are read once. This is when the application pool for a Web site starts. The **StaticSiteMapProvider** class is a good choice when the site map in the persistent store changes infrequently. When the site map in the persistent store does change, the site map provider will not reflect those changes until the application pool recycles.

Using the Hierarchical Configuration

When you implement a custom site map provider, you must decide on a persistent storage mechanism for the site map. There are number of options, including a custom database table, the SharePoint_Config database, or a SharePoint list. The Partner Portal application uses the [Configuration Manager](#) that is part of the SharePoint Guidance Library. It stores the entire XML navigation fragment as a single configuration value. In the Partner Portal application when the application pool starts, the Contoso.Common.Navigation.HierarchicalConfigSiteMapProvider reads the XML from the Hierarchical Configuration and then loads it in memory as a site map.

Other Considerations

It is important to thoroughly test the code that implements the custom site map provider. An exception that occurs while the site map provider is initializing can prevent your application from loading, even if the site map provider is not used on a page that a user requests. Also, a faulty site map provider can be difficult to troubleshoot.

One of the drawbacks of a custom site map provider is that SharePoint does not provide a user interface (UI) that allows you to modify the site map at run time. This can make an application's navigation static because it is difficult to add new nodes. On the other hand, SharePoint's site map providers are dynamic. It is simple to update and modify the site structure if you use them. It is possible but potentially time consuming to build a custom UI to modify a site map that is in a persistent store.

Understanding Custom Site Navigation

Concept

In a standard SharePoint site, users have two types of navigation:

- **Global navigation** (this is also named the Top Navigation menu). This begins at the top-level site.
- **Current navigation** (this is also named Quick Launch). This starts at the current site.

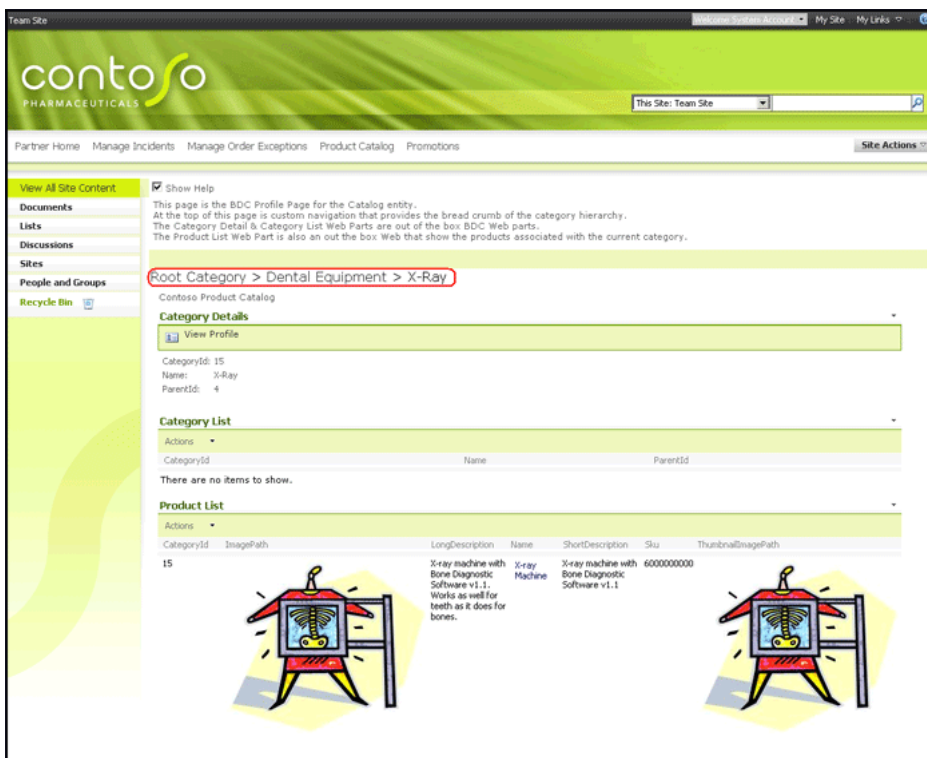
These types of navigation are based on the site hierarchy and on the pages within each site in a site collection. If you want users to have other options, such as navigating through a hierarchical structure of product categories, you must create some custom navigation.

For example, the Partner Portal application uses the Business Data Catalog (BDC) to expose information from a product catalog system. These categories are stored hierarchically, but the BDC uses a single profile page to display data about a particular category. The profile pages are distinguished from each other by query string parameters in the URL.

The Partner Portal application uses custom navigation to display the catalog hierarchy of the products as partners browse the site. These sites are displayed as a series of links at the top of the Web page. Displaying a hierarchy is sometimes referred to as *bread crumbing*. A common example of bread crumbing in Windows is the path you see in the address bar when you look at the contents of a folder. The links go from the root, such as Computer, to your present location.

The following figure illustrates an example of a Web page that displays the catalog hierarchy.

Contoso catalog page with bread crumb navigation



The custom navigation shows the root category, then the dental equipment category, and then the x-ray category. To implement the custom navigation, a custom site map provider supplies data to an ASP.NET control, which displays the data as a sequence of links at the top of the Web page.

Implementation Details

The following procedure shows the general steps required to implement a breadcrumb user interface (UI).

To implement custom navigation

1. Create a custom class that inherits from the **SiteMapProvider** class. Implement override methods in your **SiteMapProvider** class to create site map nodes for the custom breadcrumb UI that you want to display. The methods to override are **FindSiteMapNode** and **GetParentNode**. See [Creating the Custom Site Map Provider Class](#) and [Building the Navigation Path for an URL](#) later in this topic.
2. Configure and register the custom site map provider. See [Configuring and Registering the Site Map Provider](#) later in this topic.
3. Display the ASP.NET **SiteMapPath** control. See [Displaying the ASP.NET SiteMapPath Control](#) later in this topic.

Each of these steps is explained in the following sections.

Creating the Custom Site Map Provider Class

The Partner Portal application includes the **BusinessDataCatalogSiteMapProvider** class to implement custom navigation. This class inherits from a .NET Framework class that is named **SiteMapProvider**. Override the methods of this class to create a custom class. Unlike the **HierarchicalConfigSiteMapProvider** class, the **BusinessDataCatalogSiteMapProvider** does not store the navigation structure in memory.

Building the Navigation Path for a URL

The **SiteMapPath** control runs each time a partner clicks on a category page. The control invokes the **FindSiteMapNode** method of the **BusinessDataCatalogSiteMapProvider** class. The raw URL of the page that is displayed is the method's argument. The method returns an object of type **SiteMapNode** because this is the type that the site map path control requires to construct the breadcrumb UI.

Each component of the breadcrumb path corresponds to a **SiteMapNode** object. Each **SiteMapNode** object has a display string and an associated target URL. In the case of nodes that are created by the custom site map provider, the URL represents a category profile page URL with a particular ID as a query string parameter.

When the ASP.NET **SiteMapPath** control calculates the sequence of **SiteMapNode** objects for the Web page that is displayed, it begins by finding the site map node for the current page. To do this, it calls the **FindSiteMapNode** method to obtain a **SiteMapNode** object that is created from the raw URL of the current page. This method is in the **BusinessDataCatalogSiteMapProvider.cs** file that is located in the folder `\Source\PartnerPortal\Contoso.PartnerPortal.ProductCatalog\Navigation`.

The **FindSiteMapNode** method extracts the category ID from the query string of the currently requested page and calls into the repository to retrieve the corresponding category record from the BDC. The resulting site map node includes the category ID, the raw URL, and the name that corresponds to the category ID.

Note:

The repository that holds the category information is cached to improve the performance of the data lookups. By default, BDC does not cache any data. Whenever data is requested through the BDC, it always calls the external system. This can cause considerable performance degradation when many users are navigating through the categories.

The following code shows how the **FindSiteMapNode** method obtains a **SiteMapNode** object.

C#

```
public override SiteMapNode FindSiteMapNode(string rawUrl)
{
    SiteMapNode node = null;

    string queryString = rawUrl.Split("?".ToCharArray())[1];
    string[] query = queryString.Split("&".ToCharArray());
    string[] queryParts = query[0].Split("=".ToCharArray());
    string categoryId = queryParts[1];

    Category category = this.productCatalogRepository.GetCategoryById(categoryId);
    if (category != null)
    {
        node = new SiteMapNode(this, categoryId, rawUrl, category.Name);
    }

    return node;
}
```

After the **SiteMapPath** control has the **SiteMapNode** object for the currently displayed page, it iteratively calls the **GetParentNode** method of the current node, backtracking until it discovers the breadcrumb trail from the current page to the root page. It then formats and displays links for each element of this hierarchy, starting with the root node and proceeding to the site map node for the current page. The **GetParentNode** method is in the **BusinessDataCatalogSiteMapProvider.cs** file located in the folder `\Source\PartnerPortal\Contoso.PartnerPortal.ProductCatalog\Navigation`.

The following code shows how the **GetParentNode** method constructs a navigation path.

C#

```
public override SiteMapNode GetParentNode(SiteMapNode node)
{
    SiteMapNode parentNode = null;

    Category currentCategory =
        this.productCatalogRepository.GetCategoryById(node.Key);
    if (currentCategory != null)
    {
        Category parentCategory =
            this.productCatalogRepository.GetCategoryById(currentCategory.ParentId);
        if (parentCategory != null)
        {
            parentNode = new SiteMapNode(this,
```



```

        parentCategory.CategoryId,
        string.Format(this.profileUrl,
                      parentCategory.CategoryId),
        parentCategory.Name);

    if (parentCategory.CategoryId == "0")
    {
        this.rootNode = parentNode;
    }
}

return parentNode;
}

```

Any category that has the ID of 0 is a root category node. After the method finds a category with an ID of 0, it stops calling the **GetParentNode** method.

For more information about the **SiteMapNode** class, see [SiteMapNode Class](#) on MSDN.

Configuring and Registering the Site Map Provider

The new site map provider is registered in the Web.config file of the current Web application. It is associated with the symbolic name **CategorySiteMapProvider** that is referenced by the site map's ASP.NET control.

The **WebFeatureReceiver** class defines the modification to the Web.config using an instance of **SPWebConfigModification**.

```

C#
public WebFeatureReceiver()
{
    SPWebConfigModification = new
        SPWebConfigModification("add[@name=\"CategorySiteMapProvider\"],
        \"configuration/system.web/siteMap/providers\");
    SPWebConfigModification.Owner = GetType().Assembly.GetName().ToString();
    SPWebConfigModification.Type =
        SPWebConfigModification.SPWebConfigModificationType.EnsureChildNode;
    SPWebConfigModification.Value = @"<add name=\"\"CategorySiteMapProvider\"\"
description=\"\"Provider for category navigation using Business Data Catalog\"\"
type=\"\"Contoso.PartnerPortal.ProductCatalog.Navigation.BusinessDataCatalogSiteMapProvider,
Contoso.PartnerPortal.ProductCatalog, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=0dcd9137292eac97\"\" />";
}

```

For more information about using the **SPWebConfigModification** class to modify the Web.config file, see [Managing Application Configuration](#).

The **FeatureActivated** method of the **WebFeatureReceiver** class registers the provider during deployment. The method is declared in the WebFeatureReceiver.cs file of the Contoso.PartnerPortal.ProductCatalog project.

```

C#
[SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
public override void FeatureActivated(SPFeatureReceiverProperties properties)
{
    /// ...
    web.Site.WebApplication.WebConfigModifications.Add(spWebConfigModification);

    web.Site.WebApplication.Update();
    web.Site.WebApplication.WebService.ApplyWebConfigModifications();
}

```

The new node registers a site map provider that is named **CategorySiteMapProvider**. This provider is mapped to the **BusinessDataCatalogSiteMapProvider** class and used in the Category.aspx page.

For more information about the **SPWebConfigModification** class, see [SPWebConfigModification Class \(Microsoft.SharePoint.Administration\)](#) on MSDN.

Displaying the ASP.NET SiteMapPath Control

The ASP.NET **SiteMapPath** control is a UI control that displays a sequence of links, from a root page to the currently displayed page. To display the links, the **SiteMapPath** control must be included on the category page. It must also be associated with the **BusinessDataCatalogSiteMapProvider** class that calculates the breadcrumb links, which are based on the BDC category information. The **SiteMapPath** control retrieves these links from the site map provider.

In the Partner Portal application, the category page includes a site map path control that references the site map registration that was added to the Web.config file. The following XML code includes the **SiteMapPath** control and associates it with the **BusinessDataCatalogSiteMapProvider**. It uses the symbolic name **CategorySiteMapProvider** that was registered as part of the configuration process. The code is located in the \Source\PartnerPortal\Contoso.PartnerPortal.ProductCatalog\WebPartPages\Category\Category.aspx file in the Contoso.PartnerPortal.ProductCatalog project.

XML

```
<tr>
  <td>
    <asp:SiteMapPath
      SiteMapProvider="CategorySiteMapProvider"
      runat="server"
      ID="SiteMapPath1"
      NodeStyle-CssClass="ms-sitemapdirectional" />
  </td>
</tr>
```

The **SiteMapPath** control uses the Web.config file to look up the type (this is the **BusinessDataCatalogSiteMapProvider** class) for the site map provider that is named **CategorySiteMapProvider**. It then instantiates the class and calls the methods that are on that class. For more information about the **SiteMapPath** control, see [SiteMapPath Class](#) on MSDN.

An Example of Constructing a Navigation Path

Here is an example that demonstrates how breadcrumbing works, using food as an example. The catalog has a simple hierarchy that has the following structure: food, fruit, apples, Jonathan apples. The repository is a table that relates the category IDs, names, and parent nodes. Food is the root node and has a category ID of 0. The following table shows the repository.

Category ID	Name	Parent node
2	Jonathan apples	3
3	Apples	5
5	Fruit	1
1	Food	Null

For example, Jonathan apples have a category ID of 2 and the parent node has a category ID of 3, which has the name Apples.

A user has bookmarked the page for Jonathan apples. The URL for the page is <http://localhost/category.aspx?categoryID=2>. The **SiteMapPath** control calls the **FindSiteMapNode** method with this URL. The method parses it to extract the category ID value 2. It then calls into the repository with the ID and retrieves the corresponding name, Jonathan apples. It creates an instance of **SiteMapNode** with fields that contain the raw URL, the name, and the category ID.

After the **SiteMapPath** control has the current site map node, it attempts to find the parent node that corresponds to the next more general category in the breadcrumb hierarchy. To do this, the **SiteMapPath** control calls the **GetParentNode** method of the current site map node. The **GetParentNode** method calls into the repository, locates the record for the site node, and finds the parent node. It then constructs the URL for the parent node by substituting the category ID of the parent node for the category ID of the child page.

This process is repeated until the **SiteMapPath** control reaches a node with no parent. This is indicated by a null return value from the call to **GetParentNode**.

After the **SiteMapPath** control has all the site map nodes beginning with the root category down to the current page, it constructs the breadcrumb UI and renders it to the HTML page.

Additional Overridden Methods

To complete the implementation of a class that is derived from the **SiteMapProvider** class, the following properties and method are required:

- **RootNode** property. This returns the top-most node in the custom navigation hierarchy.
- **GetChildNodes** method. This returns a collection of all nodes that have as their parent node the node provided as an argument to the **GetChildNodes** method.
- **GetRootNodeCore** property. This is the same as the **RootNode** property.

Developing Custom Publishing Page Layouts

Creating publishing page layouts is a fundamental development activity in SharePoint. Page layouts are templates that govern the appearance of SharePoint Web publishing sites. They determine the structure (where information is placed) as well as the style (the format). The following sections give guidelines for developing page layouts. For additional information, see [Create a Publishing Page Layout](#) on the Microsoft Office site.

Defining a Content Type

When you develop a page layout, you should first define a content type. A content type is metadata that describes the kinds of information that can be authored for a page. Every publishing page in Microsoft Office SharePoint Server 2007 is based on a specific content type. You can use a content type for more than one page layout.

In some cases, you can create page layouts from existing contents types, but it is more common to extend an existing base content type to accommodate the information for a specific type of page. When you extend an existing content type, you save time and effort by reusing the existing fields.

Note:

Content types can be derived from other content types through their ID attributes. For a description of how to derive one content type from another content type, see [Creating Content Types Based on Other Content Types](#) on MSDN. You can find the content type IDs that are used by publishing pages in the [appendix](#) of this guidance or by browsing in the Office 12 hive folder.

Using Rich HTML Field Controls

The rich HTML field control is one of the most powerful and frequently used controls in page layouts. A rich HTML field control allows authors to structure and style their content with much more flexibility than with plain text. For example, content authors can use HTML to apply different fonts, colors, bullet points, and headings.

Rich HTML field controls can only be used with rich HTML fields. If you are developing a custom content type for your page layout, you must indicate that the field can be edited as rich text. For example, all the Partner Portal promotion pages contain a description field. The following XML shows this field.

XML

```
<Field ID="{aaedlaca-3f64-4b13-8e87-863ebc659432}"
  Type="HTML"
  Name="PromotionDescriptionField"
  DisplayName="Promotion Description Field"
  StaticName="PromotionDescriptionField"
  Hidden="FALSE"
  Required="FALSE"
  Sealed="TRUE"
  Group="Contoso Publishing"
  RichText="TRUE"
  RichTextMode="FullHtml"/>
```

The **Type** attribute indicates that this field will hold HTML as its content. The **RichText** and **RichTextMode** attributes allow authors to use rich HTML.

If you use Office SharePoint Designer 2007, you can drag your fields onto your pages and automatically see the HTML editor for that field. If you are developing your page layout by editing the markup directly, you can use a **RichHtmlField** control element with the name of the field as the value for the **FieldName** attribute. The following code shows an example.

C#

```
<PublishingWebControls:RichHtmlField ID="RichHtmlField3"
  FieldName="PromotionDescriptionField" runat="server" />
```

Developing Custom Field Controls

Many times, developers find that standard field controls do not provide enough functionality to manage content on a page layout. For example, you might want a field control to perform some custom validation on the content. Other examples of custom field controls are a rich Silverlight media control or an editor that provides a drop-down box.

The Training Management application creates a custom field (**Contoso.TrainingManagement.FieldTypes.CourseDifficultyLevelField**) with a custom field control named **Contoso.TrainingManagement.FieldTypes.CourseDifficultyLevelFieldControl**. For more information, see [Using Custom Field Types and Field Controls](#).

The Partner Portal application creates a custom field control but not a custom field. This approach is useful when you want to use an existing field type, such as **Text**, but you want to also associate custom behavior with it. With this approach, provide the name of the field as the value for the **FieldName** attribute of the control in the markup. The following code shows an example.

C#

```
<WindowsMediaFieldControl:WindowsMediaFieldControl ID="WindowsMediaField"
FieldName="WindowsMediaField" runat="server"/>
```

You must make sure that your field control supports the type of field that is provided in the **FieldName** attribute.

Composing with Business Data and Web Parts

The content on a publishing page can either be authored content or dynamic content that originates from an external source. An example of dynamic content is a shopping cart on an e-commerce Web site. None of the data that is displayed in the shopping cart is authored. It is dynamically displayed as users add items.

In the Partner Portal application, authors provide content for specific product promotions. The product information is stored in an external system. One way to retrieve the correct data from the external system is with Web Parts that use authored content to locate the data.

The Partner Portal application uses the product stock-keeping unit (SKU) that is stored in the product's SKU field to find the correct information in external pricing and catalog systems. The Page Context Filter Web Part can read values from any page field. It reads the SKU field and provides that value to the product details and pricing Web Parts with Web Part connections. These Web Parts are on the page layout. This means that every page instance automatically displays product details and prices that are tailored to a specific promotion.

Options for Site Branding

Site branding is the process of customizing a Web site to reflect an organization's identity. Branding includes the addition of logos, colors, and styles.

Branding is useful when many different organizations access the same application. The company that hosts the application may want each participant's site to have branding that is specific to them. There are a variety of ways to brand a SharePoint site. This topic discusses master pages and custom navigation.

Branding with SharePoint Themes

Themes are collections of cascading style sheets and images that you can use to customize a SharePoint site. Themes are sometimes referred to as *skins*. Themes are the primary way to customize a Windows SharePoint Server 3.0 site. Themes can be applied to content pages and to application pages. (Application pages are also known as `_layouts` pages.) Themes can be applied to Microsoft Office SharePoint Server sites in addition to Windows SharePoint Services sites. In general, applying a theme to a site is a safer option than using master pages because it is not possible to write malicious code in cascading style sheets. Themes can only change a site's style. They cannot change the organization of the elements on the site. For example, you can change the font, but you cannot change the placement of the Quick Launch.

For information about how to create and apply a theme, see [Applying the Contoso Theme](#).

Integrating Master Pages, Themes, and Cascading Style Sheets

Master pages are special types of ASP.NET pages that allow you to create a consistent layout for all pages in a Web site. SharePoint, which is built on the ASP.NET 2.0 framework, also supports master pages. In Microsoft Office SharePoint Server, master pages are the primary method for controlling the structure and aesthetics of sites. Typically, master pages are used in conjunction with cascading style sheets. The master page controls the structure of a page such as its wire frames and the placement of navigational controls and menus. The cascading style sheet changes the site's graphical qualities such as the background colors, fonts, and borders. Developers can include controls and script in master pages to provide some functionality to all the pages in the site. Microsoft Office SharePoint Server sites include a simple graphical user interface that allows you to choose master pages for your sites; Windows SharePoint Services sites do not include an interface that helps with this.

Typically, custom master pages contain a significant amount of code, which can be problematic. Master pages affect all pages, and code with bugs can cause your entire site to function improperly. Because of this, it is important to control who can modify the master page and when these modifications occur. The cost of developing and testing master pages is significantly higher than for themes. Master pages cannot control the look and feel of application pages (_layouts pages). Also, master pages are supported on Windows SharePoint Services only with a significant amount of custom code.

If possible, you should not include styles in the master page. Instead, place styles in cascading style sheets (.css files). This decreases payload sizes. A site can use both a theme and master page. One recommendation is to wrap custom .css files with a theme and apply the theme in conjunction with the master pages.

Note:

For more information about using master pages to brand Microsoft Office SharePoint Services sites, see [Customizing and Branding Web Content Management-Enabled SharePoint Sites](#) on MSDN.

Branding Decision Matrix

The following table summarizes characteristics of master pages and themes.

Capability	Master page only	Theme only	Hybrid
Brands the system page	Can use alternate CSS setting	X	X
Makes structural changes	X		X
Applies to an entire site collection	X		X
Allows new blocks of script or controls	X		X

Using Delegate Controls

Delegate controls are typically used with master pages. They allow you to replace standard SharePoint functionality with your own customizations. When a page is requested, the delegate control looks for controls that are registered for the specific page instance. If it finds them, the control with the lowest sequence number is added in place of the delegate control. By placing a delegate control on a master page, you can designate a location for your control on every page of the site.

The Partner Portal application uses a delegate control to provision the custom global navigation. The default SharePoint master page provides a delegate control for the top navigation data source control. The Partner Portal application takes advantage of this. It reuses the existing master page and modifies it with the custom global navigation code. The ID of the modified delegate control is **TopNavigationDataSource**. The following code is from the contextualHelp.master file, which is based on default.master file.

C#

```
<SharePoint:DelegateControl runat="server" ControlId="TopNavigationDataSource">
<Template_Controls>
<asp:SiteMapDataSource
  ShowStartingNode="False"
  SiteMapProvider="SPNavigationProvider"
  id="topSiteMap"
  runat="server"
  StartingNodeUrl="sid:1002"/>
</Template_Controls>
</SharePoint:DelegateControl>
```

The Partner Portal application replaces the **SiteMapDataSource** control with another instance of the same control, but it sets different properties. The following code shows how the Partner Portal application sets the control's properties.

C#

```
<Control Id="TopNavigationDataSource"
  Sequence="20"
  ControlClass="System.Web.UI.WebControls.SiteMapDataSource"
  ControlAssembly="System.Web, Culture=neutral, Version=2.0.0.0,
PublicKeyToken=b03f5f7f11d50a3a">
  <Property Name="ShowStartingNode">false</Property>
  <Property Name="StartingNodeUrl"></Property>
  <Property Name="SiteMapProvider">HierarchicalConfigSiteMapProvider</Property>
  <Property Name="ID">topSiteMap</Property>
</Control>
```

You should try to take advantage of all the available delegate controls that are in existing master pages before you decide to create a custom master page. You should also consider placing delegate controls in custom master pages if the implementation may change from one site to another site. This makes the custom master page reusable.

Note:

For more information about how to use delegate controls in SharePoint, see [Delegate Control \(Control Templating\)](#) on MSDN.

Understanding Publishing and Content Deployment

This topic explains how to deploy content from an authoring farm to a production farm. The Web Content Management features in Microsoft Office SharePoint Server contain publishing features for creating and managing authored content. Authored content provides the text and graphics that make up the content of a site. The Microsoft Office SharePoint Server publishing infrastructure helps companies create, version, and approve content that is typically published to a Web site, such as a public Internet site, a knowledge base, or an extranet portal.

It is easy to confuse publishing with content deployment. They are often spoken of together, but they are actually separate SharePoint capabilities. To better understand content deployment, you should familiarize yourself with the concepts that are explained in [Plan content deployment](#) on TechNet.

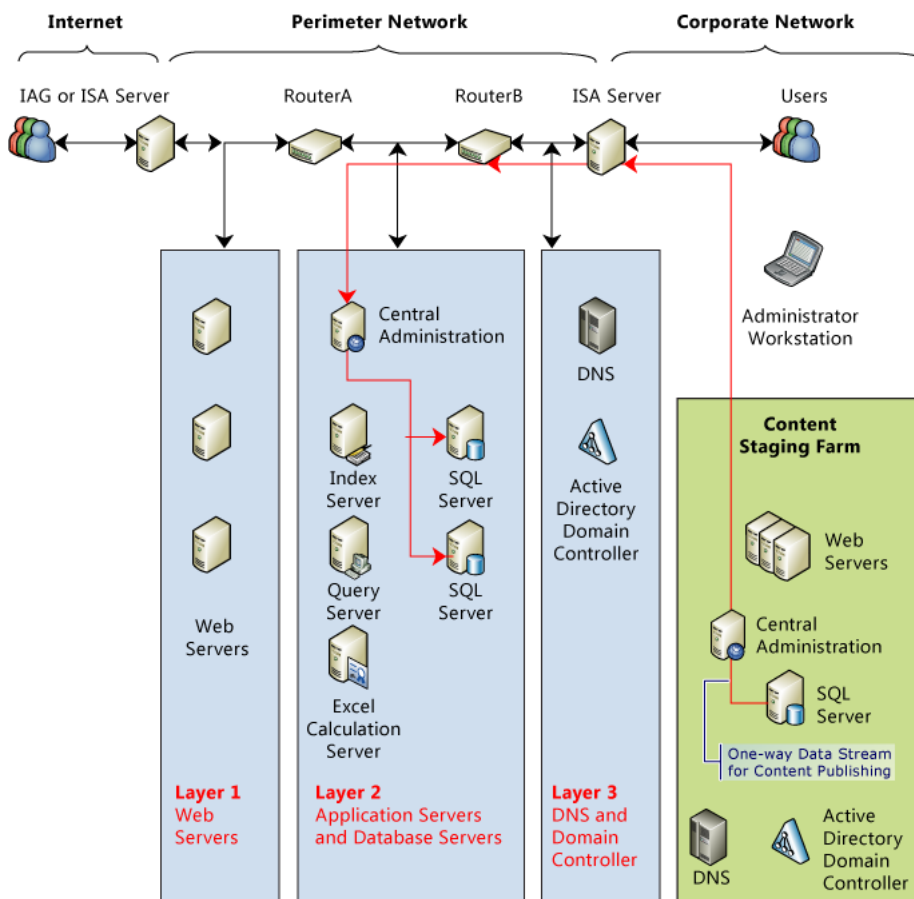
Content deployment is the ability to copy content from one site collection to another. It relies on the content migration APIs that are exposed by Windows SharePoint Services (WSS). For more information, see [Microsoft.SharePoint.Deployment Namespace](#) on MSDN.

The authoring farm is the server farm that hosts content. This is where you use the Microsoft Office SharePoint Server publishing capabilities to create, review, and approve content. The authoring farm exists behind the corporate firewall and it has read/write permissions. After the content is ready, it is published to the production server farm. The production farm is typically located in the *perimeter network* (also known as demilitarized zone, DMZ, or screened subnet) for Internet-facing sites. The production site has read-only permissions. The perimeter network and the intranet are separated by a fire wall such as an Internet Security and Acceleration (ISA) server.

For the production infrastructure, the Partner Portal application uses the back-to-back perimeter network with content publishing topology. For information about this topology, see [Design extranet farm topology](#) on TechNet.

The following diagram shows this topology.

A topology for publishing content to the Internet



You should plan your content deployment strategy before you begin to implement it. For information about how to do this, including a downloadable worksheet, see [Plan content deployment](#) on TechNet.

Note:

The following are now deployed: workflow state, workflows that are designed with SharePoint Designer, items in the recycle bin, audit trails, and alerts.

Content deployment relies on internal SharePoint change logs that are located in both the source and destination site collections. When a content deployment job runs, it creates and saves a change token in both the source and destination environments. The next time the content deployment job runs, it checks these tokens against the tokens that were saved from the previous run. If the tokens do not match, SharePoint raises errors but still tries to redeploy the content. This commonly results in errors in subsequent content deployment job runs. In other words, changes to the destination site collection can cause deployment jobs to fail. To avoid these issues, treat

destination site collections as read-only sites and consider restricting permissions on the destination site collection so that users cannot make changes to the production site content.

The source and destination site collections must exist in different SharePoint Web applications. This is because all of the GUIDs that define sites, Web pages, lists, and list items are preserved and transferred with the site during deployment. Therefore, you cannot deploy one site collection to the same content database as the source site collection.

Deploying Content to the Production Farm

This topic describes the resources and steps that are required to deploy content from an authoring farm to a production farm. Setting up and running content deployment can be complex, and only the basic steps are included here. For greater detail, see [Administer content deployment](#) on TechNet and an excellent [blog series](#) by Stefan Gobner.

To prepare the destination site collection

1. Create an empty site collection in the production farm. This is where the content will be deployed. An empty site template differs from a blank site template in that it contains no content, libraries, or activated features. A blank site contains some content and has some features that are activated. The only way to create an empty site collection is with the following STSADM command:

STSADM.EXE -o createsite -url <url-to-site-collection> -ownerlogin domain\user -owneremail <email-address>

2. Add and deploy all the SharePoint Solution Packages (WSP) to the production farm. Typically, deploying the WSP should install all features to the production farm. However, if you are not using WSPs or if you have an exceptional case, ensure that all features are installed but not activated.
3. Make any necessary file system changes. For example, modify the Web.config files and install or place in the global assembly cache any .NET Framework assemblies that were not included in the WSP deployment.

Do not activate any of the features in the empty site collection. The content deployment process handles feature activation.

After you create the production site collections, you can configure the import and export settings for incoming (production) and outgoing (authoring) content. For information about how to configure the authoring and production farms for content deployment, see [Configure content deployment settings](#) on TechNet.

After you configure the content deployment settings, you must create the deployment paths and jobs. For more information, see [Manage content deployment paths and jobs](#) on TechNet. (The Partner Portal application uses the default settings.) After you create the path and the job, you can deploy content from the authoring farm to the production farm. To run the deployment job manually, click **Run Now** on the deployment job's shortcut menu. To see the updated status, refresh the page. After you do this, the status should display **succeeded**.

Guidelines and Recommendations for Content Deployment

This section includes some guidelines and recommendations that you should consider before you begin to deploy content to the production farm. Guidelines are general statements, and recommendations are more specific applications of the guideline. The following describes some guidelines and recommendations for those guidelines:

- **Guideline:** Never directly modify the destination site collection's content database, either before or after deployment. Consider the following recommendations:
 - **Recommendation:** Do not use the blank site template or any other site template to create a site collection in the destination farm.
The blank site template is not empty. A non-empty template populates the content database. This is contrary to the guideline. You should use STSADM to create a site collection and not specify a site template. This is also named an empty site template.
 - **Recommendation:** Do not directly activate any features in the destination site collection.
The content deployment process activates features that are already activated in the authoring farm. If you directly activate features in the destination site collection, there will be duplicate activations. Directly activating features can also add content to the content database. This is contrary to the guideline.
 - **Recommendation:** Do not add Web Parts to a Web Part zone during feature activation if the feature is going to be activated when the content is deployed. More generally, do not add or change data to the content database during feature activation.
If you add Web Parts to a Web Part zone during feature activation, the Web Parts might be duplicated in the destination site collection. This is because the Web Part will be added twice, during content deployment when the content is copied from the authoring farm to the production farm) and during feature activation.
- **Guideline:** Make all the required changes to the file system before content deployment. Consider the following recommendations:
 - **Recommendation:** Deploy all the required WSPs to the destination Web application before you run the content deployment job.
The WSP deployment adds the required files to the 12 hives of all the Web front-end servers and installs the included assemblies in the global assembly cache.
 - **Recommendation:** Make all the required changes to the Web applications IIS folder \Inetpub\wwwroot\wss\VirtualDirectories\yourport in all Web front-end servers before you run the content deployment job.
For example, if you need to modify the Web.config file, do it before you run the content deployment job.
 - **Recommendation:** Make sure that all the required assemblies are installed in the global assembly cache of all the Web front-end servers before you run the content deployment job.

The WSP might not include all the required assemblies. In this situation, you must install the additional assemblies in the global assembly cache before you run the content deployment job.

Techniques for Securing Published Information

You should secure published information so that only the appropriate users can see it. For example, in the Partner Portal application, promotion pages must be secured so that only certain partners can see them. To accomplish this, an access control list (ACL) is attached to each promotion page. The general process for creating and securing promotions is as follows:

- There is a dedicated site collection for promotions. This means that the content in this site collection can be deployed independently of any other content.
- The root Web of the Promotions site collection hosts the pages library for all promotion pages.
- Each promotion page is attached to an ACL. The ACL gives access to specific partners. A promotion page is assigned an ACL in the authoring farm after the page is authored.
- The ACL, together with the promotion page, is deployed from the authoring farm to the production farm by the content deployment job.

This approach is straightforward because there is only one site collection, one site, and one page library to manage. It is flexible because any promotion page can be assigned to any partner.

If the number of pages becomes large, it can be difficult to manage the individual ACLs. For example, if you want to change the ACLs for all the pages, you may need to do it one page at a time. You should also consider the impact on performance. The Partner Portal application did not have any performance problems with up to 2,000 promotion pages. However, more pages or a more complex use of ACLs might present performance issues.

One way to solve the manageability and performance issues is to have more than one pages library and to group the promotions according to some criteria, such as the discounts they offer. The ACL is attached at the page library level. All the pages in the pages library then have the same ACL. For example, you could set up three page libraries, such as the Gold library, the Silver library, and the Bronze library. You then assign partners to the appropriate group. This means that they only see the promotions that are meant for them. The drawback to this approach is that you lose some flexibility.

Content Deployment with the Partner Portal Application

You can use the Partner Portal application as a way to familiarize yourself with the content deployment process. This requires two stand-alone SharePoint servers. One server is for the authoring farm and should be located in the intranet. The other server is for the production farm and should be located in the extranet.

To set up the authoring server, run the ContosoSetup.bat file that is located in the Setup\PartnerPortal folder. To set up the production server, you must first make the following change to the 00_Parameters.bat file that is located in the Setup\PartnerPortal\SupportingFiles folder before you run the ContosoSetup.bat file.

Change the following code:

```
Set ContentDeployment=Authoring
```

Change the preceding code to the following:

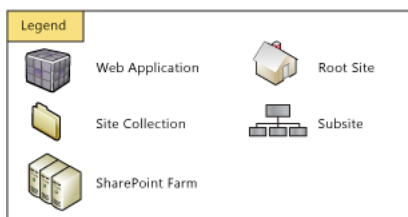
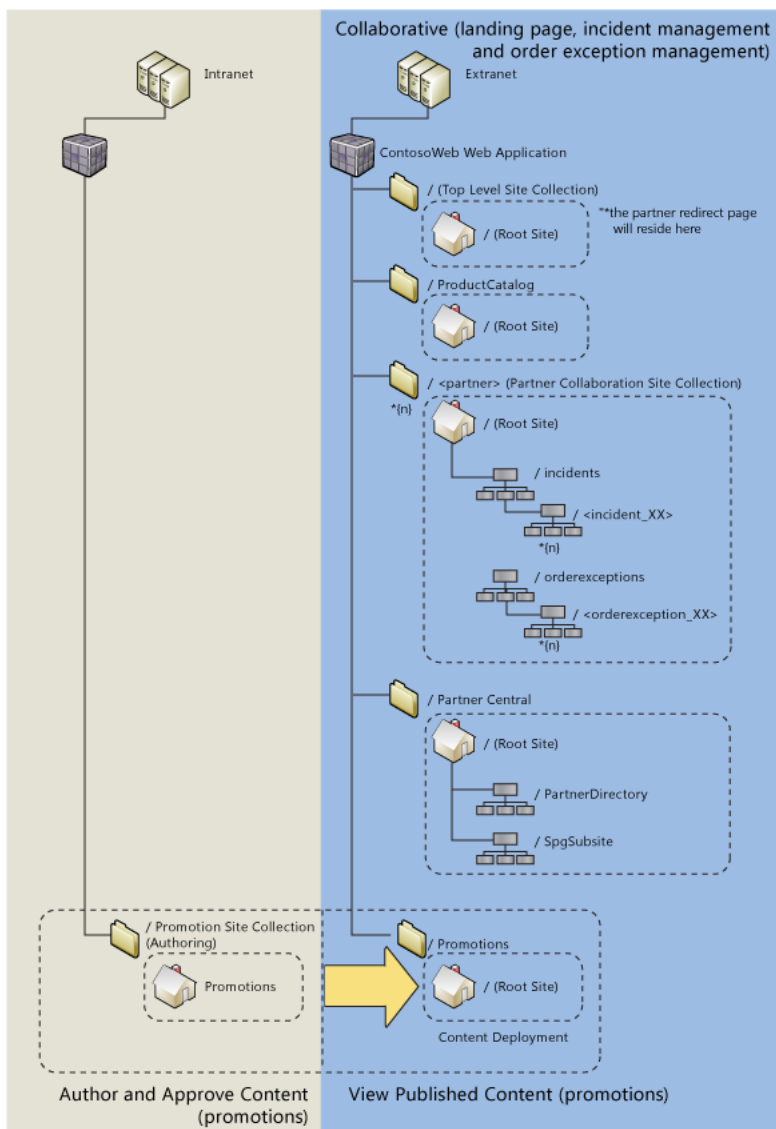
```
Set ContentDeployment=Targeting
```

The 05A_CreatePublishingPortal.bat file that is located in the Setup\PartnerPortal\SupportingFiles folder specifies a site template for the authoring server. This creates the Promotions site collection and activates the features. There is no template specified for the destination server. This is in accordance with the guidelines that are discussed in [Guidelines and Recommendations for Content Deployment](#).

After you set up the authoring and destination servers, you can create the promotion pages on the authoring server and then deploy the promotions site collection from the authoring server to the production server. After you run the deployment job, verify that the features on the production server's Promotions site collection are activated and that the site collection is populated with the promotion pages.

The following illustration shows the structure of the Partner Portal authoring and productions servers.

Structure of Partner Portal servers



More Information

The following articles discuss content deployment in more depth:

- [Content Deployment](#) on [TechNet](#).
- [Content Deployment](#) on the [Microsoft SharePoint Team Blog](#).
- [Content Deployment – Step by Step Tutorial](#) on [Jack Bodine: SharePoint Blog](#).
- [White Paper: End-to-End Content Deployment Walkthrough](#) on [TechNet](#).
- [Content Deployment in WCM](#) on [Sharing...SharePoint](#).
- [Best Practices for publishing portals](#) on [TechNet](#).
- [Deep Dive into the SharePoint Content Deployment and Migration API](#) blog series by Stephan Gobner

Using the Central Template Gallery

The Partner Portal application uses a site template when it creates an incident site in a partner's site collection. Each partner's site collection must access the incident site template to create the site. Site templates that must be globally accessible are installed in the central template gallery (this is sometimes named the global template gallery). The template title must be unique within the central template gallery. The only way to add a template to the gallery is with the STSADM [addtemplate](#) command or with the SharePoint object model. For more information, see [Custom Site Templates](#) on MSDN.

The following STSADM example shows how to add an incident site template to the central site template gallery:

```
stsadm -o addtemplate -filename "incidentsubsite.stp" -title "SPGSubsiteTemplate" -description "SPG Sub Site Template"
```

To modify the template in the central site template gallery, delete the original template and replace it with the updated version. If you want to retain the original template and also add an updated template, you must provide a unique title and file name for the updated template when you add the new version.

The following STSADM example shows how to add a second version of the original template as a new template in the central site template gallery:

```
stsadm -o addtemplate -filename "incidentsubsiteV2.stp" -title "SPGSubsiteTemplateV2" -description "SPG Sub Site Template V2"
```

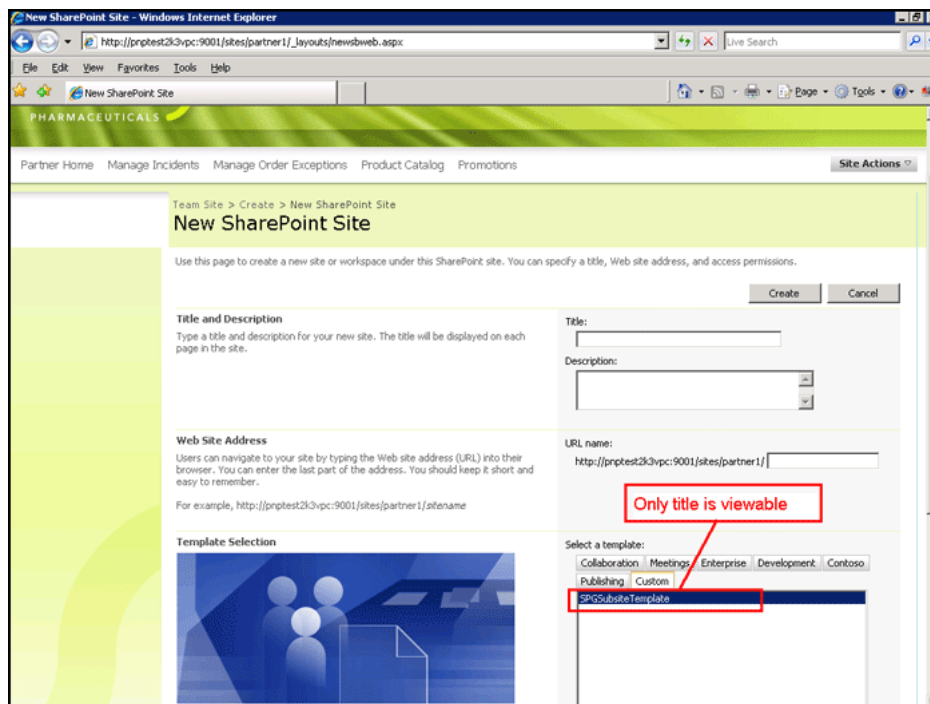
This example adds the updated version to the gallery as a new template. Notice that both the name and title differ from the original template. This approach leaves both the original and new template available for use.

Viewing the Gallery Contents

There are two ways to view the template gallery to discover the global site templates that are available. You can either use the browser or you can use the STSADM [Enumtemplates](#) command.

To use the browser, navigate to the root of your site collection. Click the **Site Actions** drop-down box, and then click **Create Site** in the drop-down list. In the template selection area, a **Custom** tab will be visible if there are any custom templates installed in the gallery. Click **Custom** to display the list of available templates. Only the titles of the templates appear in the list. The following illustration shows an example.

Template selection area



The **Enumtemplates** command can also be used to list the templates to the gallery. The following STSADM example shows how to do this:

```
stsadm -o enumtemplates
```

The following is a sample result:

```
SPGSubsiteTemplate - Language: 1033 - Site Template: _GLOBAL_#0 - Template Id: 1  
SPGSubsiteTemplateV2 - Language: 1033 - Site Template: _GLOBAL_#1 - Template Id: 1  
MyOtherSiteTemp - Language: 1033 - Site Template: _GLOBAL_#2 - Template Id: 1
```

The **Enumtemplates** command provides more information than the browser. It includes the language, where 1033 is American English, a global identifier that SharePoint uses, and the template ID.

If you use the **Addtemplate** command, you must update any references in your code to point to the updated template with the new name. Microsoft Office SharePoint Services does not recognize the updated template as a replacement for the original template.

An alternative to renaming the updated template is to first delete the original template and then add the updated template. The following STSADM example shows how to do this.

```
stsadm -o deletetemplate -title SPGSubsiteTemplate
stsadm -o addtemplate -filename "incidentsubsite.stp" -title "SPGSubsiteTemplate" -description "SPG Sub Site Template"
```

The first STSADM command deletes the site template with the title SPGSubsiteTemplate. The second command adds the updated version of the template. Because the file name and title are the same as the original template, there is no need to update the references to the template.

Mapping a Business Event to a Site Template or Site Definition

The Partner Portal application uses a service that automates the creation of collaboration sites. The service is triggered by two business events. One event is when there is a support incident, and the other event is when there is an order exception. There is a mapping between the business event and a site template. You can use the browser to change the mapping between a site and a template. The following procedure describes how to do this.

To map a site to a global template

1. Navigate to the root page of the site collection.
2. Under **Lists**, select the site template mapping list.
3. Select the list item whose template you want to change, and then click **Edit Item**.
4. Enter the name of the new site template in the **Site Template** box.
5. Click **OK**.

As an example, the following procedure describes how to change the site template that is used when an incident management site is created. This example assumes that you have an updated template to replace the original template.

To change the site template

1. Go to Partner Central, click **Sites** on the **Quick Launch**, and then click **SPGSubsite**.
2. On the **Quick Launch** menu, click **Business Event Type Configuration**.
3. Click **Incident**.
4. Click **Edit Item**.
5. Enter the name of the new template in the **Site Template** box.

More Information

For information about how an incident subsite is created, see [Workflow-Driven Site Creation](#).

For information about custom site templates and the template gallery, see [Custom Site Templates](#) on MSDN.

Extracting Artifacts Created with SharePoint Designer and the Browser

You can use either SharePoint Designer or the browser to create and customize SharePoint Web sites. These authored artifacts are stored in the content database. If you choose, you can extract them from the content database to file representations and manage them as solution artifacts in version control. This topic explains how to extract the following artifacts:

- Site columns and content type definitions
- Files and modules
- List workflow associations

There are existing tools and techniques for site columns, content type definitions, files, and modules. For more information, see the following:

- [AC's WCM Custom Commands for STSADM.EXE](#) on Andrew Connell's [blog](#).
- [How to extract aspx files out of a SharePoint content database](#) on Michael Washam's [blog](#).

You can extract list workflow associations with Windows PowerShell.

Extracting List Workflow Associations from the Content Database

You can determine list workflow associations by traversing the SharePoint object model. The following procedure describes how to do this.

To determine list workflow associations

1. Determine the URL of the site collection.
2. Construct an **SPSite** object.
3. Iterate through each Web site in the site collection with the **SPSite.AllWebs** method.
4. Iterate through each Web site's lists with the **SPWeb.Lists** method.
5. Iterate through each list's workflow associations with the **SPList.WorkflowAssociations** method.

The following Windows PowerShell script iterates through all workflow associations and displays the workflow association's **SoapXml** property.

```
if ($args.Count -eq 0)
{
    write-host "siteurl parameter is missing. Please provide the Url of a site collection as a parameter" -foregroundcolor red -backgroundcolor black
    exit
}

Function Show-WorkflowAssociation($workflowassociation) {
    write-host "Web: " -nonewline; write-host $_.ParentWeb.Url
    write-host "List: " -nonewline; write-host $_.ParentList.Title;
    write-host "Soap Xml:"; write-host $_.SoapXml
}

[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SharePoint")

$mysite=new-object Microsoft.SharePoint.SPSite($args[0])

$mysite.Allwebs | foreach { $_.Lists | foreach { $_.WorkflowAssociations | foreach {
Show-WorkflowAssociation($_) } } }
```

You can adapt this technique to extract other values that are exposed through the SharePoint object model.

Considerations for Enterprise-Scale Applications

This section addresses some common scenarios in enterprise-scale application development. This class of application tends to have requirements for high availability and for handling many transactions per second. These applications also have a relatively long lifetime in the enterprise, and they can be complicated to configure.

Maintainability is an important aspect of enterprise applications. They must periodically be modified or upgraded over their multiyear life spans. To support extensions and refactoring, developers must make these implementations highly testable.

Enterprise applications occasionally require troubleshooting. Developers must anticipate this need by making it easy to diagnose problems that may arise.

Considerations for enterprise-scale applications include the following topics:

- [Managing Application Configuration](#). This topic covers the various application configuration options available to SharePoint developers. Because enterprise applications usually involve non-trivial systems integration and are built for extensibility, configuration is a major concern. SharePoint itself is very flexible, and it is important that applications that are built on top of it maintain that flexibility.
- [Providing Application Diagnostics](#). This topic addresses some techniques and tools for creating code that is easy to instrument and troubleshoot.
- [Improving Application Quality Through Testing](#). This topic provides a comprehensive overview of the types of testing needed for enterprise applications.
- [Building for Scale](#). In order for an enterprise application to scale appropriately, the underlying code must be written in a scalable way, and sufficient stress testing needs to be performed to evaluate its behavior under normal and excessive user load. This topic covers some of the decisions and techniques needed for creating scalable SharePoint code. It also describes scale and stress testing for SharePoint applications.

Managing Application Configuration

Configuration data is data that controls the program's logic instead of data that controls the business or user logic. An example of configuration data is a connection string to a database. Storage options that SharePoint provides for other types of data are also available for configuration data. The following tables outline the available storage options for configuration data.

Storage mechanism	Web.config
Advantages	APIs such as the ConfigurationSection class and SPWebConfigModification class are already designed to hold modifications that are made to the Web.config file. There are some configurations such as HTTPModules, FBA membership and Role providers that have to go in web.config
Disadvantages	Additional work required to deploy changes across a server farm.
Tools	None.
Security model	It requires administrator privileges.
Issues with serialization	The configuration data must be serializable.
Hierarchical (can store at different levels)	No. Only at the Web application level.

Storage mechanism	Hierarchical object store (HOS) (SPPersistedObject)
Advantages	Classes that derive from SPPersistedObject can persist strongly typed data.
Disadvantages	There is no UI. You must derive from SPPersistedObject for each entity to be stored and manage these entities. For example, If you need an entity for each SPWeb in your application, you will need to potentially maintain a tree of SPPersistedObject entities in parallel with the tree of SPWebs .
Tools	See the "Manage Hierarchical Object Store" feature at http://www.codeplex.com/features . This feature only stores values in the Web application. You can build a hierarchy of persisted objects but these objects don't necessarily map to SPSites and SPWebs .
Security model	Users need administrator privileges to access the related SharePoint object (such as SPFarm or SPWebApplication).
Issues with serialization	The objects are strongly typed. This means that there is less concern about accidentally persisting an unserializable object.
Hierarchical (can store at different levels)	Yes, but only SPFarm and SPWebApplication are SPPersistedObjects ; therefore, they can be parents to other SPPersistedObjects . Other persisted objects will need to build off trees rooted to SPFarm or SPWebApplication .

Storage mechanism	Property bag
Advantages	It is probably the most light-weight and easy to use of all the configuration storage options. You can create a _layouts page for your own settings, which has the benefit of making your solution feel much more integrated.
Disadvantages	There is no standard user interface available.
Tools	See the SharePoint Property Bags Settings tool at http://www.codeplex.com/pbs .
Security model	Users need the appropriate access to the related SharePoint object (such as the SPFarm , SPWebApplication , or the SPWeb). You can only write into SPWebApplication property bags when your code runs under the security context of the central administration application pool or farm administrator (an account that has enough privileges to the config database). However, you can read from such property bags regardless of its application pool security context.
Issues with serialization	Persisting non-serializable objects can cause corrupted data.
Hierarchical (can store at different levels)	Yes. This option provides storage at all levels, from a server farm to a list.

Storage mechanism	Lists

Advantages	There is a SharePoint UI for editing lists. Optionally, you can apply permissions to specific items in the list. It already has workflows and event handlers. These allow you to program actions that should occur when the configuration changes. Also, lists have auditing and alerts when settings change.
Disadvantages	This option is not advisable for farm-level or Web application-level configuration data. This is because it can require cross-site list queries. It is also inappropriate for sensitive data that end users should not see or modify, such as connection strings. You can secure read-access permissions to the list, but this complicates the ability to query the list programmatically.
Tools	See the SharePoint Config Store tool at http://www.codeplex.com/SPConfigStore .
Security model	Access depends on the access control lists on the list.
Issues with serialization	No problem if the data can be persisted in the list.
Hierarchical (can store at different levels)	Yes. This option can be used at the farm, site collection, and Web levels.

Managing Web.config Modifications

There are two common techniques for adding configuration data to the Web.config file:

- You can use the declarative approach and add sections of XML code to the file.
- You can use the **SPWebConfigModification** class to make the modifications programmatically.

The next sections describe each of these approaches.

Using the Declarative Approach

The declarative approach is best suited to situations where you need to add large amounts of XML code. In addition, all the configuration changes are applied to every application on the server. Any file that includes extensions to the Web.config file must begin with the webconfig prefix. The file must be deployed to the 12\Config directory.

The following is a sample file that adds a **RuntimeFilter** node as a child of configuration/SharePoint.

XML

```
<actions>
  <add path="configuration/SharePoint">
    <RuntimeFilter
      Assembly="Company.Product, Version=1.0.1000.0,
        Culture=neutral, PublicKeyToken=1111111111"
      Class="MyRuntimeFilter",
      BuilderUrl="MyBuilderUrl"/>
    </add>
  </actions>
```

The file describes where the extensions belong in the Web.config file and the actions that add the new nodes with the new configuration data.

To merge these changes with the Web.config file, you can either use the stsadm **copyappbincontent** command or you can use a Web service method **ApplyApplicationContentToLocalServer()**.

This method locates the current Web service and merges the extensions into the Web.config file. Potentially, the **ApplyApplicationContentToLocalServer** method merges all configuration changes each time it is called. For this reason, the **id** attribute in the extension XML file must include a unique GUID so that existing changes are not merged a second time.

Calling the **ApplyApplicationContentToLocalServer** Web method or the stsadm command **copyappbincontent** command only merges the modifications on the Web front end (WFE) server where the method or command is called. It does not propagate these changes throughout the farm. You must use the **copyappbincontent** command on each WFE where you want the configuration data to be merged with the Web.config file. You must also use the **copyappbincontent** command each time a new WFE is added to the farm.

Using the SPWebConfigModification Class

The **SPWebConfigModification** class provides more control than the declarative approach. You can modify any node or attribute in the Web.config file. This means that the modifications can be as granular as you require. In addition, you can also target the modifications to a specific Web application instead of applying them to every application on the server.

In general, to use the **SPWebConfigModification** class, you should do the following:

1. For each configuration change, add an **SPWebConfigModification** instance to the **WebConfigModifications** collection of the current **WebApplication** object.
2. Invoke the **Update** method of the current **WebApplication** object.
3. Invoke the **ApplyWebConfigModifications** method of the current **WebApplication** object's Web service.

An example of using the **SPWebConfigModification** class is in the Contoso.LOB.Services.Client project, in the WebAppFeatureReceiver.cs file. The following code demonstrates how to programmatically add XML code to a Web.config file.

C#

```
[SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
public override void FeatureActivated(SPFeatureReceiverProperties properties)
{
    //...

    SPWebApplication webApp = (SPWebApplication)properties.Feature.Parent;

    webApp.WebConfigModifications.Add(GetModification());

    webApp.Update();
    webApp.WebService.ApplyWebConfigModifications();

    //...
}
```

The **FeatureActivated** method adds the instance of **SPWebConfigModification** returned by **GetModification()** to the **WebConfigModifications** collection. The **Update** method applies the modifications to the SharePoint

database. The **ApplyWebConfigModifications** method schedules a timer job to deploy the changes throughout the server farm.

The **GetModification** method creates an instance of **SPWebConfigModification**, setting the name, XPath, owner, type, and value properties.

C#

```
private SPWebConfigModification GetModification()
{
    return new SPWebConfigModification("system.serviceModel", "configuration")
    {
        Owner = GetOwner(),
        Type = SPWebConfigModification.SPWebConfigModificationType.EnsureChildNode,
        Value = GetWcfServiceConfiguration()
    };
}
```

If you use the **SPWebConfigModification** method, modifications are added to the SharePoint database. A service on the farm named **SPWebService.ContentService.ApplyWebConfigModifications** applies the Web.config modifications to every WFE. (There is no corresponding functionality for deploying updates to layouts.sitemap.)

Managing SharePoint Configuration

This section describes how configuration information is stored in SharePoint.

Hierarchical Object Store

The hierarchical object store is the mechanism by which Windows SharePoint Services stores information about the farm, Web applications, features, and so on. You can use it to store custom configuration data by creating classes that inherit from the **SPPersistedObject** class. This option is appropriate if you need to store complex objects. The hierarchical object store allows you to store data at either the farm or Web application level. Additionally, you can create a custom hierarchy of **SPPersistedObjects** rooted by an instance of **SPFarm** or **SPWebApplication**.

Property Bags

A property bag enables you to add and retrieve properties to objects. Property bags are implemented as a simple hash table of property names and values. Do not attempt to store anything in property bags unless you are sure it can be serialized.

At the site level, a property bag can be accessed with the **Properties** property of an instance of the **SPWeb** class. If you want to store list-level configuration data, use the property bag of the root folder of the list.

You can use a property bag to store configuration information at different levels of the SharePoint hierarchy. The following table summarizes the options.

Level	Storage location
SPFarm	The Properties property that is inherited from the SPPersistedObject class.
SPWebApplication	The Properties property that is inherited from the SPPersistedObject class.
SPSite	The AllProperties property of the root Web.
SPWeb	The AllProperties property.
SPList	The RootFolder.Properties property.

Make sure to use prefixes with your keys to help you distinguish your custom configuration properties from other properties. In particular, be careful not to remove properties that are added by SharePoint. You must call the **Update** method when you add, change, or remove a property bag entry.

Lists

Use lists if you want to store configuration data that end users can edit. Otherwise, consider using property bags. Lists can also be used as a way of storing configuration data at different levels of the SharePoint hierarchy. The following table summarizes the options.

Level	Storage location
SPFarm	List in Central Administration site or subsite.
SPSite	List in the root Web of the site collection.
SPWeb	List in the SPWeb

More Information

To learn more about using the Web.config file to store configuration data, see [Working with Web.config Files](#) on MSDN.

There is a tool named Web.config Modification Manager for SharePoint that you can use to add, edit, and delete modifications to the Web.config file made through the SPWebConfigModification API. This tool was created by Vincent Rothwell and is available on his blog.

Providing Application Diagnostics

Developers and architects should have a strategy for monitoring their applications, recording any problems, and providing enough information to help diagnose those problems. Including logic in an application that exposes its state, both while it is under development and after it is released, is referred to as *instrumentation*. Microsoft Windows provides several mechanisms for instrumentation, including event logs, trace files, Windows Management Instrumentation (WMI), and performance counters.

Logging and tracing provide many advantages. For example, you can anticipate potential system failures by detecting approaching boundary limits such as a database that is becoming full. If an application does fail, well-planned, informative error messages can reduce the time needed to restore operations.

This guidance provides several examples of instrumentation, including exception management, event logging, and tracing. Logging and tracing are packaged into a reusable component that is part of the SharePoint Guidance Library. This means that you can incorporate these capabilities into your own application. For more information, see [The SharePoint Logger](#).

Performance counters and WMI are more advanced techniques for representing run-time state. They were omitted from this guidance to reduce the complexity of the reference implementation. After you understand how the Partner Portal application uses exception handling, tracing, and logging, you can consider adding these other forms of instrumentation. The basics of instrumenting a SharePoint application are similar to the techniques that are used with other Microsoft .NET Framework applications. For an overview of instrumentation, see [Tracing and Instrumenting Applications](#) on MSDN.

The code that implements logging and the code that implements tracing often use the same infrastructure. However, event logs have a different audience and intent than trace logs. Tracing helps developers detect and analyze problems when they cannot easily access some aspect of an application and debug it directly. Logging helps information technology (IT) professionals detect, troubleshoot, and resolve problems in deployed applications.

Using Event and Trace Logs in SharePoint

Tracing

The following are some examples of information to trace:

- **Errors and exceptions.** If an exception occurs in your application, you want to know about it. Even if your code handles the exception, it is often useful to record the event with a trace statement. For example, a download can initially fail but succeed on the second attempt. The user may not need to know this information but the developer might find it useful.
- **Remote system calls.** In a service-oriented world, it is common for an application to access functionality on a different system. Logging the request and response messages can be very helpful if you must debug the application. Many frameworks, such as Windows Communication Foundation (WCF), allow you to trace this information.
- **Queries.** If your application creates and executes queries, knowing the text and the parameters of these queries can be valuable debugging information. Queries are not limited to SQL queries against a database. Collaborative Application Markup Language (CAML), Lightweight Directory Access Protocol (LDAP), and Entity SQL are three of many other possibilities. It can be helpful to trace any querying syntax that your application uses.

Typically, tracing information is written to a local file. Although trace information is valuable, the amount of trace information can overwhelm both the developer and the resources of the system that records the information. For this reason, tracing levels, which correspond to degrees of severity, are typically adjustable at run time.

In addition to diagnosing run-time problems, trace logs are often used to detect hidden problems. For example, SharePoint writes to a trace file every time it believes that an object has not been properly disposed. This can occur when it finds objects that contain a reference to an **SPRequest** object (such as **SPWeb** and **SPSite**) and were not disposed as expected. The SharePoint tracing infrastructure is named the Unified Logging Service (it is sometimes also named the Universal Logging Service) or ULS.

Developers can program their applications to write custom operations messages to the ULS. Writing to the same trace log that Windows SharePoint Services uses enables you to view your custom application traces in the larger context of Windows SharePoint Services operations without having to correlate multiple trace logs.

The Partner Portal application demonstrates how to log to the ULS and provides a simple logger to help you. Although logging to the ULS requires you to also analyze the surrounding SharePoint traces, these additional traces often help to diagnose the problem.

Note:

You can use the SharePoint Guidance Library's default logger or you can customize this component to write to a logging framework that you provide. Some developers prefer to keep their application tracing separate from the traces generated by SharePoint itself.

SharePoint trace logs can become very complicated, particularly in server farms. The SharePoint Administration Toolkit includes a tool named the SharePoint Diagnostic tool (SPDiag) that helps analyze these trace logs. This tool brings together logs from various sources including ULS, Internet Information Services (IIS), the performance counter, Windows event log, and WMI (Windows Management Instrumentation)—and it presents a unified view of the information. For more information, see [Trace Logs](#) on MSDN and [SharePoint Diagnostics Tool](#) on TechNet. There are also tools on CodePlex for viewing ULS trace data, such as the [SharePoint ULS Log Viewer](#).

Considerations for Tracing

Performance and noise, or irrelevant information, are both potential issues you may encounter when using tracing. Performing detailed tracing on a production system that is under heavy load will impact areas such as disk I/O and CPU utilization. This can degrade the application's performance. If performance becomes a problem, consider putting the logs on a separate drive to offload I/O from the system disk and to prevent inadvertently filling your system partition. You can also consider the option of implementing rolling logs so that the log files never grow too large. SharePoint uses this approach for the ULS, where you can configure the maximum number of log files that are saved to each server in the farm and the duration (in minutes) you want for each log file.

Detailed tracing can add irrelevant information to the trace log files. This affects performance and makes the logs difficult to analyze. More advanced logging systems have trace levels and categories to help control the amount of information that gets logged. Trace levels are applied to categories and control the level of detail that is logged (this is often referred to as *verbosity*). Categories define an area of functionality. For example, you can apply the **verbose** trace level to the **Search Indexing** category. You should raise trace levels to perform diagnostics only on the categories that require investigation, and the levels should be lowered after the diagnostics are complete.

Programmatically Changing Diagnostic Settings

The SharePoint object model has a class named **SPDiagnosticService**. This class enables you to do programmatically what you can do manually through the user interface in the Trace Log and Event Throttling sections on the SharePoint Central Administration's Diagnostics Logging page. You can also programmatically change these trace settings with PowerShell. For more information, see [Redirecting IIS7 and SharePoint 2007 logging in PowerShell](#) in Jason Cahill's [blog](#).

For more information about the diagnostic logging settings, see [Configure diagnostic logging settings \(Windows SharePoint Services\)](#) on TechNet.

SharePoint Guidance Library supports tracing to the ULS. It also allows you to configure the tracing levels that are written to the ULS trace logs based on category and trace severity.

Logging

Logging helps IT professionals detect, diagnose, and troubleshoot problems with deployed applications. This information is often aggregated into a central monitoring system such as the [System Center Operations Manager \(SCOM\)](#).

Logging information should provide enough information so that someone can understand the problem and know what to do about it. A logging message such as "Object reference not set in **MySampleMethod**" can be helpful to a developer for diagnostics and is appropriate for tracing, but it is not useful to an IT professional. In fact, these sorts of messages can be detrimental because they clutter the logs with irrelevant information and make it difficult to find the actual problem. A better message is "Could not connect to database **MySample**: the connection timed out." This message gives enough information for an IT professional to understand where the problem is and how to address it.

The following are some examples of information to log:

- **Predictable application failures.** There are situations where you can expect applications to fail. For example, if an application calls a Web service and there is no response, the call fails. Logging these events can alert the IT professional to potential problems in the network.
- **Installation or configuration issues.** Typically, IT professionals are responsible for installing and configuring applications. If there are problems around these tasks, they need to know what they are.
- **Unknown application failures.** If an application has an unhandled exception, log the event. This information alerts the IT professional to potential problems. An example of such a log message is "Authentication failed for unknown reason." Although the reason is unknown, the message pinpoints the area of the application that failed and potentially indicates the need to bring in a developer to assist with diagnostics.
- **Situations that can potentially cause issues in the future.** Some issues can be anticipated, such as when a resource is about to become exhausted. Examples of possible messages are "The hard drive is becoming full" and "Deadlock detected in query **MyQuery**."

Event Sources for Logging to the Event Log

When you log data to the event log, you must set the **EventSource** parameter. This is the name of the application that can log information to the event log. The default event source for the Partner Portal application is the Office SharePoint Server event source, which is available with Microsoft Office SharePoint Server.

If you want to use an event source other than the SharePoint event source, you must create the event source on all Web front-end servers and application servers before you install and activate the features that are going to log messages. To create an event source, you can either edit the registry or use the [Eventcreate](#) utility. This must be done on each Web front-end server. For information about programmatically creating an event source, see [EventLogInstaller](#) and [CreateEventSource](#) on MSDN.

You must also have the correct privileges to create an event source. You can either be an administrator on the system or you can grant the ASP.NET application pool identity extra privileges to create event sources. By default, the ASP.NET user account cannot create event sources. However, this can also be done by creating a timer job that will, in turn, create the events. Timer jobs run on each Web front-end server. The use of timer jobs is beyond the scope of this guidance. For more information, see [Creating Custom Timer Jobs in Windows SharePoint Services 3.0](#) on MSDN and [Creating Custom SharePoint Timer Jobs](#) by Andrew Connell.

It is recommended that you set the **EventID** parameter in addition to the **EventSource** parameter. This is the ID of the type of event that has occurred. Together, the **EventSource** and **EventID** parameters identify a particular problem with a particular application. Creating a new **EventID** does not change the registry, so it does not require a special privilege level.

The following code shows how to use the logging functionality that is provided by the SharePoint Guidance Library to write an error message that includes the **EventID** parameter.

```
C#
logger.LogToOperations("Could not connect to database: " + databaseName,
    DATABASE_CONNECTION_EVENTID);
```

The logging functionality that is provided by the SharePoint Guidance Library also allows you to route particular levels of events to the event log based on category and event severity.

Storing Logging and Tracing Information

Logging is information that is intended for IT administrators. Typically, logging information is stored in the Windows event log because it is easily accessible with tools such as the System Center Operations Manager (SCOM). Therefore, as a general practice, logging information should be written to this location. Tracing information is typically intended for developers and product support professionals, but it is also used by experienced IT professionals. The ULS writes trace information to a trace log. For the Partner Portal application, tracing is written to the ULS trace logs at the default location of ...\\12\\LOGS.

Neither SharePoint diagnostics nor the implementation of logging that is provided in the SharePoint Guidance Library provides the ability to log to a database. However, in some cases, organizations prefer this approach. One reason is that a centralized database might offer better reporting features. Another reason is that a centralized database allows you to synchronize entries across all the Web servers in the farm. This can provide more complete diagnostics when a problem occurs. A third advantage is that you do not need to perform complex post-processing steps to aggregate the different logs.

A central database incurs higher costs and is more complex than file-based logging. It is possible for the logging activity itself to become a bottleneck in your system. Make sure that the tracing database never becomes full or else implement a stored procedure to control the size of the tracing table. You should carefully consider whether you want to use a database to augment or replace file-based logging. However, when a centralized database solution is used appropriately, it can be helpful.

The following table summarizes the tradeoffs between logging and tracing approaches. The headings have the following meaning in this table:

- **Approach.** This is the logging or tracing approach.
- **In context with other SharePoint events/traces.** This means that the information is present along with trace or event information from SharePoint. Frequently, this can help you identify the root cause of an issue because the application-specific traces are surrounded by SharePoint trace information that may relate to the issue. However, this additional information also adds complexity.
- **Primary audience.** This is the audience that is typically targeted by the approach.

Approach	In context with other SharePoint events/traces	Primary audience
ULS	Yes.	Developer and support specialists, although also used by experienced IT professionals.
Event log	Yes.	IT professionals.
Custom trace log file	No.	Implementation-dependent.
Custom database	No.	Implementation-dependent.
ASP.NET output trace	No.	Developers.

 Note:

Some system administrators may need to use information that the SharePoint system stores in ULS, and there are tools created by the SharePoint community that make ULS viewing and filtering easier. However, in general, it is recommended that in your SharePoint applications you should always record events that require human response in the Windows event log. When diagnosing a problem, it is a common practice for system administrators and developers to start with the event logs and then move to the SharePoint logs and IIS logs if more information is needed. In these cases, managing custom SharePoint applications can require collaboration between developers and system administrators.

Exception Management in SharePoint

Correct exception handling is an essential part of reliable and maintainable SharePoint applications. In general, exception handling for a SharePoint application is not very different from any other .NET Framework application. The general guidelines in the .NET Framework Design Guidelines chapter [Design Guidelines for Exceptions](#) also apply to SharePoint applications.

This section gives general exception handling guidelines and also specific guidance for handling exceptions in SharePoint contexts, such as Web Parts, list item event receivers, and workflow execution. The Partner Portal Reference Implementation demonstrates how to implement some of these techniques.

General Exception Handling Guidelines

This section describes the following guidelines:

- [Catch only exceptions that you handle.](#)
- [Catch the most specific type of exception.](#)
- [Avoid empty catch blocks.](#)
- [Implement a handler for unexpected exceptions.](#)

Catch Only Exceptions That You Handle

In general, you should catch only exceptions that you handle. Handling an exception includes one or more of the following actions:

- **Take an alternative action to compensate for the exception.** In some cases, the exception is a correctable error. For example, if you try to call a Web service and it is too busy to handle the request, you could handle the "service to busy" exception by retrying the invocation a few more times.
- **Log or trace the exception information.** It is sometimes useful to record exception information for later use. For example, if you encounter an unexpected exception, you should log the exception information and let the process fail gracefully. However, if it is not an unexpected exception, and your code can compensate for the exception, it is often helpful for debugging purposes to write the exception information to the trace log.
- **Show an error message.** In some cases, exceptions indicate problems that require action by the end user. For unknown or unexpected exceptions, you should display a generic error message that explains that there is an unknown problem. If you know what problem the exception indicates, you can catch that specific exception type and show an error message that clearly explains the situation to the user.
- **Add additional information to the exception or increase the level of abstraction of the exception.** An exception that is not caught will propagate up the call stack. Methods at a higher level in the call stack may have more information about the task being performed. For example, lower-level methods in a configuration manager might throw a **KeyNotFoundException** if a configuration key could not be found. Within the higher-level methods, this exception information could then be wrapped within a **ConfigurationSettingNotFoundException** instance.

Catch the Most Specific Type of Exception

Always try to catch the most specific type of exception. Do not catch the general **Exception** type. Failure to follow this guideline can result in unexpected behavior in your code and hard-to-find bugs.

```
C#
// Do not catch the general type exception:
// bool isValid;
// try
// {
//int.parse(inputValue);
//isValid = true;
// }
// catch(Exception)
// {
//isValid = false;
// }

// Instead, catch a specific exception: FormatException.
bool isValid;
try
{
int.parse(inputValue);
isValid = true;
}
catch (FormatException)
{
isValid = false;
}
```

Note:

There is one exception to this recommendation: implementing an unhandled exception handler, which is described later in this topic.

Avoid Empty Catch Blocks

You should avoid catching an exception with an empty catch block.

C#

```
// Try to avoid this construct:
// int convertedValue = 0;
// try
// {
//     convertedValue = (int) myValue;
// }
// catch (InvalidCastException)
// {
//     // Do nothing if the cast fails.
// }

// Instead, use this construct.
int convertedValue = 0;
if (myValue is int)
{
    convertedValue = (int) myValue;
}
```

Never catch the general **Exception** type with an empty catch block. This can result in bugs that are very hard to find. For example, consider a try/catch block around a statement that catches the top-level **Exception** type. While executing the statement, another thread wants to abort the current thread. A **ThreadAborted** exception will be raised. However, the catch block will intercept the **ThreadAborted** exception and handle it. This prevents the thread from being aborted.

C#

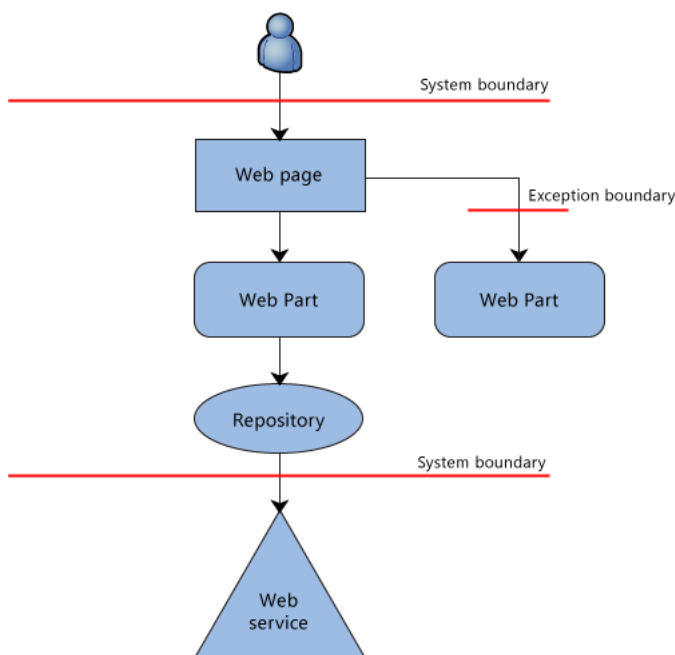
```
// Never do this:
// try
// {
//     // operation that can potentially fail
// }
// catch(Exception)
// {
//     // Do nothing (hide the exception). // }
```

Implement a Handler for Unexpected Exceptions

It is not possible to predict all exceptions that can occur with a system. Therefore, you must sometimes implement handlers at system boundaries for otherwise unhandled exceptions.

The following illustration shows an example of this.

Example of system and exception boundaries



For example, consider a Web page that contains several Web Parts. Suppose that the Web Parts invoke methods of a repository that eventually calls into an external Web service. In this scenario, there are several locations where an unhandled exception handler is appropriate:

- **Page.** In a SharePoint application, if an unhandled exception is not caught, it propagates to the page that was requested by the end user. SharePoint has a built-in global exception handler that will catch any unhandled exceptions and redirect the user to an appropriate error page. Although it is hard to implement a custom global exception handling policy for SharePoint, it is possible to handle unhandled exceptions in custom pages by responding to the **Page_Error** event provided by ASP.NET. The error page should not display any unhandled exception details to the end user because this can pose a security risk.
- **Web service.** Within the Web service, you should catch all unhandled exceptions, log them and return a general exception message. This is also called exception shielding. For more information, see [Exception Shielding](#) on MSDN. A Web service should also not expose any unhandled exception information.
- **Web Part.** By default, if an unhandled exception occurs within a Web Part, the exception propagates to the Web page and triggers the unhandled exception handler of the page. This is not always the desired behavior. For example, it is possible in SharePoint to give end users the option of composing their own pages using Web Parts. When a Web Part throws an exception, it is difficult for the user to remove that Web Part from the page. To avoid this problem, you can implement an unhandled exception handler in this location.

Logging or Tracing Exceptions

Unhandled exceptions should be logged so that system administrators know when a component is not performing as expected. You should augment the log message with additional information that is helpful to the system administrator. The following code shows how to log an exception with additional information for the system administrator. This code uses the logging component of the SharePoint Guidance Library.

```
C#
catch(Exception unhandledException)
{
    ILogger logger = SharePointServiceLocator.Current.GetInstance<ILogger>();
    logger.LogToOperations(unhandledException, "Could not connect to the database");
}
```

In some situations, you should also write exception information to the application's trace log. Trace logs are intended for developers who need to analyze application failures. The following code shows how to write exception information to a trace log using the SharePoint Guidance Library.

```
C#
int retryCount = 0;
while (retryCount < 3)
{
    try
    {
        // Call into a service.
        break; // exit while loop
    }
    catch(ServiceBusyException exception)
    {
        retryCount++;
        ILogger logger = SharePointServiceLocator.Current.GetInstance<ILogger>();
        logger.TraceToDeveloper(exception, "Service was busy. Attempt " +
                                retryCount);

        if (retryCount == 3)
            throw;
    }
}
```

This code tries to execute a Web service. If it succeeds within three tries, no exception is propagated to the caller of this code. However, if the developer needs to find out why the application is slow, it can be useful to know that the application needed to retry Web service invocations.

Handling Exceptions in Web Parts

Normally, unhandled exceptions that occur within a Web Part propagate to the Web page and are resolved by the unhandled exception handler of the page. The handler redirects the user to an error page that contains an informative error message.

As a result, an unhandled exception in a single Web Part can prevent the Web page as a whole from being displayed. This behavior is sometimes desired because an unhandled exception can potentially corrupt data on the page. In other situations, it can be better to catch unhandled exceptions within Web Parts. With this approach, an unhandled exception results in an appropriate error message being displayed by the Web Part that received the exception. All other Web Parts on the page will continue to function normally.

Note:

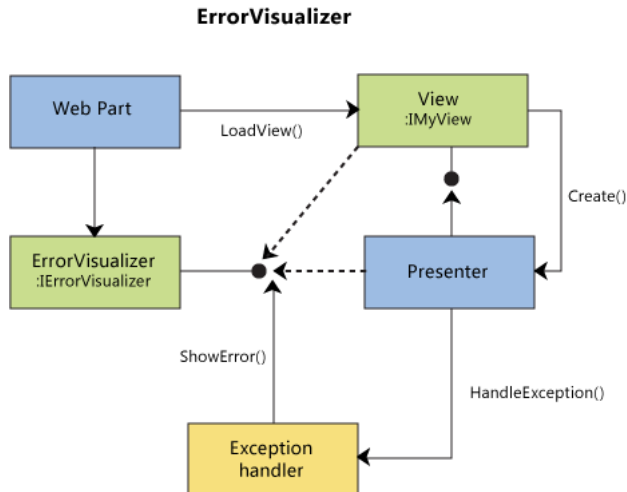
Unfortunately, because of the way the ASP.NET controls are implemented, it is not possible to create a single unhandled exception handler that catches all unhandled exceptions that might occur during the lifetime of a control or its child controls. For this reason, the Partner Portal application implements its unhandled exception handler in the presenter layer. This allows the application to catch most "logic"-related errors.

The Partner Portal Reference implementation demonstrates how to create Web Parts that implement unhandled exception handlers. These handlers log errors and let the user interface display a friendly error message. The goal is to do the following:

- Catch the exception.
- Log the exception to the event log and the trace log.
- Replace the entire user interface (UI) of the Web Part with an informative error message.

The following illustration shows the implementation of this kind of Web Part in the Partner Portal application. The Web Part implements the Model View Presenter (MVP) pattern. For more information, see [The Model-View-Presenter \(MVP\) Pattern](#).

Web Parts that implement unhandled exception handlers



This diagram consists of the following parts:

- **The Web Part.** The Web Part itself is only the host for the view and the presenter objects. The Web Part should contain as little code as possible. Therefore, the chances of anything failing in the code of the Web Part are low.
- **The view.** The view is a user control that displays the UI of the Web Part. To make the view as simple as possible, it is designed only to handle positive code paths. It has the ability to display functional errors, such as validation errors, but it does not contain logic for displaying unhandled exceptions. The view is added directly to the control tree. The view is a child control of the **ErrorVisualizer** control. Unhandled exceptions are intercepted by the **ErrorVisualizer**, and the **ErrorVisualizer** can suppress the UI if an error has occurred.
- **The presenter.** The presenter contains the logic of the Web Part. This is the best location to implement an unhandled exception handler. However, when an unhandled exception occurs, it needs a mechanism for displaying the friendly error message. The view does not have this functionality, so the presenter calls the **ErrorVisualizer** to display errors. Because the steps to handle an unhandled exception are usually the same (log and display an error message), this behavior is encapsulated in the **ViewExceptionHandler** class.
- **The ErrorVisualizer control.** The **ErrorVisualizer** control makes it easier to display technical errors and hide the view. This control has an interface that allows consumers to set an error message on it. When an error message is set, it will display that error message and hide any child controls that might be present. The view is added as a child control of the **ErrorVisualizer**; when an error message is set, the view will no longer be displayed.
- **The ViewExceptionHandler class.** The code that would go in the unhandled exception handler is often the same for each Web Part. To maximize code reuse this code is factored into a **ViewExceptionHandler** class.

Web Parts with unhandled exception handlers occur in several places in the Partner Portal application. The following code comes from the Product Details Web Part. It shows how to create the **ErrorVisualizer** control and how to add the view (**ProductDetailsControl** class) to the **ErrorVisualizer**.

```

protected override void CreateChildControls()
{
    base.CreateChildControls();

    // Create a control that will display any errors that might
    // occur in the ProductDetailsControl.
    ErrorVisualizer errorVisualizer = new ErrorVisualizer();

    this.Controls.Add(errorVisualizer);
    // Add the ProductDetailsControl to the host. This way, if an error has to be
    // rendered, the host can prevent the ProductDetailsControl from being displayed.
    this.productDetailsControl = (ProductDetailsControl)Page.LoadControl(
        "~/_controltemplates/Contoso/ProductDetailsControl.ascx");
    this.productDetailsControl.ErrorVisualizer = errorVisualizer;
    errorVisualizer.Controls.Add(this.productDetailsControl);
    this.productDetailsControl.LoadProduct(this.productSku);
}
  
```

- The following code from the Product Details Presenter shows how the unhandled exception handler was implemented. The call to the **HandleViewException** method logs the error and uses the **ErrorVisualizer**

control to display an informative error message to the user.

```
public void LoadProduct(string sku)
{
    try
    {
        /// ...
    }
    catch(Exception ex)
    {
        // If something goes wrong, make sure the error gets logged
        // and a non-technical message is displayed to the user.
        new ViewExceptionHandler().HandleViewException(ex,
            this.ErrorVisualizer,
            Contoso.PartnerPortal.ProductCatalog.Properties.Resources.ProductDetailsErrorMessage);
    }
}
```

Handling Exceptions in List Item Event Receivers

SharePoint allows you to attach an event receiver object to a list. A list item event receiver is an instance of a class that derives from the **SPItemEventReceiver** class. Methods of the list item event receiver are invoked when an item in the list is added, removed, or modified.

If an unhandled exception occurs during the execution of a list item event receiver method, you should perform the following actions:

- Log the exception.
- Set the **Cancel** property of the **SPItemEventProperties** object passed as an argument to **true**. This cancels the action.

If an unhandled exception occurs while executing a method of a list item event receiver, the action is not canceled by SharePoint, by default. Failing to cancel the action can cause corruption of list data. For example, consider the situation of validation code in the list item event receiver. If the validation fails with an unhandled exception, the list item's pending changes should not be made. Allowing the changes can be a security risk. Also, invalid data can be very hard to remove after it is added to the list.

Note:

There are some situations where it is not appropriate to set the **Cancel** property to **true**.

The following code from Partner Portal application's **IncidentTaskReceiver** class shows how to implement an unhandled exception handler for a list item event receiver:

C#

```
public override void ItemAdding(SPItemEventProperties properties)
{
    try
    {
        incidentManagementRepository.WriteToHistory(
            string.Format(CultureInfo.CurrentCulture, TaskCreatedMessage,
                properties.AfterProperties[TitleField]));
    }
    catch (Exception ex)
    {
        ListExceptionHandler handler = new ListExceptionHandler();
        handler.HandleListItemEventException(ex, properties,
            Resources.HandleListItemEventExceptionValue);
    }
}
```

ListExceptionHandler is a helper class declared in the Partner Portal application's Contoso.Common project. Calling the **HandleListItemEventReceiver** method of this class logs the exception to the EventLog and cancels the event. There are overloads of the available for this method if you do not want to cancel the event.

The **ListExceptionHandler** class includes logic that handles exceptions that occur during exception handling itself.

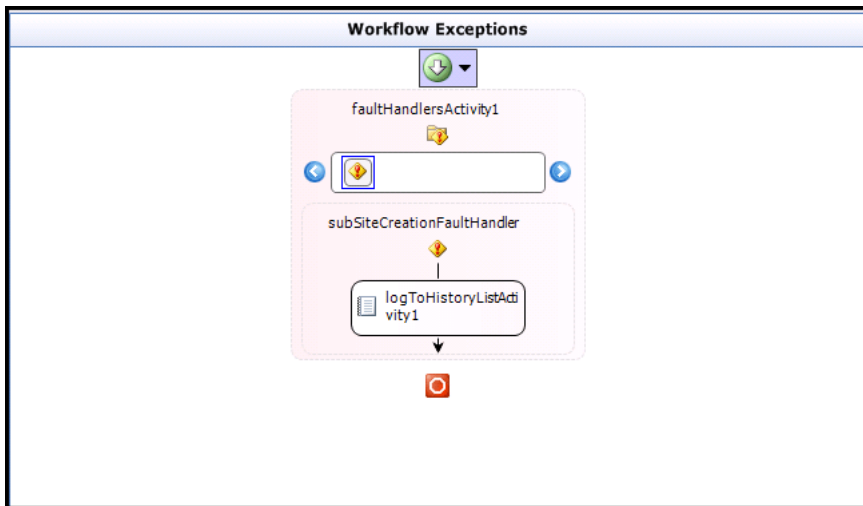
Handling Exceptions During Workflow Execution

In custom workflow code, you should follow the standard guidelines for handling exceptions with the try/catch pattern. Windows Workflow Foundation provides using the **FaultHandlerActivity** activity as an additional way to catch exceptions thrown by workflow activities.

The SharePoint Guidance Library's subsite creation workflow demonstrates how to wrap a **LogToHistoryListActivity** activity in a **FaultHandlerActivity** activity. The fault handler activity handles any exceptions of type **SubSiteCreationException** and logs a message to the workflow history list associated with the workflow.

The following illustration shows the fault handler for the subsite creation workflow.

Fault handler for subsite creation workflow



The following illustration shows how the **FaultHandlerActivity** object is configured using the **FaultType** property to catch exceptions of a specified type.

FaultHandlerActivity configuration

The Properties window for the **subSiteCreationFaultHandler** activity is shown. The title bar reads "Properties". The main area displays the following properties:

subSiteCreationFaultHandler System.Workflow.ComponentModel.FaultHandlerActivity	
Activity	
(Name)	subSiteCreationFaultHandler
Description	
Enabled	True
Misc	
Fault	
FaultType	Microsoft.Practices.SPG.SubSiteCreation.SubSiteCreationException

Below the properties table, there is a section for the (Name) property with the text: "Please specify the identifier of the activity. It has to be unique in the workflow."

For more information about handling exceptions in Windows Workflow, see [Windows Workflow Tutorial: Introduction to Fault Handling](#) on MSDN.

Improving Application Quality Through Testing

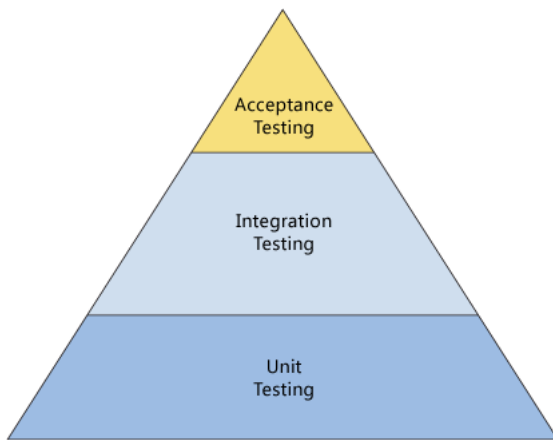
This section includes the following topics:

- [Testing Overview](#)
- [Unit Testing with Mock Objects](#)
- [Integration Testing](#)
- [Acceptance Testing](#)
- [How to: Create Automated UI Test Methods for SharePoint](#)

Testing Overview

Testability was one of the primary design goals for the Training Management and Partner Portal applications. These applications, in addition to the SharePoint Guidance Library, use a layered testing approach. The following diagram from the Acceptance Test Engineering Guidance shows the testing layers.

Testing layers



Each type of test has a distinct purpose. They are the following:

Unit tests. Unit tests are written by developers and run under a unit testing framework, such as Microsoft Visual Studio Team System or NUnit. Unit tests isolate and verify discrete units of program logic. They isolate the logic by replacing dependencies on the run-time environment, such as SharePoint, with test-provided substitutes. Isolation allows unit tests to run quickly, and developers can run unit tests frequently.

Integration tests. Integration tests differ from unit tests in that the code under test is not isolated. Integration tests are written by developers or testers. They run in a unit testing framework.

Acceptance tests. Acceptance tests consist of multiple steps that represent realistic usage scenarios of the application as a whole. These tests verify that an application meets the needs of the intended users. Their scope includes usability, functional correctness, and performance. Generally, test engineers create these tests.

Note:

Acceptance tests can be automated by programmatically simulating user behavior. This form of user interface testing for browser-based interfaces is often called a Web test. Acceptance tests may directly invoke methods of application components if the target user scenario is a development scenario. For example, a scenario may be a developer writing code for customization.

This guidance uses Web tests for acceptance testing. It does not cover other forms of acceptance testing such as usability tests. For more information, see [Stress and Scale Testing](#) and [Acceptance Testing](#).

For more information about acceptance testing and test layers, see [Acceptance Test Engineering Guidance](#) on CodePlex and [Testing .NET Application Blocks](#) on MSDN.

Unit Testing with Mock Objects

The Training Management and Partner Portal applications include unit tests that use *mock objects*. Mock objects are instances of test-provided classes that simulate the behavior of external components. Mock objects isolate the application code under test. They create conditions that are otherwise difficult to produce, such as a disk full exception. For more information about mock objects, see [Exploring The Continuum Of Test Doubles](#) in *MSDN Magazine*.

Mock objects are implemented in the following three ways in the Training Management and Partner Portal applications' unit tests:

- **Mock views.** The Model-View-Presenter (MVP) pattern that is used by the Training Management and Partner Portal applications contains presenter classes that encapsulate business logic. Presenter classes update the properties of view interfaces that are provided by the top layer of the application. Unit tests exercise the functionality of a presenter class in isolation by providing a mock view implementation when running the unit test. For more information about the MVP pattern, see [The Model-View-Presenter \(MVP\) Pattern](#).
- **Mock services.** The components that comprise the data layer of the Training Management and Partner Portal applications are exposed using the Service Locator pattern. This pattern puts the selection of the interface implementation under programmer control at run time. The Training Management and Partner Portal unit tests take advantage of this architecture when testing the presenter layer by substituting test-specific stub implementations that provide the inputs and outputs that are needed for the specific test scenario. For more information about the Service Locator pattern, see [The Service Locator Pattern](#) and [The SharePoint Service Locator](#).
- **Tool-generated mock objects for system services.** Mock views and mock services are insufficient for testing the lowest layers of the application. For example, components that are provided by SharePoint are sealed classes with internal constructors that do not provide interfaces that allow for substitution. In this situation, it is necessary to impersonate SharePoint classes in a special execution environment.

Note:

There are a variety of tools available that support unit testing with mock objects. The Training Management and Partner Portal applications use a commercially available testing tool named Typemock Isolator that is provided by [Typemock](#). You must install this tool if you want to run the unit tests.

By using mock objects to isolate the code to be tested, the Training Management and Partner Portal applications' unit tests overcome many of the challenges that are inherent in testing SharePoint applications. The following are some of these challenges:

- Usually, application code that invokes SharePoint-provided classes can only execute on the SharePoint server. This means that the code must be deployed before it can be unit tested with any testing framework and that remote debugging must be used. By simulating SharePoint objects and by using the MVP and Service Locator patterns that permit run-time-swappable components, the unit tests can be executed in isolation on a local development computer.
- It is difficult to test error conditions and exceptions with live, system-level tests. By replacing system components with mock objects, it is possible to simulate error conditions within the unit test. An example is when the business logic handles a Database Full exception that is thrown by a dependent service. Obviously, you do not actually fill up the service's database. Instead, the mock service throws the exception when it is appropriate for the unit test.
- The complexity of configuring the testing environment is greatly reduced. It is undesirable for a unit test to depend on many subcomponents. For example, without mock objects, a unit test would fail if there were a problem with the SharePoint database, even though the test was not related to persistence.
- One problem with running unit tests against an actual instance of SharePoint is that it is difficult to configure the application that is running on SharePoint so that it is in a known state. This is required so that the unit tests perform as expected. A benefit of using mock objects is that you can always control the state.
- Mock objects improve the performance of the unit tests themselves. This is important because a best practice is to run all unit tests before each check-in. If the unit tests run slowly, this may become difficult or impossible.
- It is possible to write and run unit tests before all the application components are written. Only the interfaces need to be defined. This allows parallel development by multiple programmers.

The following sections include examples of the unit testing approach used by the Training Management and Partner Portal applications.

Testing the Course Registration Presenter with a Mock View and Mock Service

The **CourseRegistrationPresenter** class calculates the values that are used by the Training Management application's Web interface when an employee attempts to register for a training course. For more information, see [Register for a Course Use Case](#).

The Mock View

Unit tests for the **CourseRegistrationPresenter** class provide a mock view as an argument to the **CourseRegistrationPresenter** constructor. The implementation of the mock view is shown in the following code. This code is located in the `CourseRegistrationPresenterFixture.cs` file of the `Contoso.TrainingManagement.Tests` project in `Contoso.RI (VSTS Tests).sln`.

C#

```

private class MockCourseRegistrationView : ICourseRegistrationView
{
    public string PageTitle { get; set; }

    public string HeaderTitle { get; set; }

    public string HeaderSubTitle { get; set; }

    public string ContentMessage { get; set; }

    public bool ShowConfirmationControls { get; set; }

    public IList<TrainingCourse> Courses { get; set; }
    public bool ShowCourseSelectionControls { get; set; }
    public System.Collections.Specialized.NameValueCollection QueryString
        { get; set; }

    public string SelectedCourse { get; set; }
    public string SiteLink { get; set; }
}

```

The preceding code illustrates that the **MockCourseRegistrationView** class, which is used by the unit test, is a stub implementation that provides the properties of the interface.

The Mock Service

Unit tests for the **CourseRegistrationPresenter** class replace the application's **Repository** components with mock services. The following code is for a mock Registration Repository service. The code is located in the **MockRegistrationRepository.cs** file of the **Contoso.TrainingManagement.Mocks** project.

C#

```

public class MockRegistrationRepository : IRegistrationRepository
{
    public static Registration RegistrationReturnedByGet { get; set; }
    public static Registration UpdateCalledWithRegistrationParam { get; set; }

    public static void Clear()
    {
        RegistrationReturnedByGet = null;
        UpdateCalledWithRegistrationParam = null;
    }

    public int Add(Registration registration, SPWeb spWeb)
    {
        RegistrationReturnedByGet = registration;
        return 1;
    }

    public void Delete(int id, SPWeb spWeb)
    {
        throw new System.NotImplementedException();
    }

    public Registration Get(int id, SPWeb spWeb)
    {
        return RegistrationReturnedByGet;
    }

    public Registration Get(int courseId, int userId, SPWeb spWeb)
    {
        return RegistrationReturnedByGet;
    }

    public void Update(Registration registration,
        SPWeb spWeb)
    {
        UpdateCalledWithRegistrationParam = registration;
    }

    public string GetFieldName(Guid key, SPWeb spWeb)
    {
        switch ( key.ToString().ToUpper() )
        {
            case "FA564E0F-0C70-4AB9-B863-0177E6DDD247":
                return "Title";
            case "E5509750-CB71-4DE3-873D-171BA6448FA5":
                return "TrainingCourseCode";
            case "7E4004FA-D0BE-4611-A817-65D17CF11A6A":
                return "TrainingCourseCost";
            case "8E39DAD4-65FA-4395-BA0C-43BF52586B3E":

```

```

        return "TrainingCourseDescription";
    case "43568365-8448-4130-831C-98C074B61E89":
        return "TrainingCourseEnrollmentDate";
    case "AE2A0BBD-F22E-41DC-8753-451067122318":
        return "TrainingCourseStartDate";
    case "F5E6F566-FA7C-4883-BF7F-006727760E22":
        return "TrainingCourseEndDate";
    default:
        throw new NotImplementedException();
    }
}
}

```

The preceding code demonstrates how to create a mock registration repository service that impersonates the real registration repository during the unit tests. Note that several of the methods in the mock classes throw **Not Implemented** exceptions because they are not required for testing. You must implement the interface but not all of its methods. Implement only the methods you need to create an effective unit test.

Using the Mock View and Mock Service in a Unit Test

The following unit test for the **CourseRegistrationPresenter** class uses the **MockCourseRegistrationView** and **MockRegistrationRepository** classes. This code is located in the `CourseRegistrationPresenterFixture.cs` file of the `Contoso.TrainingManagement.Tests` project.

C#

```

[TestMethod]
public void RenderCourseRegistrationPopulatesView()
{
    string loginName = @"domain\alias";
    string courseId = "1";
    SPWeb mockWeb = CreateMockSPWeb(false);

    MockCourseRegistrationView mockView = new MockCourseRegistrationView();
    mockView.QueryString = new System.Collections.Specialized.NameValueCollection();
    mockView.QueryString["ID"] = courseId;

    TrainingCourse course = new TrainingCourse() { Id = 1, Code = "TestCode" };
    MockTrainingCourseRepository.TrainingCourseReturnedByGet = course;

    this.serviceLocator.Clear();
    this.serviceLocator.Register<IRegistrationRepository>(
    typeof(MockRegistrationRepository));
    this.serviceLocator.Register<ITrainingCourseRepository>(
    typeof(MockTrainingCourseRepository));

    CourseRegistrationPresenter presenter =
        new CourseRegistrationPresenter(mockView);
    presenter.RenderCourseRegistrationView(web, loginName);

    Assert.AreEqual<string>("Course Registration - TestCode", mockView.PageTitle);
    Assert.AreEqual<string>("Course Registration", mockView.HeaderTitle);
    Assert.AreEqual<string>("TestCode", mockView.HeaderSubTitle);
    Assert.AreEqual<string>("Would you like to register for course: TestCode?",
        mockView.ContentMessage);

    Assert.IsTrue(mockView.ShowConfirmationControls);
    Assert.IsFalse(mockView.ShowCourseSelectionControls);
    Assert.AreEqual("http://localhost/training", mockView.SiteLink);
    MockManager.Verify();
}

```

The preceding code performs the following actions:

- It creates an instance of the mock view.
- It configures the mock training course repository with specific data values (a training course with **Id=1** and **Code="TestCode"**) that are returned by the **Get** operation.
- It configures the service locator object to use the mock **TrainingCourse** and registration repository services instead of the real implementations.
- It instantiates a new **CourseRegistrationPresenter** object and passes the mock view as the argument to the constructor.
- It invokes the method to be tested, which is **RenderCourseRegistrationView**.
- It performs checks to see that the output values, as stored in the mock view, are the expected values.
- It invokes the **Verify** method of the **MockManager** class or object. This call validates that all the Typemock expectations have been met. For more information about using Typemock, see [Creating Mock Objects with the Typemock Isolator](#) later in this topic.

Testing an Event Receiver Using Mock SharePoint Objects

This section includes an example of how to use the Typemock Isolator tool to generate mock SharePoint objects

when testing the **TrainingCourseItemEventReceiver** class's **ItemAdding** event receiver method.

The Method Under Test

The following code shows the **ItemAdding** method. This code is found in **TrainingCourseItemEventReceiver.cs** file of the **Contoso.TrainingManagement** project.

```
C#
public override void ItemAdding(SPItemEventProperties properties)
{
    bool isValid = true;
    StringBuilder errorMessage = new StringBuilder();

    string title = string.Empty;
    string code = string.Empty;
    DateTime enrollmentDate = DateTime.MinValue;
    DateTime startDate = DateTime.MinValue;
    DateTime endDate = DateTime.MinValue;
    float cost = 0;

    using (SPWeb web = properties.OpenWeb())
    {
        ITrainingCourseRepository repository =
            ServiceLocator.GetInstance().Get<ITrainingCourseRepository>();
        this.Initialize(properties.AfterProperties, repository, web, out title,
            out code, out enrollmentDate, out startDate, out endDate, out cost);

        ....
    }
}
```

The preceding code takes an instance of the SharePoint **SPItemEventProperties** class as its input. It then queries this instance for the following values: **title**, **code**, **enrollmentDate**, **StartDate**, and **cost**. It also retrieves a SharePoint **SPWeb** object by invoking the **OpenWeb** method.

Note:

The standard .NET Framework run-time environment is insufficient for simulating inputs of this kind because the SharePoint **SPItemEventProperties** class is sealed. It would be impossible in this case to impersonate this class by creating a derived class (a wrapper or façade) that overrides all of the relevant methods.

The Unit Test

The following code shows the unit test for the **ItemAdding** method. This code is found in the **TrainingCourseItemEventReceiverFixture.cs** file in the **Contoso.TrainingManagement.Tests** project.

```
C#
[TestMethod]
public void ItemAddingPositiveTest()
{
    serviceLocator.Clear();
    serviceLocator.Register<ITrainingCourseRepository>(typeof(MockTrainingCourseRepository));

    // Set up the mock so the validations pass.
    SPItemEventProperties spItemEventProperties =
        this.CreateMockSpItemEventProperties("My Title",
            12345678,
            DateTime.Today,
            DateTime.Today.AddDays(1),
            DateTime.Today.AddDays(2),
            0, 100);

    ....
}
```

First, the unit test clears the **ServiceLocator** of all type mappings and then adds a mapping to a mock training course repository. The unit test then makes a call to a helper method named **CreateMockSpItemEventProperties** to create a mock object with the appropriate values.

The **ItemAdding** method is tested after the inputs are established. This is shown in the following code.

```
C#
{
    ....
    // Call the event receiver with the mocked SPItemEventProperties.
    TrainingCourseItemEventReceiver receiver = new TrainingCourseItemEventReceiver();
    receiver.ItemAdding(spItemEventProperties);

    // Assert that the cancel did not get set.
    Assert.IsFalse(spItemEventProperties.Cancel);
}
```

When the **ItemAdding** method is called, all calls to objects that are referred to by the **SPItemEventProperties** object are intercepted by the Typemock Isolator run-time environment. The environment causes the test-specific values to be returned from the calls to the mock **SPItemEventProperties** that is given as the argument to the

ItemAdding method.

After the method under test, which uses the data that is provided by the mock objects, completes, the unit test calls **Assert.IsFalse** to check that the update succeeded. (The **SPItemEventProperties.Cancel** property is set to **true** to cancel the **add** operation.)

Both positive and negative tests can be created using mock objects. For example, there is a unit test that uses incorrect data to check the behavior of the event receiver. The following code shows an example of a negative test.

C#

```
[TestMethod]
public void AddingCourseWithInvalidCourseCodeCancelsWithError()
{
    serviceLocator.Clear();
    serviceLocator.Register<ITrainingCourseRepository>(typeof(MockTrainingCourseRepository));

    // Set up the mock so the course code is invalid.
    SPItemEventProperties spItemEventProperties =
        this.CreateMockSpItemEventProperties("My Title",
            "1234",
            DateTime.Today,
            DateTime.Today.AddDays(1),
            DateTime.Today.AddDays(2),
            100);

    // Call the event receiver with the mocked SPItemEventProperties.
    TrainingCourseItemEventReceiver receiver = new TrainingCourseItemEventReceiver();
    receiver.ItemAdding(spItemEventProperties);

    StringAssert.Contains(spItemEventProperties.ErrorMessage, "The Course Code must
                                                                    be 8 characters long.");
    Assert.IsTrue(spItemEventProperties.Cancel);
}
```

In this case, the **Assert** checks that the **Cancel** property is set because the course code is not eight characters long. This should cause the operation to fail.

Creating Mock Objects with the Typemock Isolator Tool

The helper method **CreateMockSpItemEventProperties** that was used in the previous examples contains the code that invokes the Typemock Isolator tool. The following code is an example of how to use the tool. For more information about the Typemock Isolator tool, see the [Typemock](#) Web site. Also, Typemock provides a video on [Unit Testing SharePoint with Typemock Isolator](#).

Note:

The following code is written using the Natural Mocks API from Typemock. Typemock 5.0 provides a C# Arrange, Act and Assert (AAA) API. The Partner Portal unit tests use the new AAA API. For an example of the AAA API, see the unit tests that are implemented in the **PartnerSiteDirectoryFixture** class.

C#

```
private SPItemEventProperties CreateMockSpItemEventProperties(string title, string
    code, DateTime enrollmentDate, DateTime startDate, DateTime endDate, int
    courseCount, float courseCost)
{
    // Create any mock objects that will be needed here.
    SPItemEventProperties spItemEventProperties =
        RecorderManager.CreateMockedObject<SPItemEventProperties>();
    SPWeb spWeb = RecorderManager.CreateMockedObject<SPWeb>();
    SPList list = RecorderManager.CreateMockedObject<SPList>();
    SPItemEventDataCollection afterProperties =
        RecorderManager.CreateMockedObject<SPItemEventDataCollection>();
    ....
}
```

The **RecorderManager.CreateMockedObject** method is provided by the Typemock Isolator tool. It takes the type to be impersonated as a type argument and returns a mock object.

Note:

This code depends on the presence of the special run-time environment provided by the Typemock Isolator tool. The objects returned are mock objects; they are not instances provided by the SharePoint system.

Next, the code tells the tool what values to return in response to various method invocations.

C#

```
{
    ...
    // Record expectations for the AfterProperties collection.
    MockHelper.RecordSPItemEventDataCollection(afterProperties, "Title", title);
    MockHelper.RecordSPItemEventDataCollection(afterProperties, "TrainingCourseCode",
                                                code);
    MockHelper.RecordSPItemEventDataCollection(afterProperties,
```

```

        "TrainingCourseEnrollmentDate", enrollmentDate);
MockHelper.RecordSPItemEventDataCollection(afterProperties,
        "TrainingCourseStartDate", startDate);
MockHelper.RecordSPItemEventDataCollection(afterProperties,
        "TrainingCourseEndDate", endDate);
MockHelper.RecordSPItemEventDataCollection(afterProperties, "TrainingCourseCost",
        courseCost);

return spItemEventProperties;
}

```

The preceding code invokes the **RecordSPItemEventDataCollection** method of a helper class named **MockHelper** to define the argument/return value pairs. The following code is the method. It is found in the **MockHelper.cs** file in the **Contoso.TrainingManagement.Mocks** project.

```

C#
public static void RecordSPItemEventDataCollection(SPItemEventDataCollection
        properties, string fieldName, object value)
{
    using (RecordExpectations recorder = RecorderManager.StartRecording())
    {
        object val = properties[fieldName];
        recorder.Return(value).RepeatAlways().WhenArgumentsMatch();
    }
}

```

The **StartRecording** method of the Typemock Isolator tool establishes the pattern for future .NET Framework calls. It provides a context for establishing the call arguments and return values that are expected for this unit test. In this case, the code says that the property lookup operation will always return the specified value. The recording feature allows you to provide an expected pattern of calls and return values that will be used in your unit test.

Testing an Item Event Receiver Using Typemock Isolator AAA API

The Partner Portal application's unit tests use the Typemock Isolator AAA API. One example is the unit test for the **ItemAdding** method of the **IncidentTaskReceiver** class.

The Method Under Test

The following code shows the **ItemAdding** method of the **IncidentTaskReceiver** class. It can be found in the **IncidentTaskReceiver.cs** file in the **Contoso.PartnerPortal.Collaboration.Incident** project.

```

C#
public override void ItemAdding(SPItemEventProperties properties)
{
    try
    {
        incidentManagementRepository.WriteToHistory(
            string.Format(CultureInfo.CurrentCulture, TaskCreatedMessage,
                properties.AfterProperties[TitleField]));
    }
    catch (Exception ex)
    {
        ListExceptionHandler handler = new ListExceptionHandler();
        handler.HandleListItemEventException(ex, properties,
            Resources.HandleListItemEventExceptionValue);
    }
}

```

The **ItemAdding** method is the event handler that is invoked when an item of a specific content type is added to a list. The method attempts to write log information. It cancels the add operation if the call to **WriteToHistory** fails.

To call this method during a unit test, an **SPItemEventProperties** instance populated with test data is required. However, because SharePoint's **SPItemEventProperties** class is sealed and internally constructed, the unit test uses Typemock Isolator to create the required mock object.

The Unit Test

The following code is a unit test for the **ItemAdding** method of the **IncidentTaskReceiver** class.

```

C#
[TestMethod]
public void ItemAddingThrowsExceptionTest()
{
    SPItemEventProperties properties = Isolate.Fake.Instance<SPItemEventProperties>(
        Members.ReturnRecursiveFakes);

    SharePointServiceLocator.ReplaceCurrentServiceLocator(
        new ActivatingServiceLocator()
            .RegisterTypeMapping<ILogger, MockLogger>());
}

```

```

        .RegisterTypeMapping<IIncidentManagementRepository,
            MockIncidentManagementService>()
        .RegisterTypeMapping<IPartnerSiteDirectory,
            MockPartnerSiteDirectory>());

    IncidentTaskReceiver receiver = new IncidentTaskReceiver();
    receiver.ItemAdding(properties);

    Assert.IsTrue(properties.Cancel);
    Assert.AreEqual("Could not write the incident task to history. Please contact support.",
        properties.ErrorMessage);
    Assert.IsTrue(MockLogger.errorMessage.Contains("Service threw exception"));
}

```

The unit test creates a mock object representing a SharePoint **SPItemEventProperties** instance that will be passed into the method under test, **ItemAdding**.

Note:

The mock object in this test method is created using the C# Arrange, Act and Assert (AAA) API in Typemock Isolator. This API is different than the Natural Mocks API used in the unit tests for the Training Management application. The AAA API is simpler.

Next, the unit test configures the service locator with a mock logger class, mock incident management repository class, and mock partner site directory class.

The unit test calls the **ItemAdding** method. The test is able to check that the business logic in **ItemAdding** canceled the add operation and logged the appropriate error message. This behavior is expected because the mock incident management repository object used for this test was designed to throw an exception every time it is called.

The unit test includes a mock logger class that has a field that contains the message logged during the operation. The unit test uses this information to verify that the expected log message was written.

Note:

The Partner Portal reference implementation uses the SharePoint service locator implementation from the SharePoint Guidance Library. The implementation of the service locator in the SharePoint Guidance Library differs from the service locator implementation found in the Training Management reference implementation.

Integration Testing

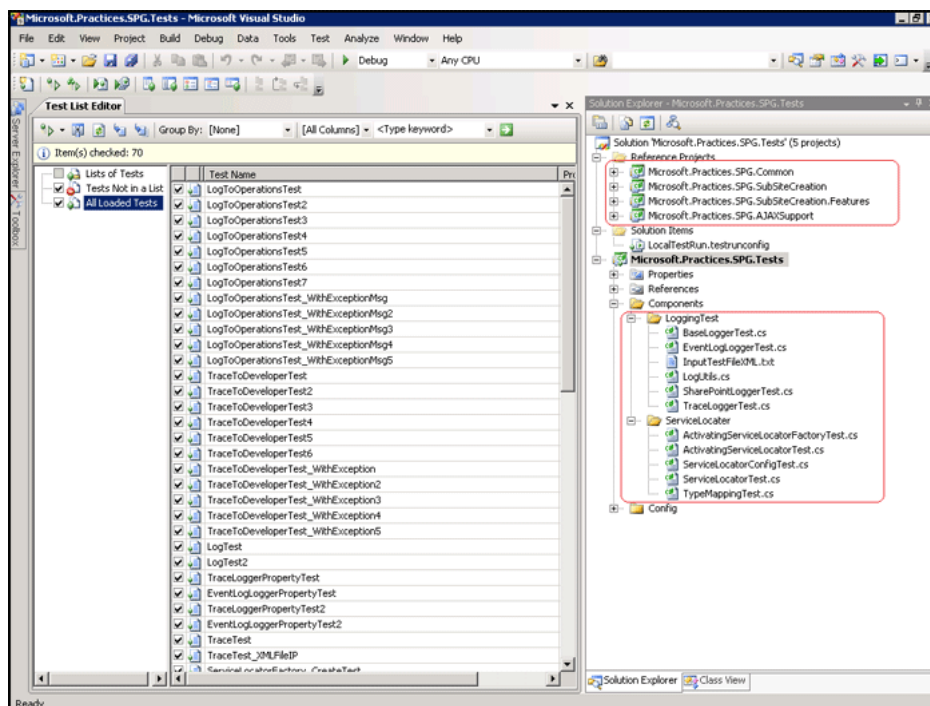
Integration testing is similar to unit testing in that tests invoke methods of application classes in a unit testing framework. However, integration tests do not use mock objects to substitute implementations for service dependencies. Instead, integration tests rely on the application's services and components. The goal of integration tests is to exercise the functionality of the application in its normal run-time environment.

Integration Testing in the Partner Portal Application

The Partner Portal application includes integration tests. These tests are located in the **Test\Microsoft.Practices.SPG.Tests** directory. Usually, tests such as these are used as part of a nightly build validation process.

The Partner Portal application includes an integration test solution in Microsoft Visual Studio Team System. This solution includes a Reference Projects folder that contains the functionality to test. The solution has test project that contains individual tests, as shown in the following illustration.

Integration Tests Solution



The test project, **Microsoft.Practices.SPG.Tests**, is organized by functional area. For example, there are directories named **LoggingTest** and **ServiceLocator**.

Logging Integration Tests

The **LoggingTest** directory includes integration tests that cover both positive and negative cases. Positive tests exercise the expected usage of the class. Negative tests set up conditions that are expected to fail and verify that the code fails as expected.

Each test sets up the conditions for the test, creates an instance of the class under test, performs an operation, and validates the expected result.

The following code shows the test method that verifies the **LogToOperations** method. This is an example of positive testing.

```
C#
[TestMethod]
public void LogToOperationsTest()
{
    string strMessage = "Custom Message by Automation Test Code. " +
        Guid.NewGuid().ToString("N");

    EventLogEntryType Severity = EventLogEntryType.Error;
    TraceSeverity TSeverity=LogUtils.MapEventLogEntryTypesToTraceLogSeverity(Severity);

    // BaseLogger is abstract so can use the SharePointLogger can be used
    SharePointLogger spLogger = new SharePointLogger();
    spLogger.LogToOperations(strMessage, EID_Default, Severity, strDefaultCategory);

    // Validation for Splog Entries
```

```

bool isValidSPL = LogUtils.ValidateSPLogs(strMessage, strDefaultCategory,
                                          TSeverity, EID_Default);

strMessage = string.Format("Category: {0}\n{1}", strDefaultCategory, strMessage);

// Validation for event log entry
bool isValidEL = LogUtils.ValidateEventLog(strEventSourceName, EID_Default,
                                          strMessage, Severity);

strMessage = null;
spLogger = null;

Assert.IsTrue(isValidSPL, "No entry is made in Share point Logs");
Assert.IsTrue(isValidEL, "No entry is made in Event Logs");
}

```

Service Locator Integration Tests

The ServiceLocator test folder includes positive and negative integration tests.

The following code is an example of a negative test from the **ActivatingServiceLocatorFactoryTest** class. The test validates that an exception is thrown when registering two interfaces.

C#

```

[ExpectedException(typeof(ArgumentException))]
public void RegisterTypeMapping_K_IType_Test2()
{
    ActivatingServiceLocator ASLocator = new ActivatingServiceLocator();
    ActivatingServiceLocator ASLocator2 =
        ASLocator.RegisterTypeMapping(typeof(IInterface1), typeof(IInterface2)
        , "Key1", InstantiationType.NewInstanceForEachRequest);
}

```

Acceptance Testing

A common form of acceptance testing consists of tests that exercise a user scenario from the user interface. These tests emulate the application keyboard and user interface interactions. A user interface test for a browser-based application is known as a Web test. Microsoft Visual Studio Team System includes automation for navigation that allows you create user interface tests. The Partner Portal application uses the following tests to conduct three forms of acceptance testing:

- **Build verification testing.** This is a regular check of application functionality from the perspective of an end user scenario of the application under development. For a list of build verification tests used by the Partner Portal application, see Build Verification Tests in the Partner Portal Application.
- **Stress testing.** This is a test of the application to handle a very high volume of user scenarios being executed concurrently. The purpose of stress testing is to drive the system beyond expected maximum load in production to see how it behaves. For more information, see [Stress Testing](#).
- **Scale testing.** This is a test of the application to determine whether the application can handle the required number of users under normal load conditions on the production hardware configuration. For more information, see [Scale Testing](#).

For more information about automated tests, see [How to: Create Automated UI Test Methods for SharePoint](#). For more information about using Visual Studio for testing Web applications, see [Introducing Microsoft Visual Studio 2005 Team System Web Testing](#).

How to: Create Automated UI Test Methods for SharePoint

This topic explains how to use the Microsoft Visual Studio Team System Web test record and playback engine. The following are the basic steps.

1. Create the scenario before you attempt to automate the test. The scenario outlines the sequence of steps to complete the operation that you are testing.
2. Record the Web test by following the steps in the scenario, and then save the Web test.
3. Clean your test environment, and then run the coded test. If your test passes, you can skip step 4.
4. Create an extraction rule to extract the GUIDs, and then replace the hard-coded GUIDs with the rule. You will receive the exception "There is no context parameter with the name ..." when you run the recorded test if a GUID needs to be extracted. There might be more than one GUID that needs to be extracted in a test.
5. Generate a coded test for the saved test.
6. Clean your test environment and run the coded test.

Software Requirements

You must have the following software installed to follow the procedures in this topic:

- Microsoft Visual Studio Team System 2008
- Windows SharePoint Services (WSS) 3.0 or Microsoft Office SharePoint Server (MOSS) 2007

The procedures in this topic use a SharePoint team site named TestGuidance, which is located at the URL <http://localhost/sites/TestGuidance>. However, the procedures will work with any other URL location for the team site.

Writing a SharePoint Test Method that Has No GUID to Extract

Use the following steps to create a very simple test method that illustrates the basic concepts involved in creating an automated test for SharePoint.

Step 1: Creating the Scenario

Creating the scenario involves identifying the pages, data, and clicks that you will enter for the test case. You will follow the scenario script when you use the record and playback engine to create the test. For this scenario example, you browse to <http://localhost/sites/TestGuidance> and verify that a successful response is returned.

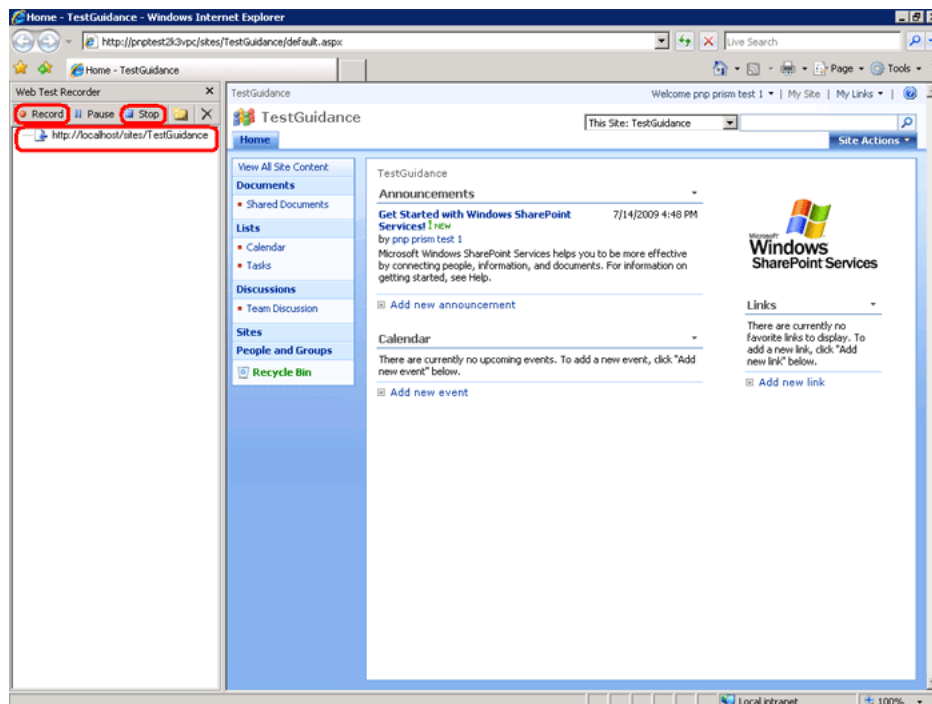
Step 2: Recording the Web Test

The following procedure describes how to use Visual Studio Team System to record a test.

To record the test

1. Start Visual Studio, and then create a test project.
2. On the **Test** menu, click **New Test**. The **Add New Test** dialog box appears.
3. Select **Web Test**.
4. Type **<http://localhost/sites/TestGuidance>** as the URL. This starts the record and playback engine, as show in the following illustration.

Record and playback engine



5. When the browser appears in recording mode, notice that in the Web Test Recorder pane (the left pane) the **Record** button is selected. Because this scenario is only a single step that retrieves the Web page, click the **Stop** button (next to the **Record** button) to stop recording and close Internet Explorer.
6. On the **File** menu, click **Save** to save the Web test.

Note:

In most scenarios, you will see pop-up messages that contain information about detecting dynamic parameters. However, this test does not generate the messages.

7. After processing is finished, the recorded Web test will appear, showing the steps that were recorded. In this scenario, the test will have a single step for viewing the page.

Step 3: Cleaning the Test Environment and Running the Saved Web Test

In this scenario, there is no clean-up required for the test environment. You should be able to run WebTest1.webtest successfully.

Step 4: Creating the Extraction Rule to Extract the GUIDs

The successful test is an indicator that there are no GUIDs to be replaced. Therefore, in this scenario, you do not need to create a GUID extraction rule.

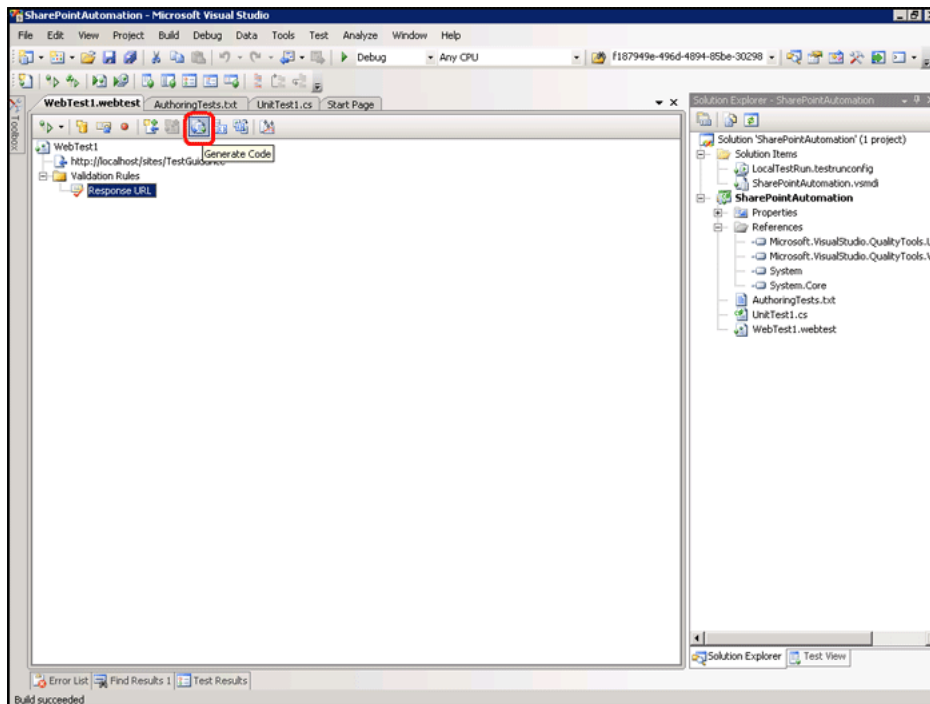
Step 5: Generating the Coded Test and Replacing the GUIDs with extracted value

The following procedure describes how to generate the coded test.

To generate the test code

1. Open WebTest1.webtest, and click **Generate Code** as shown in the following illustration. (Alternatively, you can right-click **WebTest1.webtest**, and then click **Generate Code**.)

Generating the coded test



2. This produces a C# file named WebTest1Coded, as shown in the following example code.

```
public class WebTest1Coded : WebTest
{
    public WebTest1Coded()
    {
        this.PreAuthenticate = true;
    }

    public override IEnumerable<WebTestRequest> GetRequestEnumerator()
    {
        // Initialize validation rules that apply to all requests in the WebTest
        if ((this.Context.ValidationLevel >=
Microsoft.VisualStudio.TestTools.WebTesting.ValidationLevel.Low))
        {
            ValidateResponseUrl validationRule1 = new ValidateResponseUrl();
            this.ValidateResponse += new
EventHandler<ValidationEventArgs>(validationRule1.Validate);
        }

        WebTestRequest request1 = new
WebTestRequest("http://localhost/sites/TestGuidance/default.aspx");
        yield return request1;
        request1 = null;
    }
}
```

3. You can now delete WebTest1.webtest, and use the coded test.

Step 6: Cleaning Your Test Environment and Running the Coded Test

In this scenario, there is no clean-up required for the test environment. You should be able to run the WebTest1Coded test successfully.

Write a SharePoint Test Method That has a GUID to Extract

When sites and other elements are created in SharePoint, they often have GUIDs associated with them. As a result, a recorded Web test might be tied to the GUIDs created for that site or element, and might not work with a different element instance if you run the test again. Examples of elements that create GUIDs are sites, lists, list items, and Web parts. This topic explains how to write a more complex test method to replace the GUIDs in these cases. Dynamic parameters are used to replace the GUIDs at test execution time.

Step 1: Creating the Scenario

The following scenario lists the steps that will be recorded.

To create the scenario

1. Browse to your TestGuidance team site.
2. Create a subsite under the TestGuidance team site.

3. Browse to the new subsite.
4. Create an announcement.
5. Stop the test.

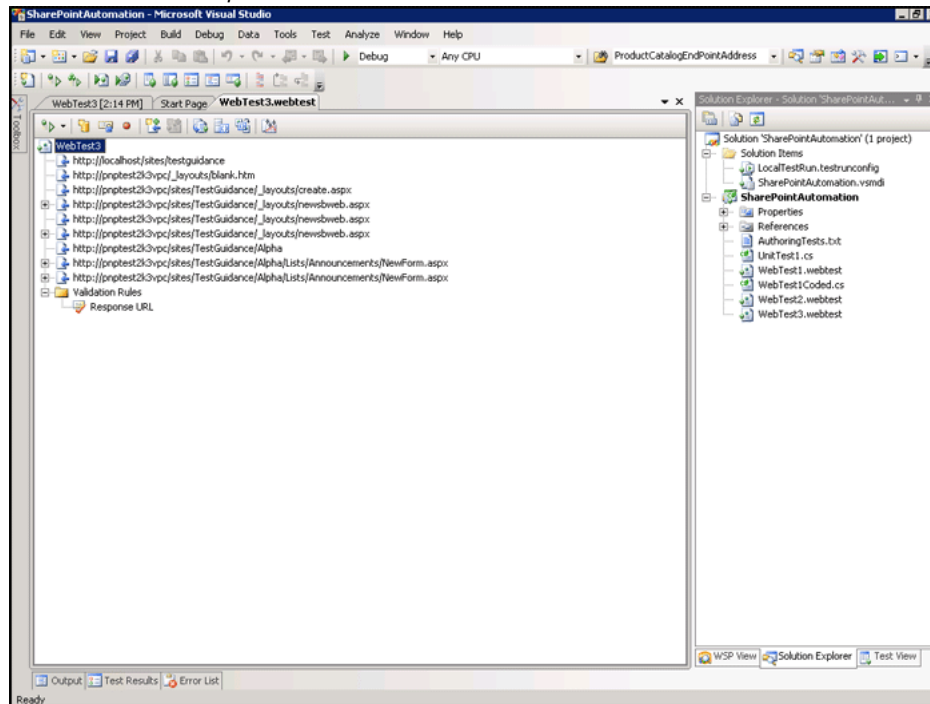
Step 2: Recording the Web Test

The following procedure describes how to use Visual Studio Team System to record the test.

To record the test

1. Start Visual Studio, and then create a test project.
2. On the **Test** menu, click **New Test**. The **Add New Test** dialog box appears.
3. Select **Web Test**.
4. Name the test **WebTest3**.
5. Start the recording, and then perform the scenario steps listed in step 1. Stop recording after you complete the scenario steps. The following illustration shows the test that you created.

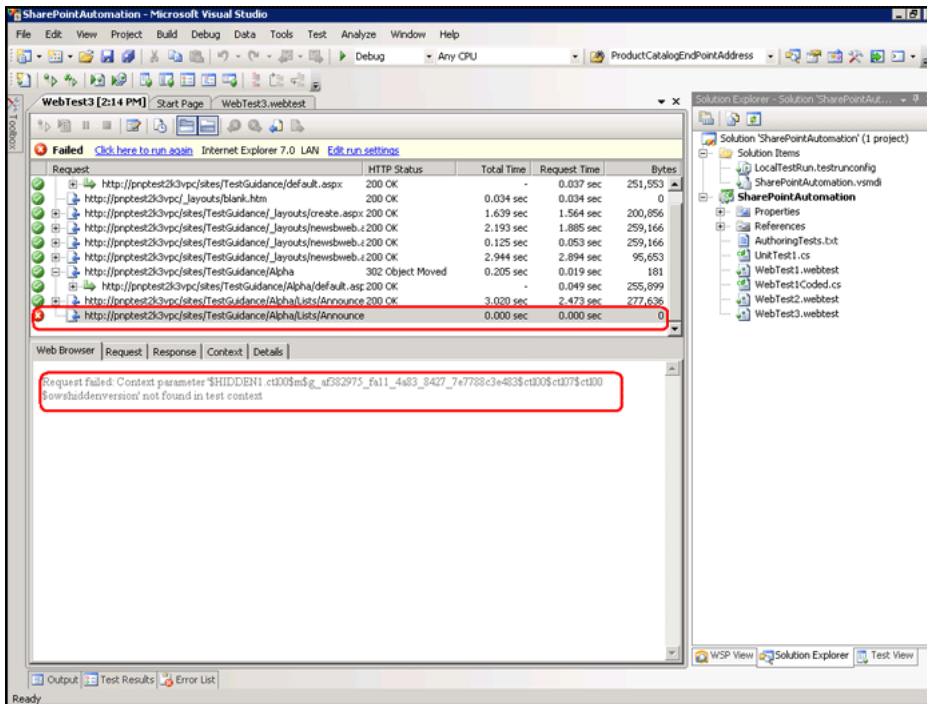
Recorded test example



Step 3: Cleaning Your Test Environment and Running the Saved Web Test

Delete the subsite that was created when you recorded the test, and then run WebTest3 again. The test will fail because the GUID is incorrect, as shown in the next illustration.

Test failure caused by invalid GUID



When you run the test again, it creates another site that has a different GUID. The test attempts to use the GUID created for the original site, which causes the test to fail. To correct the test, you have to locate the GUID, extract it, store it in a dynamic parameter, and then create code to replace the GUID in the generated test with the GUID in the dynamic parameter.

Step 4: Creating the Extraction Rule to Extract GUIDs

This step outlines how to identify and create a rule to extract GUIDs.

Before you create the extraction rule, click the error and confirm that that the error was caused by an incorrect GUID. In the example there is a 36-digit string—**af382975_fa11_4a83_8427_7e7788c3e483**—that represents a GUID from the original site created during the test recording session. If you repeat the steps and create another recorded test, you will see a different GUID value for the site that was created during your recording session.

Gathering the Extraction Information for the Dynamic GUID

The extraction rule is different for different controls. To extract its value, you must find out what the GUID represents. First, locate the failing request in your scenario.

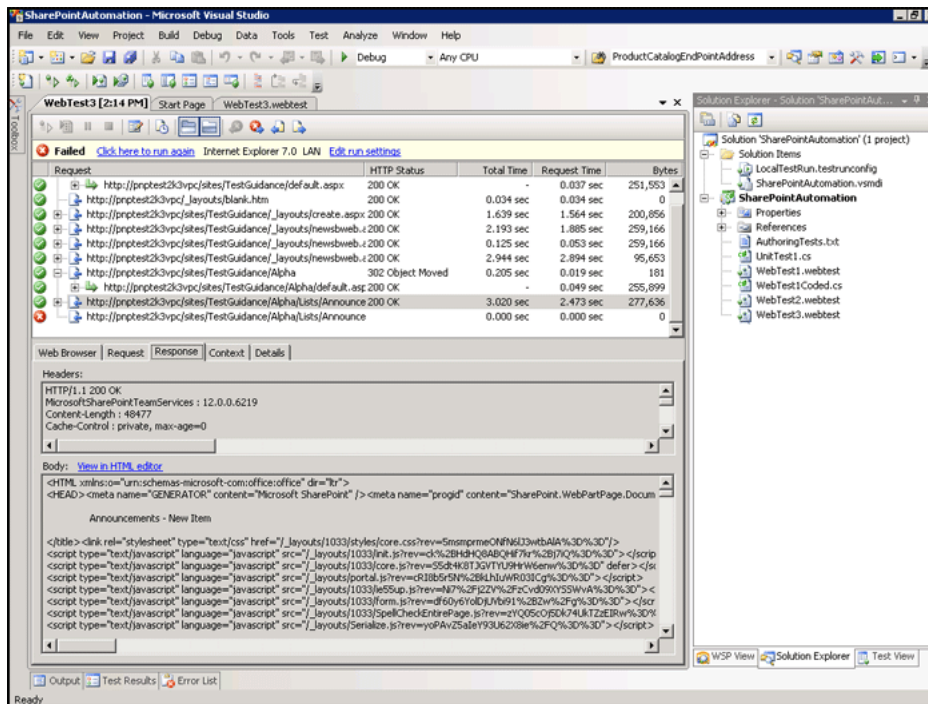
The request fails because it contains a GUID from the instance of the site used during the recording session. You must update the request to use the dynamic parameter that extracts the GUID from the response for the current instance.

Open the HTML response for the page that you submitted before you received the error (see the next illustration). To identify the GUID, you have to extract a GUID for the next request. This value will be different from the GUID described in the error message.

To extract the value of the GUID

1. Click the **WebTest3** tab.
2. Select the **NewForm.aspx** file that appears before the failing step, as shown in the following illustration.

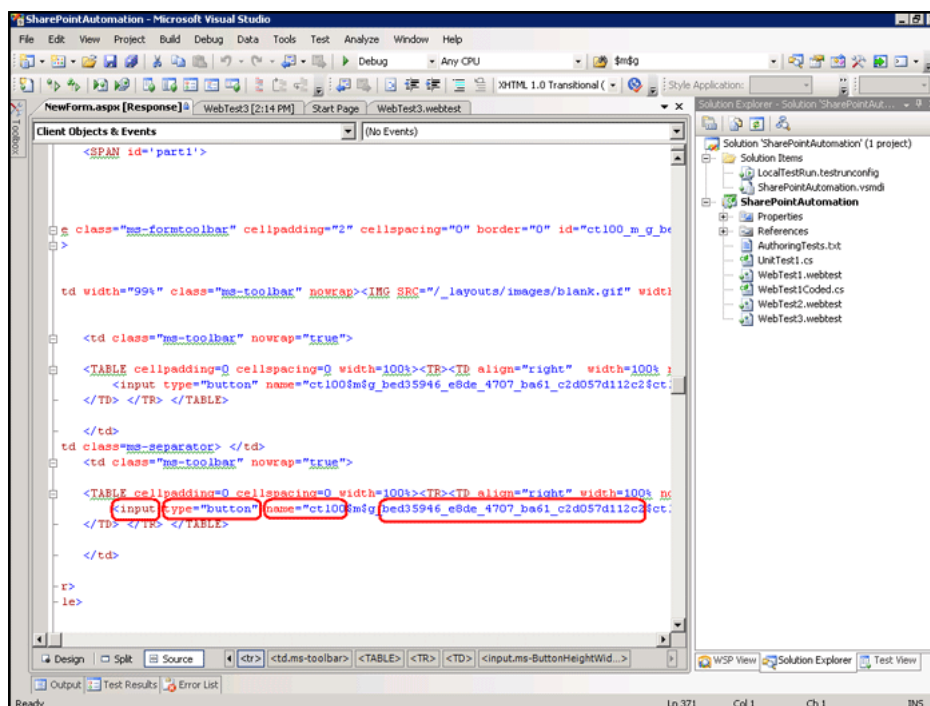
Response and HTML



3. Click the **Response** tab.
4. Most GUIDs start with **\$m\$g** in the response. Click **View in HTML editor** and search for **\$m\$g**. Look for the GUID information that follows so that you know what needs to be extracted. There may be more than one GUID, and you will need to determine the correct GUID to use. You can see the build verification test (BVT) cases included with the guidance for examples of more complex cases with multiple GUIDs. In this example, there is only one GUID. You need to extract the following GUID information:
 - Tag: **Input**
 - Attribute: **type** and its value: **button**
 - Attribute with GUID: **name**
 - String that contains the GUID

Because the GUID changes each time you run the test, the values that you see will be different from those shown in the following illustration.

GUID information to extract



Adding Extraction Rules to Your Saved Web Test

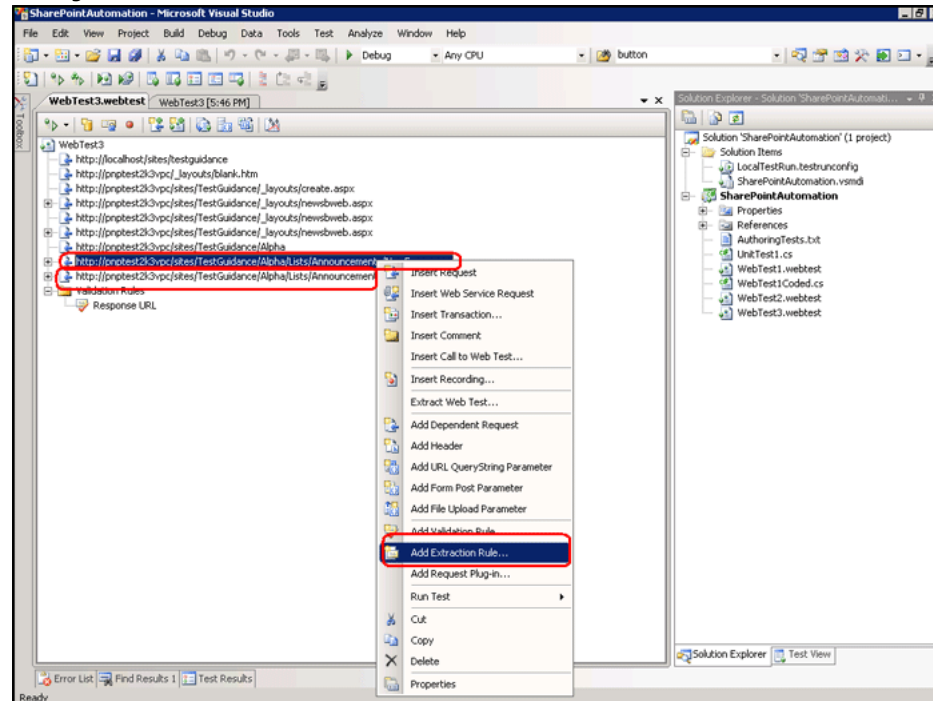
After you locate the GUID information, you have to add the extract operation to the Web request. Visual Studio

Team System has a feature for extracting any value in a Web response. You can use code or a Visual Studio Team System dialog box to add this capability. In this example, you will use the wizard to add the extraction operations. After you complete the wizard, you can generate the C# file, which will form your final SharePoint test method.

To add the extraction rule

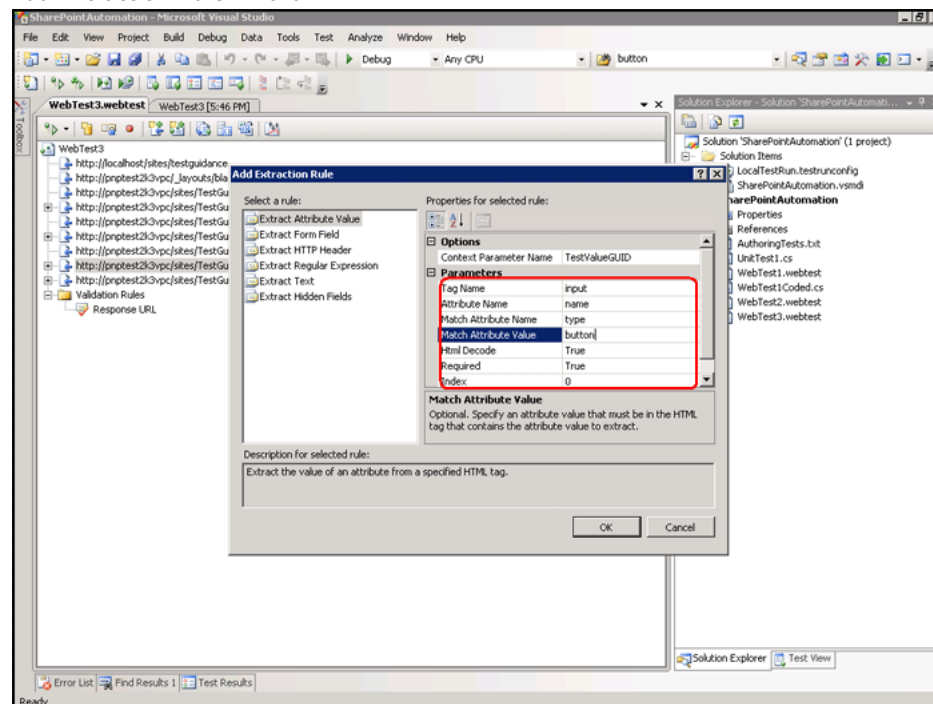
1. Open the WebTest3.webtest file.
2. Right-click the page request that occurs just before the failing request, and then click **Add Extraction Rule**. (See the next illustration.)

Adding the extraction rule

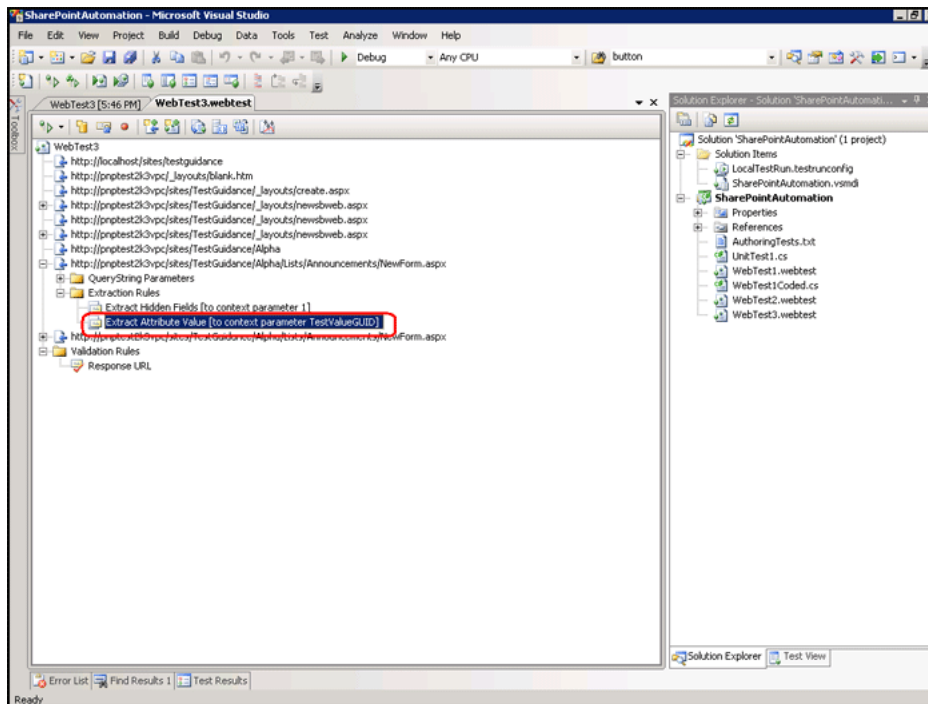


3. The Add Extraction Rule wizard appears (see the next illustration). In the **Options** box, type the **Content Parameter Name**. In the **Parameters** box, type the **Tag Name**, **Attribute Name** for the GUID, **MatchAttributeName**, and **Match Attribute Value**, based on the information you gathered in the previous procedure.

Add Extraction Rule wizard



4. Click **OK**.
You should now see the extract operation attached to the Web test, as shown in the following illustration.
Extract operation added to Web test



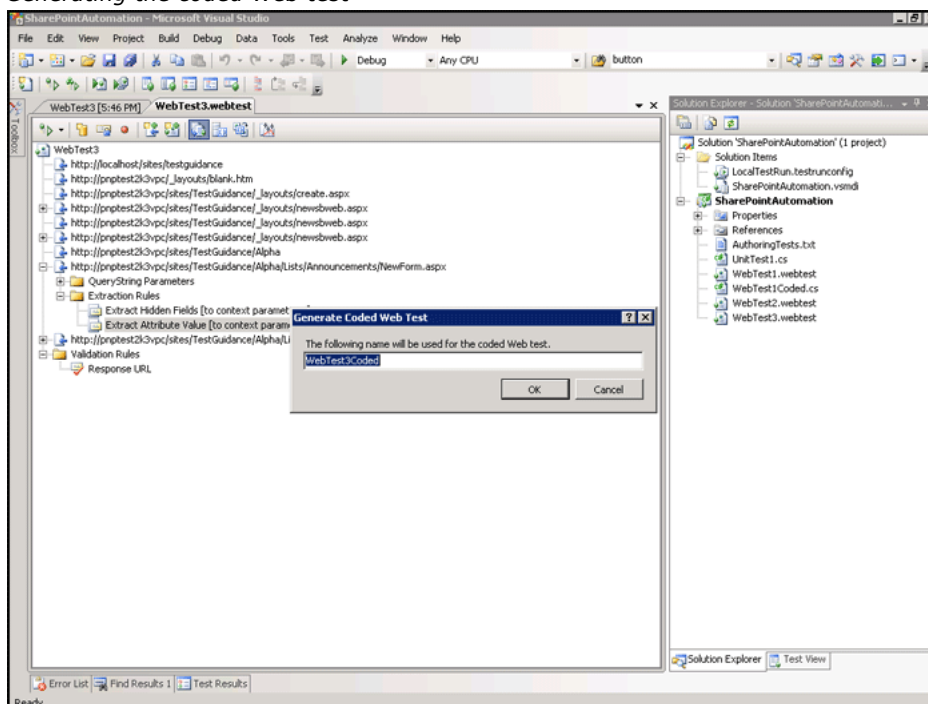
Step 5: Generating the Coded Test and Replacing the GUIDs with Extracted Values

The following procedure describes how to generate the coded test from the original recorded Web test and then replace the GUID with the extracted values from the dynamic parameter.

To generate the test code

1. Open the WebTest3.webtest file, and then click **Generate Code** as shown in the following illustration. (Alternatively, you can right-click **WebTest3**, and then click **Generate Code**.)

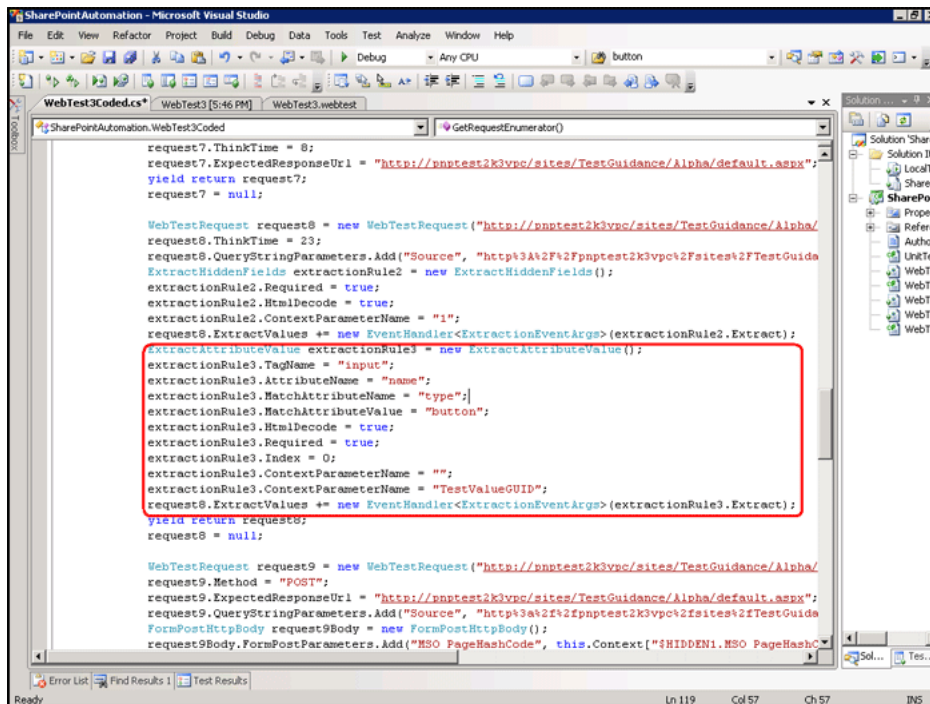
Generating the coded Web test



This produces a C# file named **WebTest3Coded**.

2. In the C# file, locate the **Extract** object by searching for the context parameter name. The example used the context parameter name **TestValueGUID** when the extraction rule was created. See the following illustration.

Locating the extraction object

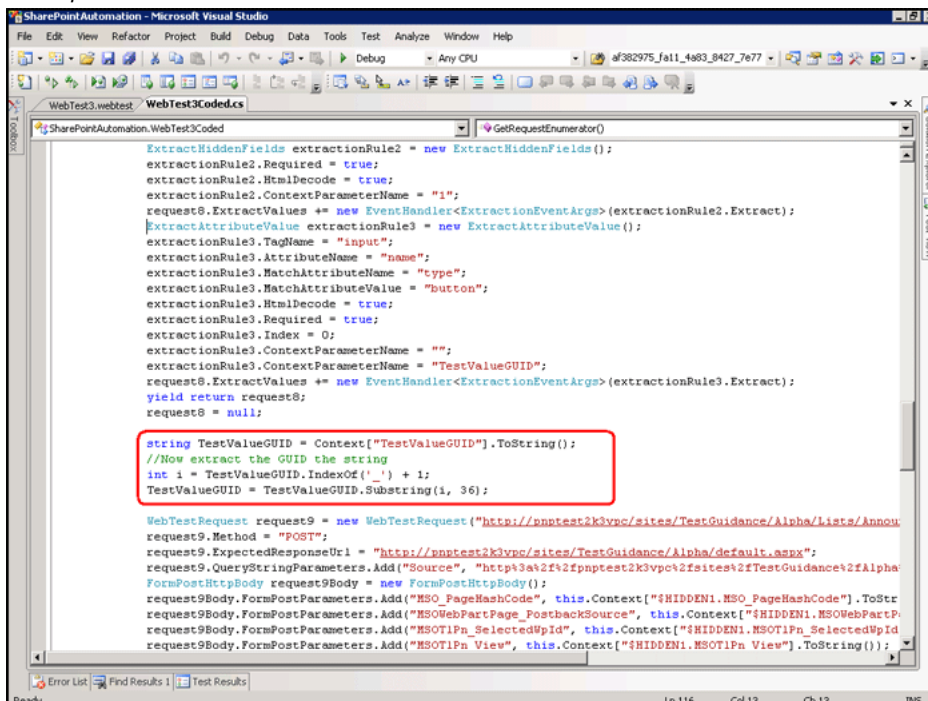


3. Create a GUID parameter to capture the extracted dynamic GUID from the extracted context parameter, as shown in the following example.

C#

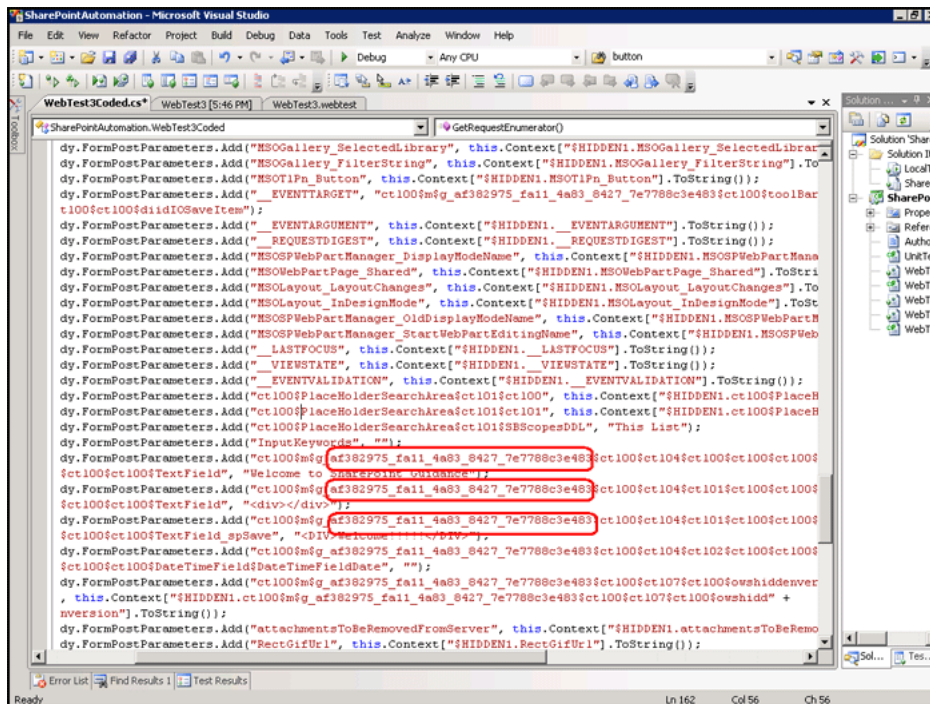
```
string TestValueGUID = Context["TestValueGUID"].ToString();
//Now extract the GUID the string
int i = TestValueGUID.IndexOf('_') + 1; //Note the underscore, not dash
TestValueGUID = TestValueGUID.Substring(i, 36);
```

4. The test framework runtime extracts the complete GUID string from the response and puts it into the context. The following illustration shows the code example above inserted in the test code to save the extracted GUID in the variable **TestValueGUID**.
5. *GUID parameter inserted in the test code*



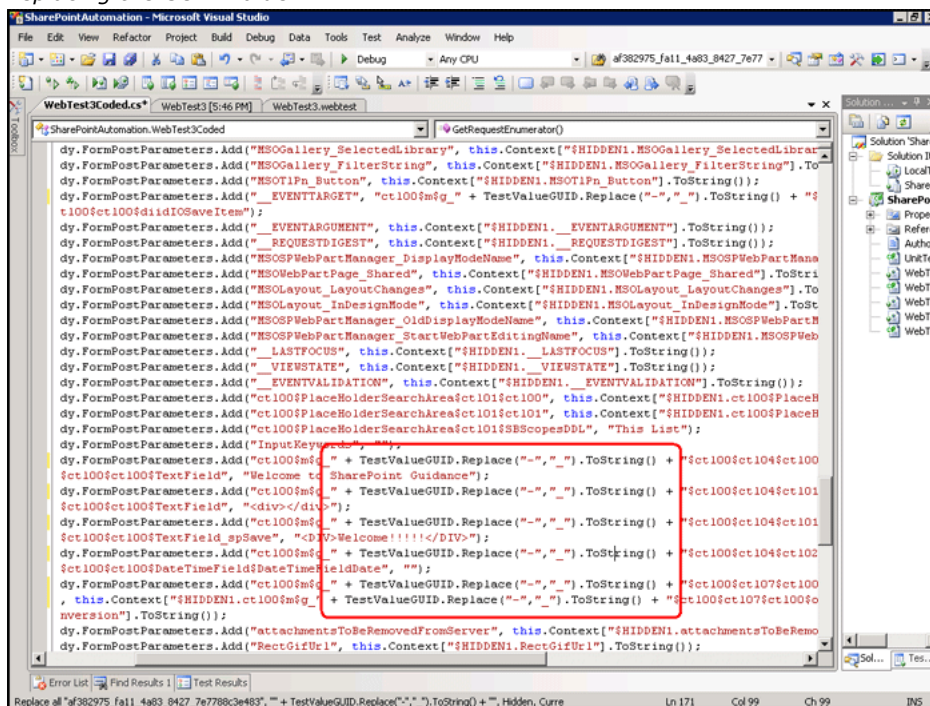
- 6.
7. Locate the hard-coded GUID value in the C# code for the failing request.
8. Find the GUID value that needs to be replaced by the dynamic parameter value from the code you created in step 3. To do this, look at the error information for the failed request. In the example, the GUID is the 36-digit string **af382975_fa11_4a83_8427_7e7788c3e483**.
9. Locate the numeric strings in the C# code, as shown in the following illustration.

Numeric strings in C# code



10. Replace the GUID value **af382975_fa11_4a83_8427_7e7788c3e483** with the GUID parameter string **" + TestValueGUID.Replace("-", "_").ToString() + "** as shown in the following illustration.

Replacing the GUID value



11. Save the files and then build the solution. Make sure that it compiles successfully.

Step 6: Cleaning Your Test Environment and Running the Coded Test

This section describes how to clean your test environment before you run the coded test.

To clean the environment

1. Delete all generated subsites under your test team site.
2. Open the **Test List View**, and then select the C# test method that you just created.
3. Run the test. It should be successful.
4. If the test fails, check the following:
 - Have you edited and extracted the GUID correctly?
 - Have you missed a GUID value?
5. If the test still fails, repeat the steps to replace the GUID value dynamically.

Building for Scale

Providing a centralized, integrated resource for information workers is a key aspect to SharePoint's popularity and cost effectiveness. SharePoint has a powerful infrastructure that integrates with line-of-business (LOB) systems. It runs multiple applications within a single shared infrastructure that provides a centralized place for information workers to access information and applications. Increasingly, SharePoint provides the infrastructure and services for both internal and external critical, high-volume Web sites. In these situations, performance and scale are important considerations. Neglecting them affects all applications that are running within the shared infrastructure. When building for scale, you need to consider both how to optimize your application and how to characterize your application behavior.

Scalability and Performance

Scalability and performance are often considered to be interchangeable. Although they are related, they are not the same thing. Performance measures how quickly your system responds to requests. Scale measures how much throughput your system can achieve, and throughput is the number of users that the application can handle simultaneously.

When you evaluate performance and scale, you should consider all components that comprise the system within the infrastructure. This includes Web front-end servers, application servers that host shared services providers, database servers, and network capacity. It is quite possible to make good decisions about your system's scalability that negatively impact its performance. For example, deploying services to multiple tiers will increase the scalability of a system, but it may degrade performance because it introduces additional network hops.

You can also make good decisions about performance that negatively impact the scalability of a system node. For example, if you cache a substantial amount of content in memory, users receive fast responses to their requests. However, using a cache decreases the amount of memory that is available to process requests. The result may be a reduction in the number of requests that you can simultaneously process on a Web front-end server.

Without performance and scale tests, it is not always obvious how a decision impacts a system. For example, in cases with computationally intensive applications, caching user information often results in improved overall throughput (and, therefore, scalability) by reducing computational bottlenecks. However, if too much information is cached, you may introduce a memory bottleneck in place of the computational bottleneck. The best way to determine the right balance is to test the application under realistic load conditions.

Not all decisions involve a tradeoff between performance and scalability. For example, optimizing the programming logic helps both. Such optimizations are always recommended, but they are best done after the system is complete. The effects of optimization are notoriously difficult to predict without load and stress testing, even when it is clearly necessary. A common example of where optimization is necessary is when you need to access a legacy line of business system that is slow to respond.

Some initial decisions about performance and scale can be made based on your experience with similar applications. This experience can help you to plan the application's architecture and design. However, after the system is developed, you must run performance and scalability tests to characterize and optimize tradeoffs between scale, performance, and cost. There are several ways to measure a system:

- Perform load and stress tests on the application in a realistic environment and under a realistic, simulated load.
- Stress test the individual components of the application by repeatedly running integration tests under load.
- Put the application into production and hope for the best. This is not a recommended strategy. The risks can be reduced by using hardware with greater capacities than what you think you require. This strategy depends on the patience of your users as you rework the application. A more moderate approach is to initially have a limited number of users. Depending on how critical the application is and the size of the user base, this may be a reasonable choice.

This topic discusses the lessons that were learned from load and stress testing the Partner Portal application. It does not provide a complete discussion of how to test all aspects of all SharePoint applications.

SharePoint supports many different types of functionality, and each type of functionality has different scalability and performance characteristics. For example, performance tuning a read-only Internet site is much different than performance tuning a collaboration site or an enterprise content management site. There is extensive guidance available on performance and capacity planning. For more information, see [Plan for performance and capacity](#) and [Performance and Capacity Planning Resource Center for SharePoint Server 2007](#) on TechNet.

For general guidance on improving the performance and scalability of .NET Framework code, see [Improving .NET Application Performance and Scalability](#) on MSDN.

Caching in SharePoint

Caching is a common technique for improving the performance and scalability of a system. There are four resources that impact scalability:

- Processing capacity
- Memory capacity
- Disk I/O
- Network capacity

In-memory caching decreases the amount of available memory in order to reduce the demands on processing capacity, disk I/O, and network capacity. Caches can be located on a disk or in memory. Disk-based caches can include information that is expensive to retrieve, such as binary large objects (BLOB) from a database, or information that is expensive to compute, such as the final output for a page.

Generally, caching improves performance at the cost of *data staleness*. Data becomes stale when the value for the information stored in the cache differs from the value of the data in the backing store. As the time interval increases between when the data was last retrieved and the current time, the chances increase that the cached data is stale.

The other characteristic that affects the likelihood of staleness is the volatility of the backing store. Volatility is a measure of how frequently the backing data changes. For example, you may have data such as city names that have low volatility because cities do not change their names very often. This data can be cached for a long time with little chance of the data becoming stale.

You may have data that changes at fixed times. For example, a catalog system may batch together and apply all editing changes to the production data every night. In these cases, you know when the data becomes stale, and you can flush the cache at that time.

In cases where data is highly volatile and must be accurate, it may not be a good candidate for caching. For example, in a high-volume system, inventory changes very rapidly. It is not a good candidate for caching if you require an accurate count. In many cases, caching indicators that are derived from the data's state may be enough. For an inventory, you may only need to know that the inventory count is adequate, that it is critically low, or that there is no inventory available. You can cache the derived inventory indicator value because it is much less likely to become stale than the actual inventory count.

Another technique that can prevent staleness is to invalidate the data in the cache when items in the backing store change. This approach detects changes in the backing store and sends an event to each instance of the cache to invalidate the cached value. Implementing this type of notification system in a farm can be complex and expensive. If the data source is a database such as SQL Server, cache invalidation capabilities may be standard features.

SharePoint-Specific Caching Mechanisms

SharePoint maintains several caches that are described in detail in [Caching in Office SharePoint Server 2007](#) on TechNet. The following sections give an overview of the caches that are implemented or extended by SharePoint.

Output Cache

The output cache is based on the ASP.NET output caching feature. Office SharePoint Server 2007 extends the capabilities of the ASP.NET output cache with *cache profiles*. A cache profile is a group of settings that gives greater control over managing the caching output than what is available in ASP.NET. For example, it allows you to customize how a page is rendered for users with different permission sets. A cache profile can be applied across a set of pages.

A cache profile is similar to but more powerful than the "vary by" capability that is native to ASP.NET. The Partner Portal application's promotion pages can use cache profiles to customize cached pages for specific security levels. The Promotions site collection can be configured to use the Extranet (Published Site) cache profile, which has Vary by User Rights turned on. This prevents users with different permissions, such as users that belong to different partner groups, from accessing the same cached publishing pages.

To select the Extranet cache profile for the Promotions site collection

1. Navigate to <http://localhost:9001/sites/promotions>.
2. Click **Site Actions**.
3. Click **Site Settings**.
4. Click **Manage All Site Settings**.
5. In the Site Collection Administration column, click **Site collection output cache**.
6. In the **Default Page Output Cache Profile** section, click **Extranet (Published Site)** in the **Authenticated Cache Profile** drop-down list.

In addition to using cache profiles, you can implement a **VaryByCustom** event handler that is supported by the ASP.NET output cache.

Note that entries in the output cache are periodically refreshed. This means that data latency can be an issue.

For more information about SharePoint output caching, see [Custom Caching Overview](#) on MSDN.

Object Cache

The object cache stores list query results and SharePoint objects. SharePoint objects such as sites and Webs are

expensive to create. Obtaining these objects through the object cache can greatly improve performance if you repeatedly access the same objects. The object cache has the same data latency limitations as the output cache with regard to cached query results.

The Partner Portal application accesses the object cache through the **PortalSiteMapProvider** class to aggregate information from within a site collection. For more information, see [Techniques for Aggregating List and Site Information](#). The application also uses the Content Query Web Part. This uses the object cache for cross-list query caching. For more information, see [Caching in Office SharePoint Server 2007](#) on TechNet.

BLOB Cache

The BLOB cache is a disk-based cache that caches binary large objects such as sound and video files. Caching these objects will reduce the load on the network and the database. However, the cache increases disk I/O on the Web front-end servers because the objects are stored and retrieved from the local disk. The cache only works with items that are in document libraries. Although the BLOB cache plays an important role in improving performance, it has limited use for developers. There are no examples in the Partner Portal application of the BLOB cache.

ASP.NET Cache

In addition to the SharePoint caches, developers can use the ASP.NET cache directly. This is the most common technique for caching application data. For an extensive discussion of caching, see [Improving .NET Application Performance and Scalability](#) on MSDN. This section provides an overview of the ASP.NET cache. The following sections describe some considerations for using the ASP.NET cache.

Consistency

Cached application data is located in the memory of the Web front-end server. One issue to consider is if the information is consistent across all the Web front-end servers in the server farm. You must also consider if that information is consistent with the back-end data sources. A specific Web front-end server generally retrieves and caches the data at different times than the other Web front-end servers in the farm. If a data source, such as an item in a SharePoint list, changes, and one of the servers refreshes the item in its cache, the representation on this server is different from that on the other servers with the older data.

One way to solve this problem is with a synchronized cache (also called a coherent cache). [NCache](#) is an example of a product that provides a synchronized cache. Microsoft currently has a beta product named [Velocity](#) that also offers a synchronized cache capability. Another strategy is to use cached data for browsing information, but to go directly to the data source for transactional operations. For example, users who browse a catalog see cached pricing information, but when they purchase a product, the pricing information is retrieved directly from the pricing source to calculate the cost. The Partner Portal application does not include any examples of resolving cache/data consistency issues.

Staleness

Staleness is related to consistency and impacts the accuracy of the information. For this reason, it is not recommended to cache data that frequently changes. The longer information is cached, the greater the likelihood of inaccuracies. The Partner Portal application only caches information for short intervals.

Another technique to avoid staleness is to invalidate the cache when the back-end data changes. This must be done by each server in the farm that is caching the data. The ASP.NET cache supports invalidation, but you often need to implement additional functionality to propagate the invalidation across all servers in a farm. Invalidation is simpler with a distributed cache because the cache only needs to be invalidated once, and the distributed cache synchronizes the change across the farm. The Partner Portal application does not include examples of cache invalidation. For more information about cache invalidation with ASP.NET, see [CacheDependency Class](#) on MSDN.

Efficiency

Caching is intended for data that is frequently accessed. To perform efficiently, caches often have expiration policies that remove information that has not been accessed within a specific time frame. The Partner Portal application uses this technique for cached product information that is from the product catalog. For more information, see [Product Catalog](#).

Another strategy is to limit the overall size of the cache. After the cache exceeds this limit, it purges the least recently used information (this is referred to as an LRU cache).

The ASP.NET cache provides performance counters that indicate the cache's efficiency. One important value to monitor is the cache hit ratio, which is the ratio of hits to misses for the cache. This value is a good indicator of how often cached information is being reused.

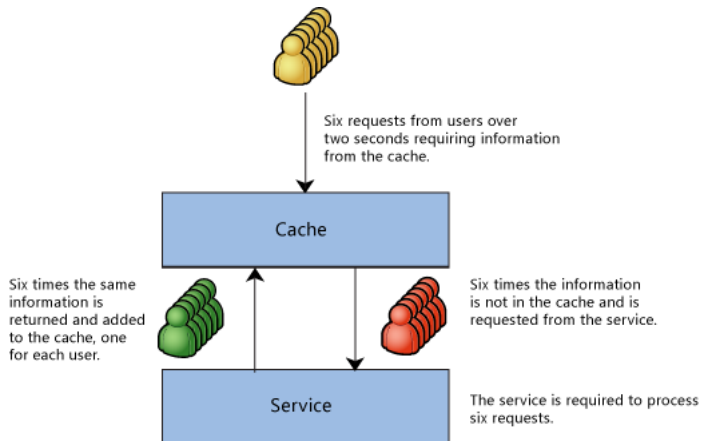
Security

Another consideration for caching is security. By caching data, you increase the likelihood that security will be compromised. The ASP.NET cache does not support permissions. If a process is compromised and an attacker runs code within that process, the attacker can access anything in the cache. If the data were not in the cache, the attacker would need to compromise additional boundaries to see it. Additionally, programming errors can allow users to see information that is not intended for them. After you understand the potential security risks and decide that they are acceptable, you can take precautions. For example, you can cache information that is specific to a set of users. The Partner Portal application applies this technique in the pricing repository to prevent one partner from seeing the pricing information that is intended for another partner.

Techniques for High Volume Sites

In high-volume sites, there is a race condition that occurs when a site is populated with an item. For detailed information about this issue, see [Best Practices: Common Coding Issues When Using the SharePoint Object Model](#) on MSDN. This section provides an overview. Although the ASP.NET cache is thread safe and does not need to be protected by a synchronization lock, there is another potential performance problem that synchronization can help solve. For example, assume that you cache information that is expensive to retrieve on a site that serves three requests per second for a page that uses the cached information. In addition, assume that it takes two seconds to retrieve the information from a back-end service. Finally, consider what happens when the cache expires the information. The following figure illustrates this.

Caching without a synchronization lock

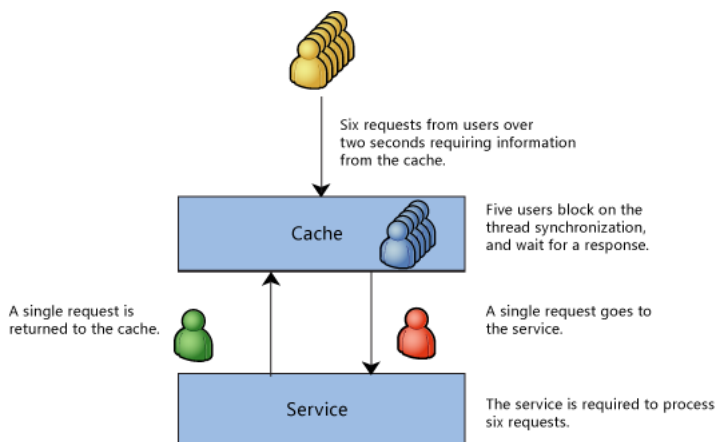


Because the information is retrieved every time the item is not in the cache, there are five unnecessary calls to the back-end server. This has two detrimental effects:

- It puts an additional load on SharePoint and on the service that responds to the requests.
- It causes the response time to be slower than necessary for five of the six users.

A synchronization lock around the logic that retrieves the information to cache eliminates this condition. This is shown in the following figure.

Caching with a synchronization lock



For code examples that demonstrate this technique, see [Best Practices: Common Coding Issues When Using the SharePoint Object Model](#) on MSDN. This approach is not implemented in the Partner Portal application. It was not implemented in this case in to reduce the complexity of the application and because the application does not include a scenario where high levels of traffic are generated.

Caching and the BDC

The Business Data Catalog (BDC) manages metadata for Web services and database services. It also provides a standard set of Web Parts that can be bound to that metadata without requiring any code. For more information, see [Consuming Web Services with the Business Data Catalog \(BDC\)](#). However, the default implementation of the BDC service and the related Web Parts do not provide a caching mechanism for storing the retrieved information. You can still take advantage of the BDC and cached information if you develop custom Web Parts to render the data. For an example, see [Repositories in the Partner Portal](#). The entity information is retrieved from the BDC and is stored directly into the ASP.NET cache.

Session State Management in SharePoint

Session state refers to user-specific information that is cached on a Web front-end server. Session state can either be persisted to a database between user requests or kept in memory. Session state can also be kept in a session state server, which is hosted in an external process. By default, SharePoint uses a SQL Server store for session state. This topic discusses aspects of session state that are specific to SharePoint. For a general discussion of how to manage session state, see the references at the end of this topic.

SharePoint uses ASP.NET to manage session state. Because ASP.NET supports Web farms, a SharePoint application does not need additional configuration to handle session state when using multiple Web front-end servers in a farm environment. The ASP.NET session state infrastructure includes session state providers that support persisting session state information to different stores. SharePoint uses the **SqlSessionStateStore** provider. For more information about ASP.NET session state providers, see [Session State Providers](#) on MSDN.

The **SqlSessionStateStore** provider supports custom partition resolvers that implement the **IPartitionResolver** interface. A partition resolver redirects the **SqlSessionStateStore** provider to a particular database based on the session ID. SharePoint implements a partition resolver that redirects SharePoint sessions to SharePoint database tables. SharePoint creates a session state database when the primary shared services provider (SSP) is installed. For information about configuring the primary SSP, see [Configure the primary Shared Services Provider](#) on TechNet. For more information about SharePoint sessions, see [Manage session state](#) on TechNet.

Microsoft Office Forms Server and Microsoft Office Project Server rely on ASP.NET session state. Excel Services relies on session state but tracks session information directly within the Excel Services SSP instead of relying on the ASP.NET session infrastructure.

Note:

SharePoint does not automatically remove old session state records from the session state database tables. Performance will degrade if you do not regularly purge expired sessions. For information about how to do this, see [SharePoint Session State: The Guest That Just Won't Leave](#), a blog post by Todd Carter.

Because SharePoint uses ASP.NET session state, the following related MSDN guidance is directly applicable to SharePoint:

- [ASP.NET Session State Overview](#)
- [Chapter 3 – Design Guidelines for Application Performance](#)
- [Chapter 4 – Architecture and Design Review of a .NET Application for Performance and Scalability](#)
- [Chapter 6 – Improving ASP.NET Performance](#)

Compensating for Slowly Responding Services

Services may be slow to respond to requests for a number of reasons. They may encounter network bottlenecks, perform computationally intensive operations, or depend on legacy systems that are not designed for Internet-scale applications or response times. Caching can mitigate the effects of slowly responding services and should be used wherever possible. However, not all data is suitable for caching. For example, the data may be highly volatile or infrequently retrieved. Even when caching is used, the first user who retrieves the data will be affected.

If the data only affects a portion of a page, consider using an AJAX control, such as the ASP.NET **UpdatePanel** control. This decouples the section that relies on the slow service from the rest of the page, which remains responsive to users. The Partner Portal application demonstrates this in its product catalog, on the product detail page. A simulated line-of-business (LOB) system with a slow response time must supply a parts list. For more information about the **UpdatePanel** control, see [UpdatePanel Class](#) on MSDN.

Stress and Scale Testing

Stress testing is designed to put a system under an unrealistic load. This load level exposes software defects that are only seen under extreme conditions. Examples of these defects are memory leaks and thread contention. *Scale* testing determines the system configuration that is required to support the anticipated load. It also verifies that the system performs correctly for a large number of users and transactions.

The typical steps for stress and scale testing are the following:

1. [Define the scenarios to test.](#)
2. [Define the counters.](#)
3. [Create the load data and test cases.](#)

Defining the Scenarios to Test

The first step is to identify the scenarios that exercise most of the functionality of the reference implementation. For example, tests for the Partner Portal application include the following scenarios:

- Browsing the product catalog
- Browsing promotion pages
- Read/write requests for incident handling
- Read/write requests for order exceptions
- Handling exceptions
- Typical SharePoint usage

Functional scenarios can be individually executed in isolation or they can be mixed during stress and scale testing. When you mix scenarios, each scenario's frequency of execution should be weighted to reflect real-world load patterns. For example, the catalog browsing scenario should occur more frequently than exception handling.

To accurately define a scenario, you need to analyze the dataset and conditions required to create the load on the application. For example, in the Product Catalog scenario, the test team created a dataset of 100,000 products.

A potential performance concern in the design of the Partner Portal application was the use of item-level permissions in large lists. Therefore, a test was created that included 100 partners with 10 promotion pages each. The test showed that the use of item-level permissions did not degrade performance significantly.

Defining the Counters

Counters measure system parameters while the tests execute. You must identify the counters that will monitor the servers during the stress and scale tests. The following table contains a subset of the counters that were used to analyze the performance of the Partner Portal scenario.

Partner Portal Performance Counters

Test result area	Category	Counters	Computer	Comment
Overall	Test	Passed tests Failed tests Tests/second	Application machine(s) and Web front-ends	This category contains counters for test results and test rates. These are the most important counters.
Computers	.NET CLR exceptions	Contentions/second	Application machine(s) and Web front-ends	.NET exceptions thrown per second.
Computers	.NET CLR LocksAndThreads	Contentions/second	Application machine(s) and Web front-ends	.NET thread contentions thrown per second.
Computers	Process	W3WP	Application machine(s) and Web front-ends	Process associated with the Application Pool. Typically, it allocates the amount of a resource. This helps your computer to be more stable and secure.
Computers	Processor	Percentage of processor time	Application machine(s) and Web front-ends	Total process cycles being used on the machine.
Computers	Memory	Available MBs	Application machine(s) and Web front-ends	Total free memory available; make sure that it is stable during the test.
Computers	.NET CLR memory	Gen 0 collections Gen 1 collections Gen 2 collections	Application machine(s) and Web front-ends	Examine the Gen number to make sure that the relationship between Gen 0, Gen 1, and Gen 2 are balanced.
Computers	ASP.NET	Requests/second	Application	Measures the requests/second to

	applications		machine(s) and Web front-ends	determine the throughput of the application.

Creating the Load Data and Test Cases

Creating the load data and the test cases is a significant portion of the testing effort. You must complete the following tasks during this step:

1. Develop utilities to create the load data set.
2. Create test cases that represent the scenarios that you previously identified.
3. Create the load tests, which bind the test cases to the load data and then execute the tests.

Typically, you have to automate the creation of load data because of the amount of data required. For example, the developers of the Partner Portal application created the following load data for stress and load testing:

- A set of 100,000 products in the catalog accessed by the Partner Portal.
- 100 partner sites, each with 10 users for each partner.
- 1000 randomly generated promotions for products, with permissions for each promotion assigned to a specific partner.
- Comma-separated value (CSV) data files that provide the test cases with data to bind to for each submission.

Between runs testers reset the agents and any initial conditions that the tests required.

Creating Web Tests

Web tests are tests that simulate user-browser interactions with SharePoint. For the Partner Portal example, the Visual Studio record and playback engine was used to create the scale tests. The recorded scripts were modified to make the tests more flexible and resilient; for example, they use dynamic variables. The modifications addressed issues such as the use of GUIDs in URLs and pages that change for each site instance created by SharePoint. To gain a deeper understanding of the approach, see [How To: Create Automated UI Test Methods for SharePoint](#). For more information about the record and playback engine in Visual Studio, see [Introduction to Record and Playback Engine](#) on MSDN.

After the Web test is parameterized, it is bound to a dataset or tables that are used to drive the test. For example, for the browsing the product catalog scenario, the tests include a CSV file that contains 100,000 rows of product information that test case required so that it could execute. The test case can bind to the CSV to sequentially or randomly access this dataset, depending on how the test is configured.

The next topics explain the following:

- [Stress Testing](#). The goal of stress testing is to run a system at abnormally high load levels to identify issues such as memory leaks, thread contention, failure modes, and bottlenecks.
- [Scale Testing](#). Scale testing determines the capacity of a target environment that is running an application so that you can adjust the system for the expected throughput.

Stress Testing

The goal of stress testing is to run a system at abnormally high load levels to identify issues such as memory leaks, thread contention, failure modes, and bottlenecks. Stress testing is a form of *negative testing*. Negative testing evaluates a system outside of normal boundaries to detect whether the system fails in a predictable and acceptable way. While stress testing can give some indications of scale behavior, scale testing is used to determine scale characteristics.

Often, diagnostic tests are simpler if they run on a single computer. Stress tests for the Partner Portal application ran on a 32-bit and a 64-bit standalone installation. Test scenarios were executed individually and also mixed together in a combined stress test. Stressing individual scenarios makes it easier to isolate problem areas, while combining them identifies interaction problems under stress.

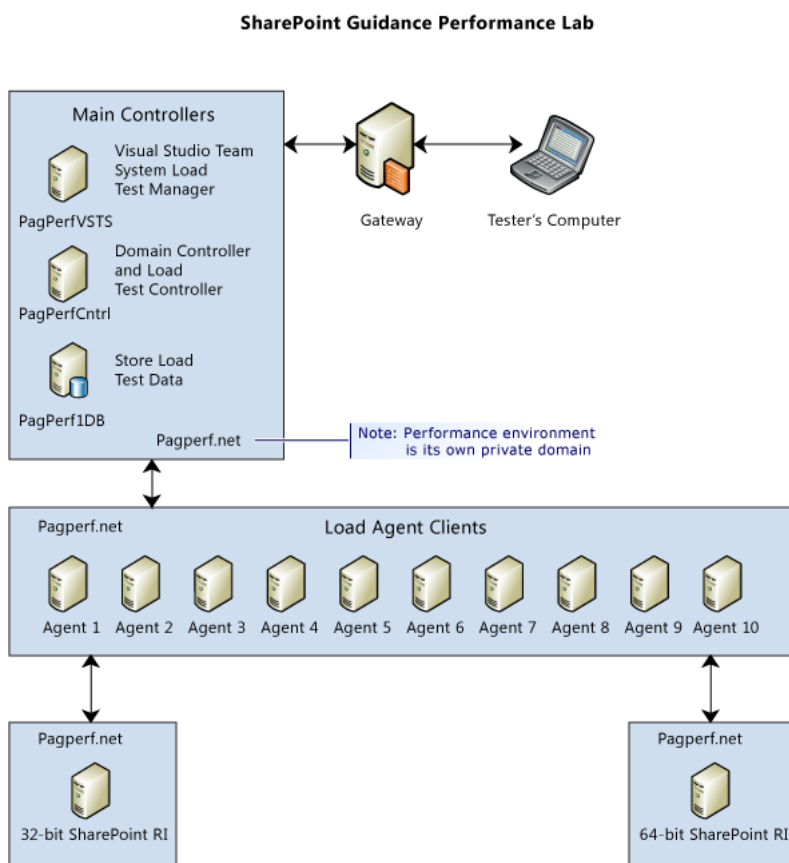
You can perform stress testing by using high loads for Web tests that simulate user behavior, or by using integration tests that call the code under test directly. Both techniques were used for stress testing the SharePoint Guidance Library components and the Partner Portal reference implementation. For more information about stress testing using integration tests, see [Integration Testing](#).

Although stress testing is not used to estimate system throughput, you should make sure that the throughput that is achieved during stress testing is reasonable. For example, if your system cannot serve one request per second while at a 50 percent processor load, it is likely that your system has bottlenecks that need to be identified and fixed.

Creating the Stress Test Environment

The SharePoint stress test environment should be created as a private network, isolated from the corporate domain, with a dedicated domain controller. This configuration allows you to control the traffic and server load without outside influences. The following illustration shows the environment that was used for stress testing the Partner Portal application.

SharePoint stress test environment

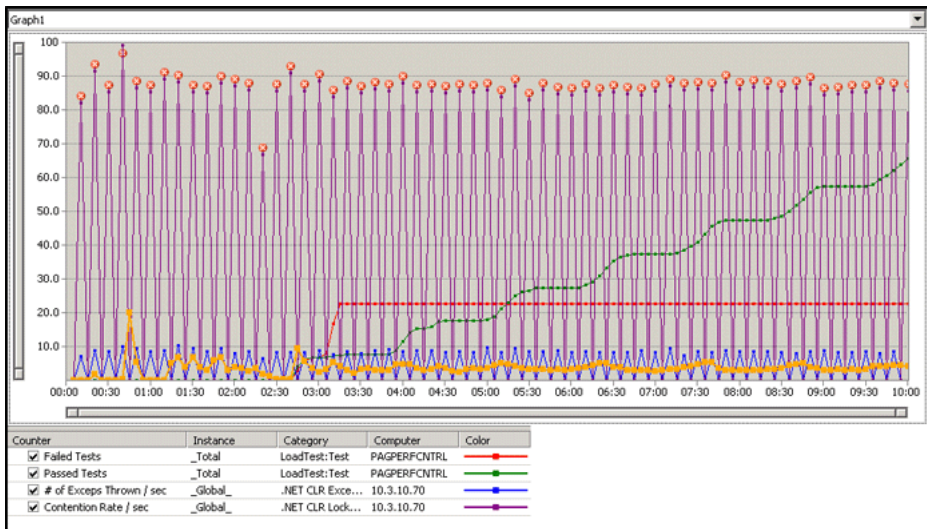


Analyzing Stress Test Performance Data

Performance analysis is an interactive process of executing tests and analyzing the results to detect abnormalities. The Partner Portal development team used the Visual Studio Team System (VSTS) test suite to run the scale and stress tests.

The targeted CPU utilization for the stress test was a steady state load of 80 percent. The number of simulated user sessions were balanced until the target load level was reached. The load tests ran for durations of 10 minutes to 12 hours. The data was collected from the specified counters every .5 seconds. When a test is complete, VSTS writes the results to a disk. The data was analyzed by creating graphs of counter values and looking at the resulting patterns. The following illustration shows a situation that identified contention that occurred in the application.

Test result graph showing significant contention

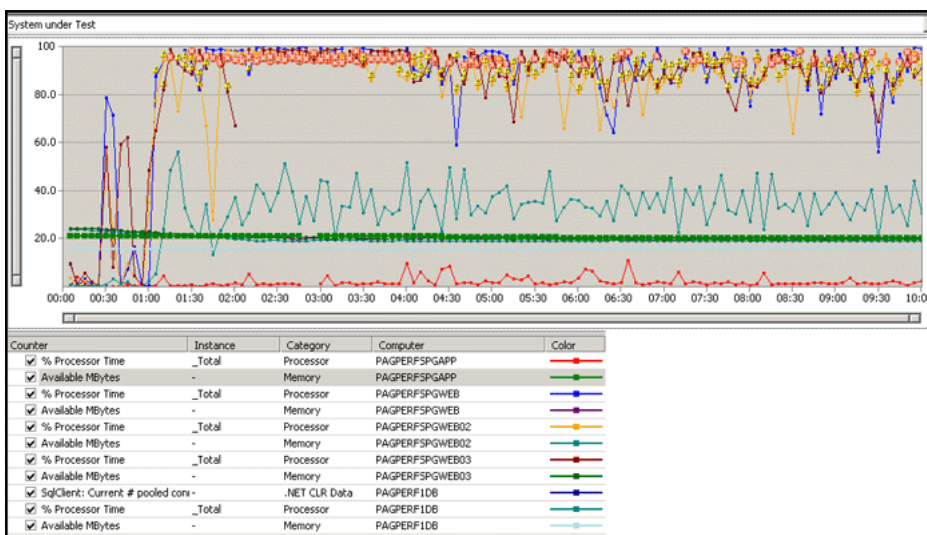


Analyzing test results requires some knowledge of expected baseline values. In this example, a normal value for contentions was approximately 1–2 per second; however, the test produced several hundred per second, with a peak value of over one million contentions per second. That level of contention had severe impact on responsiveness and throughput. After further analysis, the development team identified a problem with the trace provider. They used the WinDBG tool to trap the condition. After debugging, they discovered that the contention problem was a result of the method used to write messages to the unified logging system (ULS). They also identified an issue with the unregister call for the trace provider. As a result, they changed the logic so that it registered and unregistered much less frequently. These changes reduced the contentions per second to less than one per second.

A key aspect of becoming proficient at analysis is recognizing common patterns in the counter values and isolating the responsible logic. For example, if your system has memory leaks, you will see a decrease in free memory while the number of private bytes increases.

A successful stress test needs to ensure that the scenarios accurately reflect your production load, as shown in the following illustration. In the illustration, both Web server processes are above 80 percent, and memory from all computers is stable.

Successful stress test



For more information about counter patterns that indicate common issues, see the following MSDN blog entries:

- [Scale, Performance, and Capacity Planning](#)
- [Performance Testing and Performance Counters for SharePoint 2007](#)
- [SharePoint \(Performance, Stress\) Load Testing](#)
- [Good List of Performance Counters](#)

Scale Testing

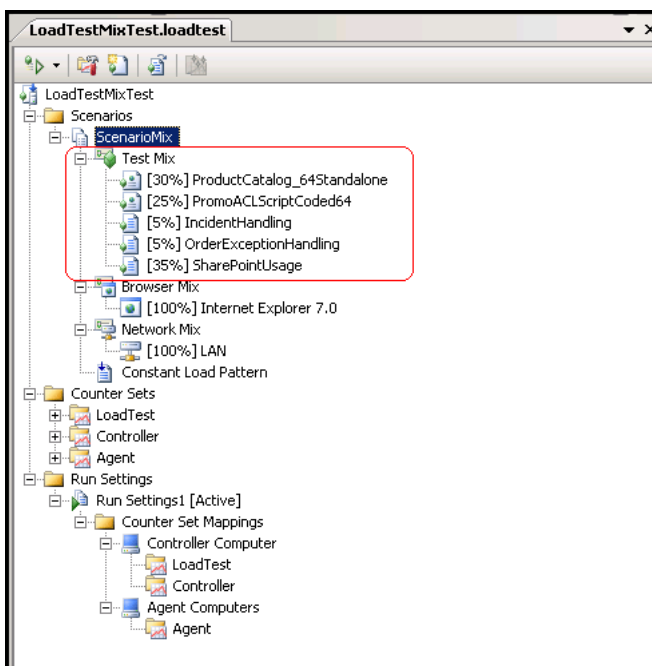
Scale testing determines the capacity of a target environment that is running an application so that you can adjust the system for the expected throughput. Additionally, the test should simulate other expected loads that will run on the same production farm as your application. The developers of the Partner Portal application used scale testing when they deployed the application to a SharePoint farm. For an outline of this approach, see [Plan for performance and capacity \(Office SharePoint Server\)](#).

To identify and eliminate performance bugs, you should perform stress testing before you begin scale testing. The steps for both tests are similar. Additionally, you need to estimate the throughput of your system before you begin the stress test.

The developers of the Partner Portal application estimated the throughput required for the production scenario and used the scale test to prove that they could meet the requirement.

For scale testing, they used the mix scenario, which produces a realistic usage pattern for each of the test cases used during the load test. The mix represents the percentage of cases in which the scenario is used by the testing tool. The following illustration shows an example of the mix used for the Partner Portal load test.

Partner Portal load test scenario



Finally, the developers selected the datasets to load into the reference implementation to produce a realistic, but high, load level. For example, in the Product Catalog scenario, they created a dataset of 100,000 products.

Estimating the Throughput

For scale testing, you need to determine the target throughput that you need to support in a production environment. The Partner Portal application uses the following table to determine the throughput in requests per hour. If a system is already in production, then you should use actual loads to determine an appropriate ratio for load tests. If you do not have an existing production system, you will need to make educated estimates about the usage you expect for each scenario. The estimates for the Partner Portal are based on projections for partners and usage.

The calculation of requests per second starts with an estimate of the number of active users per day. You then factor in the expected usage patterns for active users and a ratio for the highest number of users who would have active sessions at one time. The following are the estimates that were used for the Partner Portal application:

- The test scenario included 100 partners with 10 users each, for a total of 1000 users.
- It was predicted that up to 800 users would access the extranet in an 8-hour period.
- The partner usage mix was estimated to be the following:
 - 10 percent: light users who generate 35 requests per hour
 - 60 percent: typical users who generate 50 requests per hour
 - 25 percent: medium users who generate 125 requests per hour
 - 5 percent: heavy users who generate 250 requests per hour
- It was estimated that 80 percent of all partner users will be active on the Partner Portal at peak time.

This information was used to calculate the requests per hour (RPH) that the farm needed to support. The following table shows the calculation.

	Light load 10%	Typical load 70%	Medium load 15%	Heavy load 5%	Total
# of users	80	480	200	40	
Requests per hour per user	35	50	125	250	
Total requests per hour	2,800	24,000	25,000	10,000	61,800

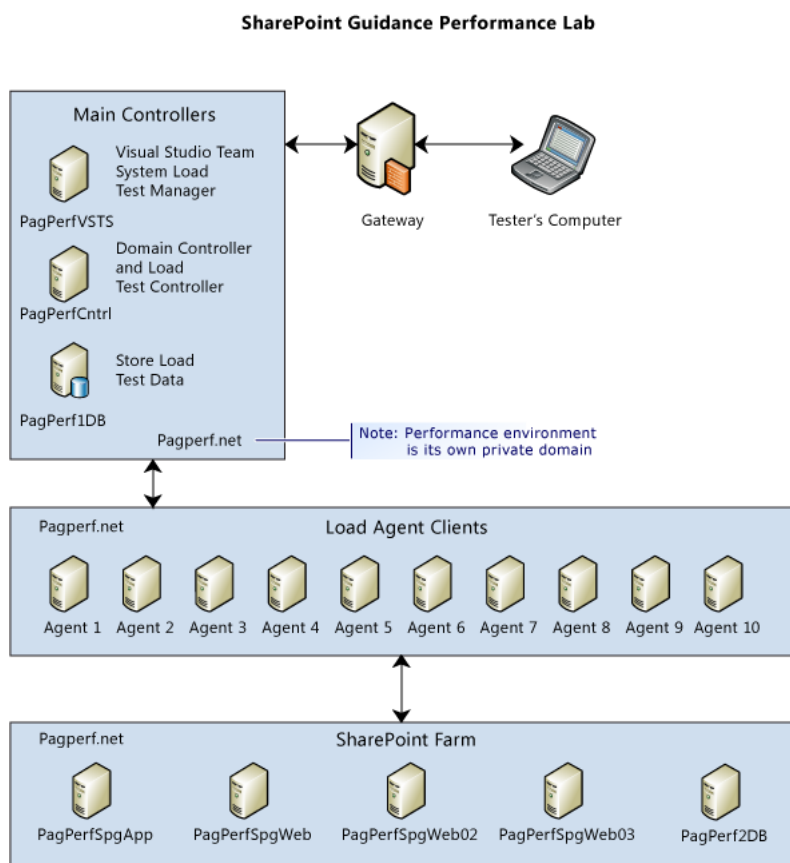
The next step was to divide the total requests per hour by 3600 to calculate the requests per second. Doing this produced 17 requests per second. Finally, the number of requests per second was multiplied by the highest percentage of active users on the site (80%). This produced a final anticipated load of approximately 14 requests per second.

After calculating the anticipated load, you need to find hardware that can support it. You might also decide to add capacity to ensure that your system can continue to operate if a single server fails.

Creating the Load Test Environment

The SharePoint performance test environment for the Partner Portal application is a private network that is isolated from the corporate domain and has a dedicated domain controller. This configuration allows you to control all traffic and server load and eliminates any external factors that might affect the environment. The following illustration shows the configuration that was used for scale testing.

SharePoint scale test environment

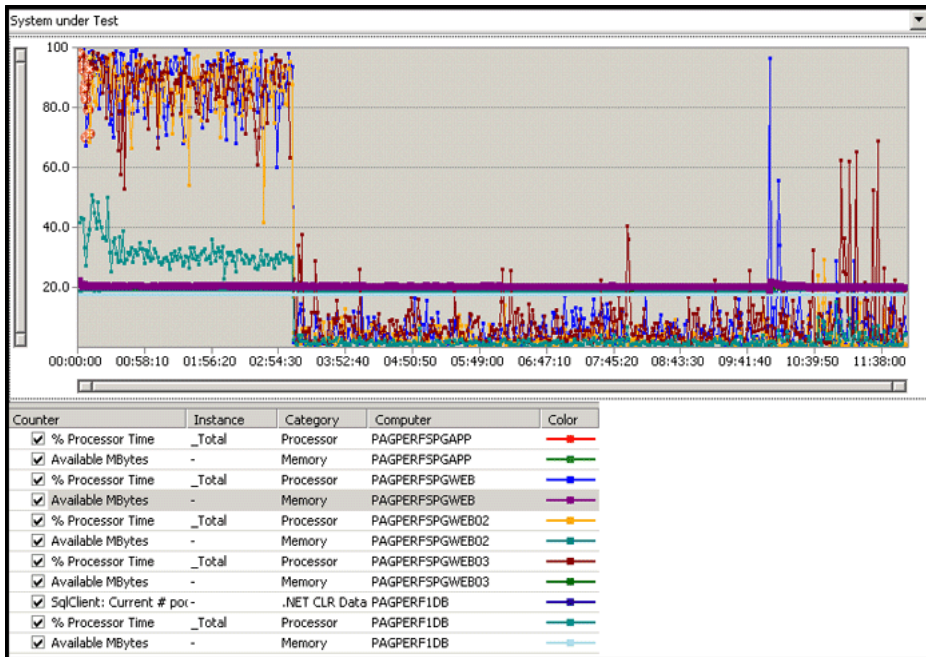


Analyzing Scale Test Performance Data

As with stress testing, performance analysis is an iterative process of setting up and running a number of load tests and examining the counters to discover patterns. The goal of scale testing is to determine system capacity for hardware resources. Therefore, scale testing might require that you change the number of agents or users to produce the targeted processor utilization. Forty percent is a typical target load for a production system. This load level can handle unanticipated peaks as well as system failures that might transfer the load to a different server in the farm. The stress tests for the Partner Portal application had durations that ranged from 10 minutes to 12 hours.

The following graph illustrates a problem that was encountered while running the scale tests.

Scale test graph



After three hours, the system performed a major reset. The cause was isolated to an environment configuration error, resulting in a modified state from a previous series of scale tests. This illustrates that it is important to make sure that the system starts from a known initial state.

After scale testing is completed, you will have tuned and optimized the production system settings for your system, and will have a good understanding of the hardware requirements for your production configuration.

More Information

For more information, see the following resources:

- [NeXpert Performance Tool](#)
- [VS Team System Test](#)
- [Controllers, Agents, and Rigs](#) on MSDN

Considerations for Extranet Development

This topic briefly describes some of the security issues you should consider when you develop an extranet for SharePoint. SharePoint has extensive documentation on designing and deploying extranets. For more information, see [Extranet Resource Center for SharePoint Products and Technologies](#) on TechNet.

This topic discusses the following subjects:

- [Planning the Extranet](#). This topic discusses some basic decisions you should make before you begin developing an extranet.
- How to: Lock Down Identity Viewing for a Site. This topic includes a procedure that shows how to prevent unauthorized users from seeing the People.aspx page.
- [How to: Expose a SharePoint Application to the Extranet and Use Forms-Based Authentication](#). This topic includes a procedure that shows how to extend a SharePoint application to the extranet and enable the application to use forms-based authentication.
- [Using the User Profile Store](#). This topic discusses how the Partner Portal application creates user profiles and associates them with partner IDs.

Planning the Extranet

Before you develop and deploy an extranet, you should consider some basic issues. Your particular business scenario will affect your decisions. You should consider the following basic issues:

- **User partitioning.** Are the users for the extranet internal users who will access information from an external location or are they external users with whom you will collaborate and share information? Are there multiple groups of external users or is there a single group?
- **Data partitioning.** How sensitive is the data? Will all the extranet users be able to access the same information or should information be limited to particular users or groups?
- **Process partitioning.** SharePoint can isolate information within a site collection, but in some high-security scenarios, you may need to provide process or server isolation.

In many cases, site collections offer an adequate level of isolation. Site collections are a security and data boundary within SharePoint. The Partner Portal application uses site collections to protect the privacy of each partner's information. If you require a high level of security, you can use the Web application as the partition. This provides process isolation as well as dedicated database instances. The drawbacks are that process isolation has more administrative overhead and requires more resources. Partitions at the site collection level are secure and adequate for most application scenarios.

The Partner Portal application is meant to support many external users, each of which is a partner. Based on this requirement, the following decisions were made:

- **To use an external identity store and forms-based authentication (FBA).** This isolates the user store and reduces operational complexity.
- **To use a site collection for each partner.** This isolates the read/write data so that only the partner can access it.
- **To use a central site collection for all partners' read-only data.** This makes it easier to manage the publishing process.

Some SharePoint functionality requires that users have Windows identities. One example of this is Microsoft Office SharePoint Server single sign-on. The Partner Portal application uses FBA, which is intended for users who do not have Windows identities. For this reason, it implements the Trusted Façade pattern instead of Microsoft Office SharePoint Server single sign-on. For more information, see [Security Considerations for LOB Integration](#). Before you decide to use FBA, you should familiarize yourself with the SharePoint implementation. For more information, see [Forms Authentication in SharePoint Products and Technologies](#) on MSDN. An alternative choice to FBA is to use a gateway product such as the [Intelligent Application Gateway](#) (IGA), which translates between external identities and a Windows domain.

The Partner Portal application uses SQL Server as the credential store. This reduces the complexity of the application. For more information, see [Security Decisions](#).

When you use extranets, you must also decide how to manage domain names. The Partner Portal application hosts all partners within the same domain. However, this approach has implications for security. If you allow users to save information to the portal, there is the possibility of a cross-site scripting attack. Generally, browsers can only prevent cross-site scripting attacks when the attacks occur across Web domains. To eliminate this possibility, you can put each user's site collection into its own Web domain. Keeping each site collection in its own domain adds some management overhead, but it is significantly less than maintaining separate Web applications. For more information about cross-site scripting attacks, see "Subsite scripting" in [Plan site security](#) on TechNet. For more information about how cross-site scripting attacks relate to the SharePoint logical architecture, see [Logical architecture components](#) on TechNet.

How to: Lock Down Identity Viewing for a Site

The People.aspx page in the _layouts directory displays all the users who have access to a site. Even if the link to this page is not exposed in the application, experienced SharePoint developers and administrators know its URL and can type it directly into the browser. In environments that are shared, such as the publishing site in the Partner Portal application, allowing any authenticated user access to the identities of other users is not acceptable. (Even if someone can see the People.aspx page, the information contained within the shared site is protected by the SharePoint security infrastructure.) You can use the following procedure to restrict a user from viewing the People.aspx page.

Note:

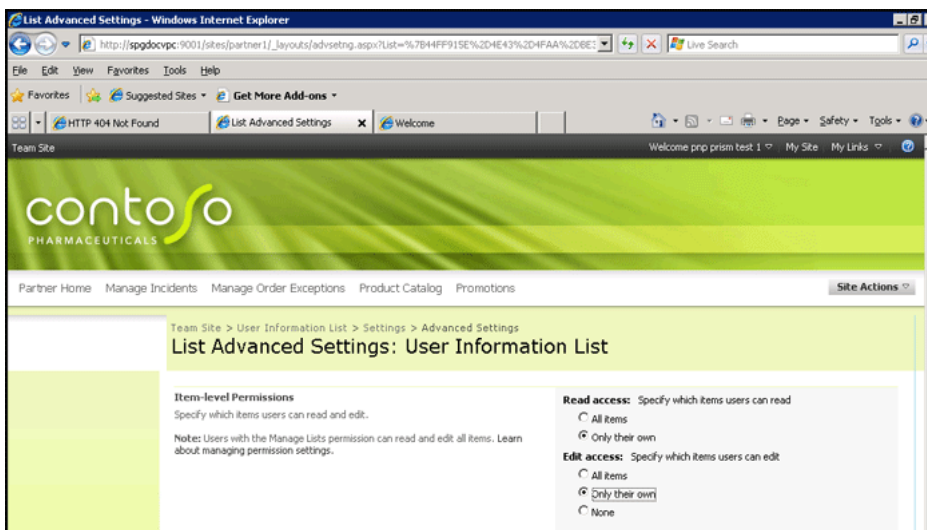
This procedure does not prevent users with full permissions from viewing and changing the page.

To secure the _layouts/people.aspx page

1. Navigate to the root of your site collection.
2. Click the **Site Actions** tab. Point to **Site Settings**, and then click **People and Groups**.
3. In the Quick Launch, click **All People**.
4. On the toolbar, click **Settings**, and then click **List Settings**.
5. In the **General Settings** section, click **Advanced Settings**.
6. In the **Read access** section, select **Only their own**.
7. In the **Edit access** section, select **Only their own**.

The following illustration shows the correct User Information List settings.

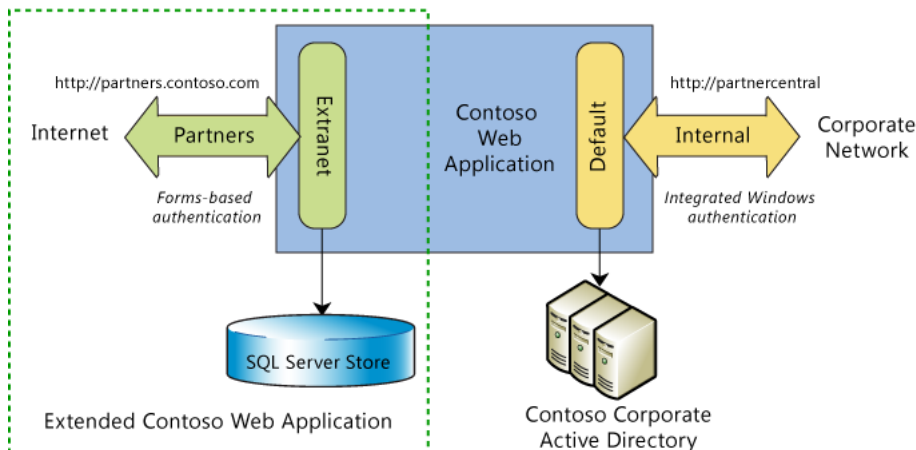
User Information List settings



How to: Expose a SharePoint Application to the Extranet and Use Forms-Based Authentication

If you want to share information between users who are within the corporate domain and external users, you must *extend* the SharePoint Web application to create an extranet-facing access point. Extending an existing SharePoint Web application provides a separate Internet Information Services (IIS) Web site. This Web site exposes the same content to all users, even if they are within different security domains. For example, extending the Partner Portal Web application provides access to the same collaboration areas, Contoso product catalog, and pricing information to extranet partners as to intranet employees.

Extending the Web application



When organizations create extranet solutions, they often group external users into a different security domain than internal users. This separation makes it easier to manage different groups of users and to maintain security. SharePoint supports this approach through *zones*. Each zone can support a different authentication method. Companies often use forms-based authentication (FBA) with Internet-facing zones because it is straightforward to set up and does not require additional hardware. FBA relies on the ASP.NET forms-based authentication framework. By using the ASP.NET 2.0 pluggable authentication provider model, SharePoint can support authentication for user identities that are stored in a Microsoft SQL Server database, in Active Directory (using Active Directory Application Mode, or ADAM), in an LDAP (Lightweight Directory Access Protocol) directory, or in any other source that implements an ASP.NET 2.0 membership provider.

The Partner Portal application uses the standard Microsoft SQL Server provider to store user credentials. Partners of Contoso authenticate to the extranet zone through FBA, while corporate users in the default zone rely on their existing accounts, which are contained in the corporate Active Directory store. For more information about the Partner Portal application's security approach, see [Security Decisions](#). For more information about choosing and implementing authentication on SharePoint, see [Plan authentication methods](#) on TechNet.

The following procedure is a brief overview of how to enable forms-based authentication. It includes references to more detailed procedures.

To enable forms-based authentication

1. Extend the SharePoint Web application. For information, see [Create or extend Web applications](#) on TechNet. The Partner Portal application uses the default security settings: **NTLM**, **Allow Anonymous** set to **No** and **Use Secure Sockets Layer (SSL)** set to **No**. In the **Load Balanced URL** section, set **Zone** to **Extranet**.
2. Configure the application to use forms-based authentication. For information, see [Configure forms-based authentication](#) on TechNet. Under **Application security for the Web application**, set the authentication provider to **Forms**.
3. Edit the Web.config file to register the authentication provider for the SharePoint Web application, the role provider for the SharePoint application and the membership provider (PeoplePicker) for the Central Administration site. For information about the Web.config modifications, see [Forms Authentication in SharePoint Products and Technologies](#) on MSDN.

Note:

When you install the Contoso Web application, the installation script creates the extended Web site, but it seems to extend the application to the Intranet zone. This is misleading, but it is only a labeling issue. The SharePoint STSADM [Extendysinwebfarm](#) command extends Web applications in the following order: Intranet zone, Internet zone, Custom zone, Extranet zone. To create the correct extranet label, every zone in between would have to be created, which significantly extends the installation time.

More Information

For more information about forms-based authentication and extending SharePoint Web applications, see the following articles on MSDN and TechNet:

- [Forms Authentication in SharePoint Products and Technologies \(Part 1\): Introduction](#)
- [Forms Authentication in SharePoint Products and Technologies \(Part 2\): Membership and Role Provider Samples](#)

- [Forms Authentication in SharePoint Products and Technologies \(Part 3\): Forms Authentication vs. Windows Authentication](#)
- [Configure forms-based authentication \(Office SharePoint Server\)](#)
- [Create or extend Web applications \(Windows SharePoint Services\)](#)

Using the User Profile Store

Many assets and services in the Partner Portal application, such as the Partner Collaboration site collection and the pricing service, rely on the association between a partner ID and a user profile to ensure that partners see only the information that is intended for them. The Partner Portal application stores the partner ID in the SharePoint user profile store. When a new partner wants to use the application, you must create a profile for the user and enter the appropriate value in the **PartnerId** field. The Partner Portal installation script creates and populates several user profiles. The following code shows how to add new property to the user profile store.

```
C#
ServerContext serverContext = ServerContext.GetContext(spSite);
UserProfileManager userProfileManager = new UserProfileManager(serverContext);

// Add the PartnerId profile property.
try
{
    Property property = userProfileManager.Properties.Create(false);
    property.Name = "PartnerId";
    property.DisplayName = "Partner Id";
    property.Type = PropertyDataType.String;
    property.Length = 50;

    userProfileManager.Properties.Add(property);
}
catch (DuplicateEntryException ex)
{
    // Catch the exception and do nothing if the PartnerId profile property
    // is already created.
}
```

The following code creates a new user profile for a specific user account and partner ID.

```
C#
private static void CreateUserProfileAndPartnerId(UserProfileManager userProfileManager,
string accountName, string partnerId)
{
    UserProfile userProfile =
        userProfileManager.CreateUserProfile(accountName);
    userProfile["PartnerId"].Value = partnerId;
    userProfile.Commit();
}
```

An instance of the **PartnerSiteDirectory** class reads the **PartnerId** property from the user's profile. The following code shows the **GetPartnerIdFromUserProfile** method of the **PartnerSiteDirectory** class.

```
C#
[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Performance",
"CA1822:MarkMembersAsStatic")]
private string GetPartnerIdFromUserProfile(string accountName)
{
    UserProfile userProfile = null;
    ServerContext serverContext = ServerContext.GetContext(SPCContext.Current.Site);
    UserProfileManager userProfileManager = new UserProfileManager(serverContext);

    if (!userProfileManager.UserExists(accountName))
    {
        ThrowPartnerIdNotInUserProfileException(null);
    }

    userProfile = userProfileManager.GetUserProfile(accountName);
    if (userProfile["PartnerId"] == null)
    {
        ThrowPartnerIdNotInUserProfileException(null);
    }
    return userProfile["PartnerId"].Value.ToString();
}
```

Partner Portal Reference Implementation

The Partner Portal Reference Implementation (referred to as the Partner Portal application in this guidance) demonstrates how to use SharePoint to create an extranet facing, enterprise-scale, content-driven application. The following are some of the activities that external partners can perform using the Partner Portal application:

- View information from Contoso such as news and promotions.
- Navigate the product categories and view products by category.
- View product information.
- View product details and related information such as specification sheets and pricing. The pricing includes discounts that reflect contractual agreements. (This is only partially implemented.)
- See special offers and their details.
- Collaborate with Contoso on high-priority incidents.
- See a summary of the incidents that are currently open and the incidents that are closed.
- Collaborate with Contoso to resolve order exceptions.
- See an order exception's current status, manage outstanding tasks, and exchange information with Contoso.

The following are some of the ways that Contoso employees can use the Partner Portal application:

- Account managers can use the partner collaboration portal home page to edit content that is intended for specific partners.
- Account managers can view outstanding tasks for all their partners to determine what needs to be done to resolve problems.
- Account managers can navigate to a partner's portal from a central location.
- Employees can use images and video to create promotions that are intended for a specific set of partners. Pricing can be different for different partners.
- Sales managers can review and approve promotions before they are released. (This is not implemented.)
- Customer service representatives can use the standard incident management system client to create a collaboration space. (In the Partner Portal application, this is implemented as a stand-in .aspx page.)
- Customer service representatives and service engineers can collaborate with a partner to exchange information and coordinate tasks in order to resolve open incidents.
- Customer service representatives can close incidents in the incident management system. The partner can see historical information about the incident.

Architectural Decisions

The Partner Portal application is a relatively complex SharePoint application. It demonstrates many SharePoint features that are used by enterprise-scale applications. These features include the following:

- It can be used to provide an extranet for external organizations. The extranet includes collaboration sites.
- It can be used to publish and deploy content.
- It can be used to integrate line-of-business (LOB) information with the application and create published content that includes LOB information.
- It can be used to aggregate and cross-link information and navigation across SharePoint sites.
- It can be used to demonstrate best practices for performance and manageability issues.

This section explains some of the design choices that dictated the Partner Portal application's architecture. Many of the individual decisions are covered in greater depth elsewhere in the guidance, and this topic does not discuss every decision. In particular, it does not give detailed information about the design patterns that were implemented. For more information about patterns, see [Design Patterns](#).

Instead, this topic discusses the following high-level considerations:

- [Site Design Overview](#). This section discusses the overall design of the Partner Portal site.
- [Security Decisions](#). This section discusses the decisions about the authentication and authorization processes.
- [Integration Decisions](#). This section discusses how the Partner Portal application was integrated with external business systems.

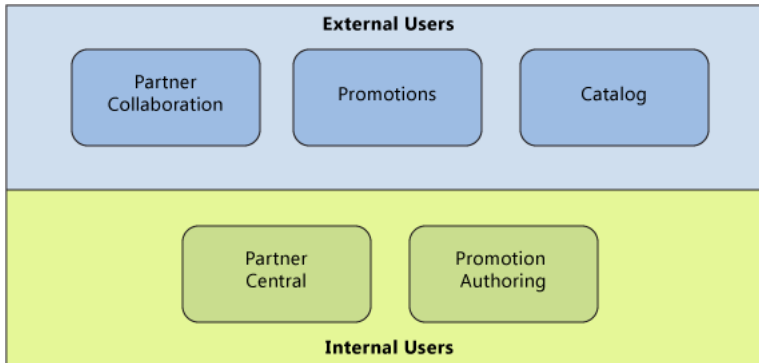
For more information about designing extranet sites, see [Planning an Extranet Environment for Office SharePoint Server](#).

Site Design Overview

The Partner Portal application is comprised of cooperating sites and site collections. However, from a user perspective, the application has a unified structure. Consistent branding provides a uniform appearance and behavior (look and feel), and navigation allows users to move from one functional area to another as if everything were contained in a single site. In fact, the Partner Portal application is divided into five areas. This structure is used for security, scalability, and functional reasons.

The following illustration shows the Partner Portal's functionality. External users are Contoso partners, and internal users are Contoso employees.

Partner Portal application's areas of functionality



The following summarizes the Partner Portal applications functional areas. They are discussed in detail later in this topic. The areas are the following:

- **Partner Collaboration.** This area provides a space for partners to collaborate with Contoso employees. The partner collaboration space is built on a standard SharePoint team collaboration site. It includes a home page with content for the partner, general collaboration capabilities, and dedicated collaboration spaces for business events such as tier-3 incidents and order exceptions. Partners have their own site collections that contain their collaboration areas. The site collection boundary ensures that each partner's data is secure and easy to manage.
- **Partner Promotions.** This area provides a read-only site where partners can view product promotions that are offered by Contoso.
- **Catalog.** This area provides a read-only site where partners can browse the product catalog. They also see pricing information that is provided by an LOB system. The SharePoint Business Data Catalog (BDC) uses the BDC shared services provider (SSP) to expose the catalog information.
- **Partner Central.** This area provides internal users with a summary of information about all Contoso partners. This information is aggregated from all the partner site collections.
- **Promotion Authoring.** This area allows internal users to create, review, and approve promotions for partners before the promotions are deployed to the **Partner Promotions** functional area. The promotion authoring functional area is located on a separate SharePoint authoring farm. All the other functional areas are located on the production farm.

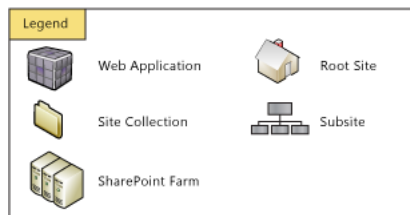
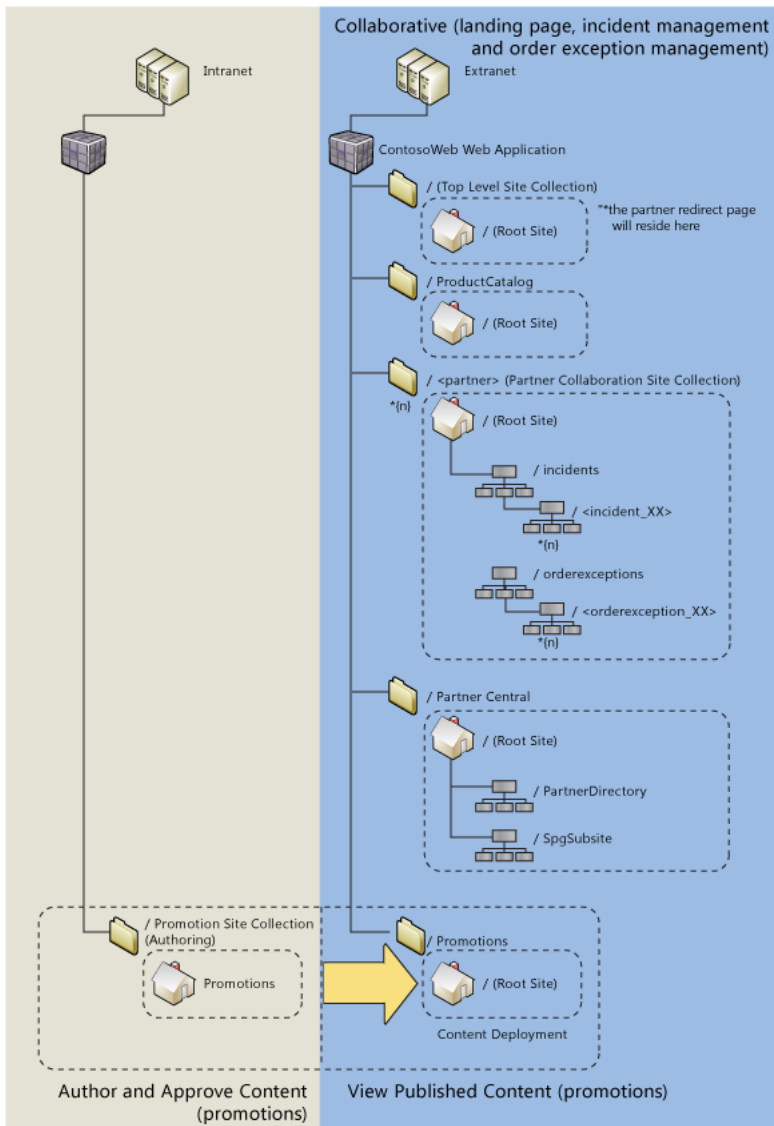
Note:

The set up script that accompanies the Partner Portal application does not configure the promotion authoring area for you. This is intentional. It reduces the complexity of the set up, and it allows you to install the application on a single system. If you want to install the promotion authoring functionality, see Deploying Content to the Production Farm.

Logical Site Design

The following illustration shows a detailed logical topology of the sites that comprise the Partner Portal application.

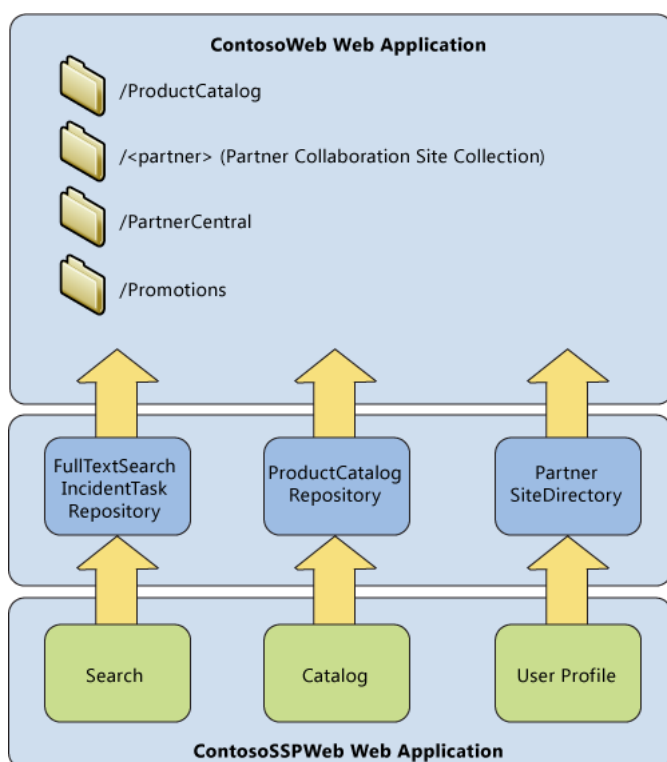
Logical topology of the Partner Portal application



The preceding illustration shows each area of functionality and its structure. The reasons for this structure are discussed in the following sections.

The following illustration shows the relationship of the ContosoWeb Web application to the shared services provider (SSP) Web application (ContosoSSPWeb). The SSP hosts a set of services that are used by the Contoso sites. For more information about extranet topologies, see [Design extranet farm topology](#) on TechNet.

Relationships between Web applications



The ContosoSSPWeb Web application provides search, catalog (BDC), and user profile services that are ultimately consumed by the ContosoWeb Web application.

Contoso Web Application

All the Partner Portal functions, except for promotion authoring, run within the ContosoWeb Web application. The Web application provides the security, process, and data boundaries for the site collections that are used by the Partner Portal. This includes the partner site collections, the promotion publishing site collection, the catalog site collection, and the partner central site collection. The authoring site collection is contained in a separate farm that is hosted within the corporate firewall. You can provide greater isolation between site collections by creating a Web application for each partner. However, this approach incurs greater administrative overhead, and the Partner Portal application does not require the extra security. The single Web application, in conjunction with SharePoint's permission infrastructure, is sufficient.

Although site collections are not actually hierarchical, the ContosoWeb application's site collection is logically located at the root URL. Therefore, it is the top-level site collection. A site collection includes a root site. By default, partners go to the Default.aspx page of the root site when they log on to the Partner Portal application. They are redirected to their collaboration site collection's root site. This means that partners do not need to remember a long URL to log on to their own site collection. For more information about using redirects, see [Application Instance Resolution](#).

Product Catalog

The product catalog site collection hosts the functionality that uses the BDC and Windows Communication Foundation (WCF) to access LOB catalog information. All partners see the same product catalog. This means that a single site can host the logic that grants access to it. SharePoint uses an SSP to access services that are exposed by the BDC. In the Partner Portal application, this SSP is named **ContosoSSPWeb**. The default configuration of an SSP exposes BDC profile pages from the SharePoint Web application that hosts the BDC shared service. An alternative approach is to expose the services from a site collection that is hosted by the extranet Web application. For example, the Partner Portal application exposes the profile pages on the product catalog site collection. You can control where profile pages are exposed with the SharePoint Central Administration. For more information, see [How to: Expose BDC Profile Pages](#).

Promotions

The Promotions site collection is a read-only publishing site that hosts product promotions. Content for the promotions site is created in the authoring farm and then deployed to the production farm, which is where the promotions site is located. A destination site should be read-only because the content deployment process expects a destination site to remain the same between deployments. To satisfy this requirement, the Partner Portal application separates the publishing site from the collaboration sites. For more information about content deployment, see [Understanding Publishing and Content Deployment](#).

Partner Collaboration

All partners have their own collaboration site collections. The site collection acts as a security and data boundary for collaboration activity. This data boundary is especially important because partners can store their own information on these sites in addition to exchange documents with Contoso. An alternative approach that is

appropriate in many situations is to use a central site collection and guarantee a user's privacy with access control lists (ACL). However, for collaboration sites, ACL permissions are more complex to manage, which increases the likelihood of incorrectly configuring permissions. In the Partner Portal application, partners can access the central product catalog site collection and the central Promotions site collection, but only their own collaboration site collections.

The Partner Portal application automatically uses the site creation workflows to create collaboration sites when particular event types occur, such as order exceptions and tier-3 support incidents. These sites integrate LOB information with the collaboration space. There is a root site for each business event type, and sites that are associated with the particular business event type are created under the root site. This creates an understandable information architecture with homogenous and logical groups of related sites. This approach is based on the Event-Driven Site Creation pattern. You can use a similar pattern to restrict where sites are created and the types of sites that are created under a particular node. For more information, see [Event-Driven Site Creation](#).

Partner Central

The partner central site collection is only exposed to internal users. This is a separate site collection that displays information about every Contoso partner. For example, internal users can see the tasks for all partners for tier-3 incidents. Internal users can also use this site to navigate to the collaboration area of a particular partner. This site collection also contains common areas of functionality that are used by all the partner collaboration areas. For example, the Microsoft Office SharePoint Server site directory that contains the partner directory is located here. The business logic that creates the sites related to business events uses the directory to resolve the location of a partner's site collection. The site directory also maintains and administers the site creation configuration data. Using a site directory as a central location for information about partner sites allows that information to be used in a variety of ways. For example, the site directory enables internal users to navigate to the individual partner site collections.

Promotion Authoring

Only internal users can author promotions. The promotion authoring site is located on a dedicated farm in the corporate intranet. This is a common configuration that is used when internally generated content must be reviewed and approved before it is deployed to an externally facing site. Isolating the authoring site within the corporate firewall provides an additional level of security.

There are other scenarios where content authoring is a collaborative process that involves external users. In these situations, it is appropriate to locate the authoring site in the perimeter network. To do this, you must use a different SharePoint Web application or even a separate SharePoint farm.

Integrated Site Behavior

One of the design challenges that the Partner Portal application addresses is how to make several sites appear to partners as a single integrated application. The Partner Portal application achieves this through the following design decisions:

- All sites have a consistent theme.
- All site collections are accessible with a custom global navigation scheme.
- All site collections, sites, and items are secured with a combination of roles and permissions. For more information, see [Security Decisions](#).
- All partners are automatically redirected to the appropriate sites.

An initial version of the Partner Portal application implemented global navigation by overriding entries in the site map definition that SharePoint uses for its standard navigation scheme. Although this technique is simple and appropriate for many scenarios, the final version of the Partner Portal application uses a custom navigation provider. The earlier approach required that some files be manually deployed to the Web front-end servers. It also meant that the global navigation was static instead of dynamic. For more information, see [Cross-Site Collection Navigation](#).

Security Decisions

The design of the Partner Portal application was strongly affected by security considerations. These considerations can be divided into the following areas:

- Design the logical site architecture and the network topology.
- Authenticate users to establish the end user identities.
- Assign users to zones and roles.
- Secure access to information that is contained within SharePoint.
- Secure access to information that is stored in line-of-business (LOB) systems. For more information, see [Integration Decisions](#).

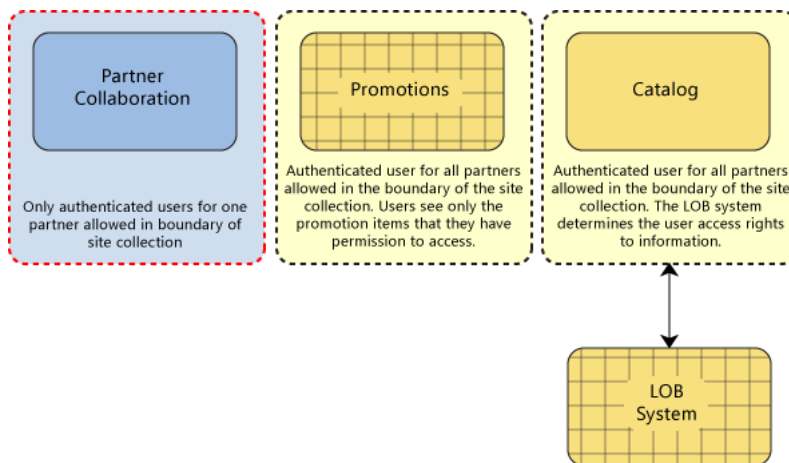
Designing the Logical Site Architecture and the Network Topology

Many of the decisions about the design of the Partner Portal application's logical site architecture were compromises between security considerations and manageability. The most stringent approach to security is to independently manage every partner in a separate SharePoint farm. This provides complete process, data, and physical isolation for each partner. However, this is a costly solution in terms of physical resources and maintenance. It is only appropriate in situations that require extremely secure environments.

A separate SharePoint Web application is another possibility. This provides process and data isolation when Web applications are assigned to dedicated Internet Information Services (IIS) application pools (otherwise, there is only application domain isolation instead of process isolation). Although this approach is less costly than maintaining separate farms, it is still costly in terms of maintenance and infrastructure costs. For most circumstances, a separate site collection is a good choice as an isolation boundary, especially when both internal and external users read and write information to the site collection.

The collaboration, catalog, and promotion areas each use different techniques to partition and secure information for external users. The following illustration shows this.

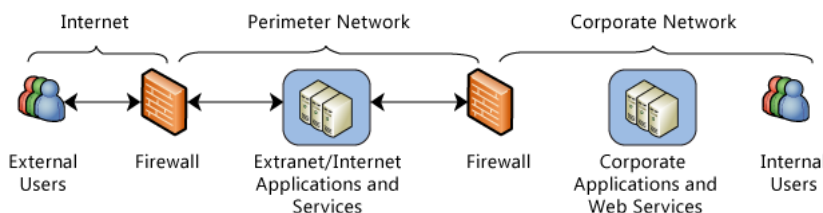
Securing functional areas



The decisions about the site architecture determined the possible options for securing data within a site or site collection. Promotions and the catalog are hosted in central site collections because the operational simplicity outweighed any additional security benefits.

The network architecture is a back-to-back perimeter network. For detailed information about the perimeter network, see [Understanding Publishing and Content Deployment](#). The following illustration shows how a perimeter network (also known as DMZ, demilitarized zone, and screened subnet) divides the network that is used by external users from the internal corporate network.

Using a perimeter network



Authenticating Users

Both external users and internal users must be authenticated. The Partner Portal application divides these users into two security areas:

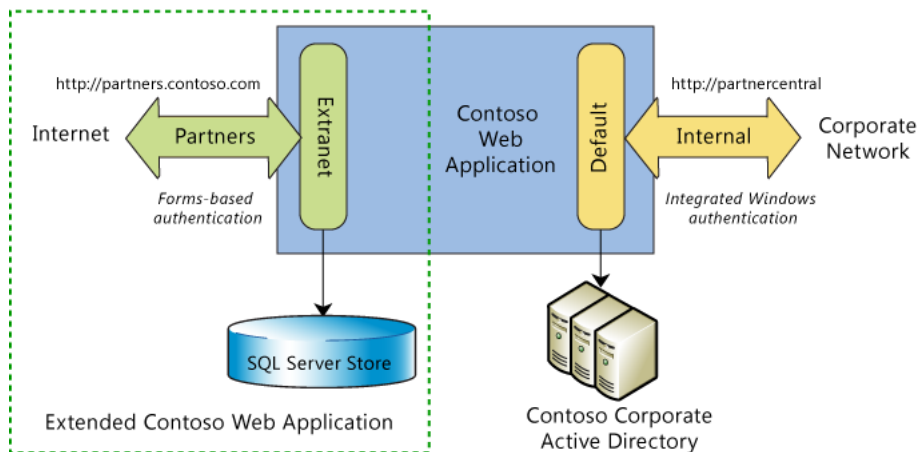
- **Internal security.** This is the corporate network; it uses Windows Integrated Security to authenticate users.
- **External security.** This is the perimeter network; it uses forms-based authentication (FBA) to authenticate

users.

The external users are only known in the perimeter network; the internal users are known in both the corporate network and the perimeter network. For simplicity, SQL Server is the store for the external users' credentials. In addition, the Partner Portal application uses roles to associate external users with partners. There is one role for each partner company, and all external users who are employees of that company are added to that role.

SharePoint supports zones. Zones enable users who are authenticated with different security mechanisms to access the same SharePoint site. For more information, see [Considerations for Extranet Development](#). The following illustration shows the authentication mechanisms that are used by the Partner Portal application.

Partner Portal authentication mechanisms



Typically, corporations use Windows integrated security for internal users because it provides a high level of security and network administrators already maintain the accounts that are stored in Active Directory. It is possible to use FBA with internal users in addition to the external users. This would allow Contoso employees to access the Partner Portal application outside of the corporate network. However, this places a burden on the Contoso network administrators and requires Contoso employees to remember their credentials for both their Windows accounts and their FBA accounts. To allow their employees external access to the Partner Portal application with only a Windows account requires that Contoso add additional hardware, such as Microsoft's [Intelligent Application Gateway](#).

Note:

For simplicity, the Partner Portal application uses local user accounts and does not show how to use Active Directory to manage user accounts management. However, the application did use Active Directory accounts within the server farm environment for scalability tests.

The following are other alternatives for authenticating external users that were considered:

- **Windows integrated security.** Windows integrated security requires additional hardware, such as Microsoft's Intelligent Application Gateway, to allow users in the perimeter network to use corporate credentials. In addition to the added hardware costs and operational complexity, external users would either need to be added to the corporate Active Directory or another domain would need to be established for external users in the perimeter network. There are many circumstances where this approach is a good choice, but it was overly complex for the Partner Portal application's business scenario. However, the following are some considerations for using the corporate domain or a dedicated domain:
 - Adding the external users' accounts to the corporate Active Directory adds to the Contoso network administrators' responsibilities. If many external users access the site, there can be an impact on the performance of the corporate Active Directory infrastructure. However, if there is a small set of external users, it may make sense to use the corporate domain and an application gateway.
 - If there are many users, establishing a separate Active Directory domain with an application gateway for external and internal users improves maintainability and scalability. This approach is relevant if there are complex security scenarios. For example, you can also use Active Directory's capabilities to establish trust relationships with other Active Directory forests. This scenario is beyond the scope of this guidance.
- **Forms-based authentication using Active Directory Application Mode (ADAM) for Windows Server 2003 or Active Directory Lightweight Directory Services (AD LDS) for Windows Server 2008.** These technologies are lightweight directory services that are intended for extranet users. AD LDS and ADAM use FBA, but they replace the SQL Server user store. This approach offers better integration with the SharePoint User Profile store. It also has better user management tools than the SQL Server membership provider. However, it is more difficult to set up and automate their membership stores than SQL Server's. The Partner Portal application uses the SQL Server store in order to fully automate the set up process. However, you should consider using AD LDS and ADAM in a production environment. For more information about using AD LDS and ADAM with SharePoint, see [Identity and Access Strategies for SharePoint Products and Technologies](#) on MSDN.

User Profile Integration

The Partner Portal application maps an external user's identity to a partner. Three ways to accomplish this were considered. The first approach that is discussed in the following section, using the SharePoint user profile store, is the one that the Partner Portal application uses. The other two approaches were rejected.

SharePoint User Profile Store

Using the Microsoft Office SharePoint Server user profile store, each user has a profile that can be extended to include custom attributes. The Partner Profile application extends the profile to include an attribute for the partner ID. This is an appropriate choice because there is a built-in user interface (UI) and an API to manipulate and extend the profile information. Also, there are additional capabilities that may prove useful if the application is upgraded in the future. An administrator must manually set up a profile if there is a new user. If either Windows authentication, ADAM, or AD LDS had been used for the credential store, it would have been possible to automatically synchronize a new user to the user profile store and avoid the manual steps. Note that the profile store is not available in Windows SharePoint Services.

The **PartnerSiteDirectory** class makes it easier to associate a user with a partner. It looks up the partner ID from the user profile store and then looks up the URL of the partner's site collection in the site directory. The Partner Portal application uses this information to redirect users to their partner home page. For more information, see [Application Instance Resolution](#).

Match Role to Partner

The Partner Portal application assigns users to roles that represent their affiliation with a partner. This is done to manage users' permissions to site collections, sites, and list items. Originally, the application iterated through the roles for a user and compared the role name with a mapping table that maintained the partner ID, the partner site URL, and the partner name in a central SharePoint list. This is the equivalent of an in-memory many-to-many join. This operation frequently occurred and could have impacted performance as the number of partners or number of roles grew. For example, if a user belonged to twenty roles, there could be as many as 20 lookups in the list. This was especially true for internal users who typically belong to many roles such as Builtin\users and Builtin\Administrators. This approach is also error-prone because the role name and the central mapping table needed to match. Another option was to use a naming convention for the role to explicitly identify a partner's role, but this approach was also prone to error.

Use a Separate Database or List

Another approach is to store the mapping between the user name and the partner ID in a separate list or database table. This requires a single lookup. This is similar to a user profile store, but it requires custom logic to create the equivalent of an extended attribute. There was no advantage to reproducing a capability that the SharePoint profile store already offers. A separate database or list is a reasonable approach if you use Windows SharePoint Services, which has no user profile store.

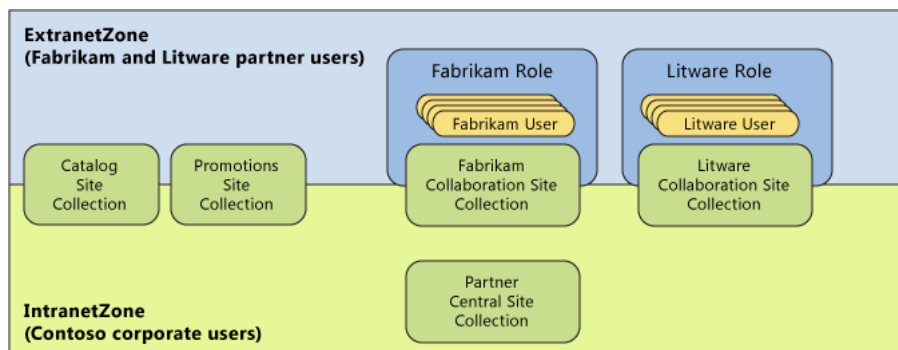
Assigning User Zones and Roles

A user's permissions are assigned based upon the user's zone and roles. The Partner Portal application has an intranet zone and an extranet zone. Users are assigned to one of these zones based on where they are authenticated. If a user is authenticated in the corporate Active Directory domain, they belong to the intranet zone. If they authenticated with FBA, they belong to the extranet zone.

Based upon their Windows permissions and group memberships, users who are authenticated in the intranet zone are assigned access permissions to their respective partner sites and the Partner Central site. A group membership is equivalent to an FBA role assignment. or convenience, the installation script for the Partner Portal application creates a Windows group for each partner. It then creates several Windows users and assigns them to these groups. All Windows users have permissions to access the Partner Central site. Anyone who belongs to the Administrator group can access all partner sites, as well as the Partner Central site. In a production environment, IT administrators would determine which groups can access the different sites.

In the current implementation, users are assigned a role that is specific to a partner. This role determines the permissions that grant access to particular sites. In some cases, no extranet users (in other words, no partners), are allowed access to a site or site collection. For example, only intranet users can access the Partner Central site. The following diagram uses two partners, Fabrikam and Litware, to show who can access different sites, based on the zone and the role.

Access permissions based on zone and role



Securing Access to Information Contained Within SharePoint

This section discusses how the Partner Portal application controls access to different types of information. It only discusses extranet users because Contoso employees have unlimited access.

Roles control the information that partners can access. In addition, SharePoint provides its own roles, such as contributor and reader. It is strongly recommended that you use groups or roles to assign permissions. SharePoint also supports permission inheritance within a site hierarchy. You should maintain this wherever possible. The

discussion of permission inheritance is beyond the scope of this guidance. For more information, see [Plan site security](#) on TechNet.

One of the important decisions about security was to determine the granularity of the permissions. An earlier section of this topic discusses how the site topology impacts the options for security. This is particularly true for ways to secure information. The following sections discuss how the collaboration sites, promotion site, and catalog site are secured.

The Collaboration Sites

Both partners and Contoso employees can read and write information in the collaboration sites. Contoso employees can access all collaboration sites, but partners can only access their own collaboration sites. This protects the privacy of one partner's information from other partners. In this case, permissions are determined at the site collection level. The criteria used is the following:

- Because partners can save information to the site, the Partner Portal application must ensure that the site cannot be compromised because of human or programming errors.
- Because partners can write to the site, and potentially attempt an attack, they can only access their own site collections.

Note:

You would additionally need to use a unique domain name to access each partner site collection to completely eliminate this risk. The Partner Portal application does not use this approach. For more information, see [Considerations for Extranet Development](#).

- Because permissions are set at the site collection level, the complexity of managing them is minimized.
- Because each site collection has its own site collection administrator, the responsibility of administering the site collections can be delegated to the appropriate Contoso employee. For example, if the partners' account managers are responsible for the site collections, they can manage all the settings, such as security or quotas.

The Promotions Site

There were several ways to manage the promotions site. All of them were compromises between ease of management, flexibility, and performance. Because partners can only read information on the site, security was less of a concern than on the collaboration sites. For more information, see "Techniques for Securing Published Information" in [Understanding Publishing and Content Deployment](#). The possible options were the following:

- The Partner Portal application could assign permissions to each promotion item that is stored in the promotions library page list. This approach is very flexible because any promotion can be assigned to any group of partners. However, promotion permissions must be managed on an item-by-item basis. In addition to the management overhead, filtering on a per-item basis has a performance impact. However, scalability tests showed that performance was adequate for up to several thousand promotions.
- The Partner Portal application could group together partners into different privilege levels, such as gold, silver, and bronze, and assign promotions to a page library for each level. For example, if there are three levels, there are three page libraries. Permissions are assigned to each library. Custom logic that uses configuration data would determine which library to use for a specific partner. This approach makes it simpler to manage permissions and has less impact on performance. The drawback is that it is less flexible, so this option was not used.
- The Partner Portal application could have used content deployment logic that is customized to filter the items in the promotions library while it was still on the authoring site. Only the promotions for a particular partner would be published to a specific page library. The page library would be secured for only that partner. This approach provides optimal performance and flexibility. However, the development and operational costs were too high.

The Catalog Site

Only authenticated extranet users can access the catalog site. The LOB system is responsible for authorization decisions. For more information, see Integration Decisions.

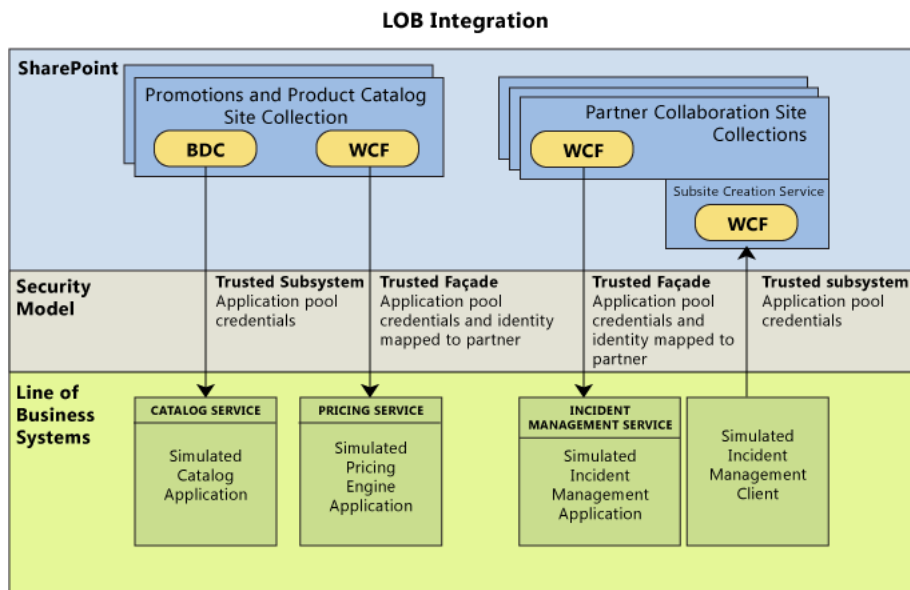
Integration Decisions

The Partner Portal application uses Web services to integrate information from several simulated line-of-business (LOB) systems. This makes the data that is available in these systems available to many more people than normal without compromising security. This topic describes two aspects of system integration: security and performance. The different types of information have different security requirements, so the means that were used to integrate with the different systems varies. In some situations, the Partner Portal application uses Microsoft Office SharePoint Server's Business Data Catalog (BDC). In other situations, it uses Windows Communication Foundation (WCF). For more information, see [Integrating Line-of-Business Systems](#).

Securing Integrated Services

The following illustration shows the areas of the Partner Portal application that are integrated with LOB systems and the security mechanisms that are used:

Secure LOB system integration



The BDC supports basic Web service capabilities. WCF supports more advanced Web service capabilities. The Partner Portal application uses two models for security, the trusted subsystem and the trusted façade.

Decisions about security were based on the following requirements:

- The LOB system should be protected from unauthorized access.
- It should be impossible to retrieve information that an end user is not authorized to view.
- The communication channel between the client and the server should be secure.

The product catalog is protected by a trusted subsystem. This is a commonly used and simple model for securing integrated services. It requires that the service trusts the identity of the calling application. The calling application identity is typically a process identity. For example, the Partner Portal application uses the process identity of its Internet Information Services (IIS) application pool. The service never sees the identity of the actual users who want to access the LOB information. The service trusts the calling application to properly protect the information and operations it provides. This approach is supported by the BDC.

The advantage of the approach is its simplicity. However, authorization decisions can be complex and difficult or impossible to represent outside of the LOB system. The trusted subsystem is appropriate when authorization decisions are straightforward and do not rely on the actual end user's identity. In the Partner Portal application, the catalog is available to all partners, so the trusted subsystem model is sufficient. As a result, the Partner Portal application can use the BDC to access the catalog. The trusted subsystem is also used for the service that creates sites for particular business events. Only internal calls from the LOB systems access this service, so this is another case where the trusted subsystem model is sufficient.

The pricing and incident management information is more sensitive than the catalog information. In this case, the SharePoint application, acting as the trusted subsystem, is not sufficient to apply pricing rules and to filter incidents. The additional security that is required to access the LOB systems relies on WCF features. Two design approaches were considered. They are the following:

- **Trusted façade.** This is the approach that the Partner Portal application uses. The approach is similar to the trusted subsystem in that the service trusts the identity of the SharePoint application pool. However, SharePoint also conveys the identity of the end user to the service. An alternative to providing the identity of the end user is to provide an identity that relates to the end user. For example, SharePoint may provide the actual user identity, a role identity for the user, or the identity of the partner for the user. The service uses this additional information to apply more specific authorization rules. In the Partner Portal application, the pricing service receives the partner identity for the user and applies pricing rules according to the contract that Contoso has with that partner. Note that no passwords are stored on the SharePoint tier. For more information about the Trusted Façade pattern, see [The Trusted Façade Pattern](#).

- **Single sign-on (SSO).** This approach stores credentials that are required to access the Web service (or database) in an SSO store. SSO maps a user identity to those credentials. Microsoft Office SharePoint Server has an SSO capability. However, it requires that a Windows identity be available to map to the stored credentials. The Partner Portal application uses forms-based authentication (FBA), which does not establish a Windows identity for the user. Therefore, the SharePoint implementation of SSO could not be used.

Another possibility was to create a custom SSO that did not rely on the Windows identity. However, there were concerns about properly storing passwords in the SharePoint tier. The SharePoint implementation of SSO has undergone stringent analysis to ensure that the encryption and retrieval of the information was highly secure. Because it would be difficult to reproduce these efforts, the Partner Portal application used a trusted façade.

Another reason to use WCF may be your company's internal IT policy. Typically, the decision about how services are exposed is made by the system administrators. A custom SharePoint application may need to conform to the policies dictated by those administrators. WCF may be required because it has more security options and greater flexibility than basic Web services. Whenever possible, use the BDC because of its metadata and security management capabilities. WCF is a viable alternative when you require more advanced capabilities.

Performance Considerations

Performance issues are an important consideration when you integrate your application with LOB systems. Performance has several facets. Many LOB systems are not designed with Internet-scale performance capabilities. As you integrate information into externally facing sites, you may be substantially increasing the demand on the system and overwhelming it. This affects your internal users who rely on these systems to perform their jobs. Secondly, a line of business system often applies complex rules to its data. For example, it may perform many calculations to derive a price for a product. These computationally intensive operations may make the systems slow to respond. This can affect external users who use your site to access the information.

To make decisions about how to optimize performance, the Partner Portal application used simulated, slowly responding services for the product catalog and pricing LOB systems.

Generally, it is difficult to anticipate how to optimize your code. For example, scale and stress tests identified a serious performance issue when the Partner Portal application logged to the Universal Logging Service (ULS). This performance issue was fixed with a small code change, but there was no way to know in advance whether there would be any problems.

As a general approach, code optimization should be based on test results. However, some bottlenecks are easy to predict. LOB services are often an obvious case, especially if you also want to minimize the LOB system load. The stress and scale tests of the Partner Portal application identified areas, particularly those areas that involved slowly responding LOB systems, that benefitted from performance optimizations.

The Partner Portal application implements the Repository pattern to access BDC and WCF Web services and to access SharePoint lists. A repository provides a central location to implement optimized data access techniques, such as caching. The BDC cannot cache data that it has retrieved. However, the BDC does retrieve, cache, and pre-compile the metadata that describes the systems it accesses.

When you cache data, you must ensure that no authorization rights are compromised. The Partner Portal application keeps the cached, partner-related pricing content reasonably separate and secure from other partners. Cached data is partitioned by partner and resides on the SharePoint application tier. Because the cache is not secured by permissions, the repository logic had to duplicate the partitioning strategy. An error in the caching logic, or a compromised SharePoint process that allowed the cache to be arbitrarily accessed, would in turn compromise the confidentiality of the pricing information. These were considered acceptable risks for the business scenario that is addressed by the Partner Portal application. However, caching data may be too risky if the information is very sensitive. Also, if authorization rules in the LOB system are very complex, it may be impossible to implement logic within SharePoint to ensure the information is properly protected from unauthorized access. In this case, the information should not be cached.

A Note on Approach to Design

The evolving architecture of the Partner Portal application was related to agile development practices. Initial solutions were the simplest that would satisfy the business requirements. The application's capabilities expanded feature-by-feature. A feature had to be simple enough to be implemented in a single iteration (an iteration was two weeks long). Most of the business logic was unit tested. There were also automated continuous integration tests that ran frequently. This approach allows you to constantly refactor and evolve an application's architecture, while remaining confident of its viability.

User Stories

This topic lists some of the different user stories that motivated the design of the Partner Portal application. Although the Partner Portal application does not implement all the stories, they represent possible ways that an enterprise-scale, content-driven application can be used.

The following are some of the user stories:

- Account managers can navigate to a partner's collaboration site and edit the content.
- Account managers can view a summary of the outstanding tasks for all partners.
- Account managers can navigate to a partner's portal from a central location.
- Product managers or account teams can create promotions with images and video that target a set of partners. Prices can be different for each partner.
- Customer service representatives and service engineers can exchange information and coordinate work with a partner from within their standard incident management clients.
- Customer service representatives and service engineers can close incidents in the incident management system. The partner can see that the incident is closed and recorded.
- Order exception collaboration sites are automatically created by an enterprise resource planning system when order delays exceed 48 hours for orders of more than \$25,000.
- Organizations can easily add new collaboration sites to the partner portal.
- Information technology (IT) professionals can manage application and configuration settings in the server farm.
- IT professionals and developers can see understandable error messages that help them diagnose problems.
- IT professionals understand how the site performs under load and when more hardware will be needed.
- Organizations can minimize the amount of hardware that is required to run the site reliably.

Partner Portal Use Cases

This topic includes procedures for the following use cases for the Partner Portal application:

- [Configuring the Sites to Appear in the Left Navigation Bar](#). This is a preliminary procedure that makes the subsites visible in the left navigation bar.
- [Creating Incident Management Sites](#). This procedure creates incident management sites for Partner 1 and Partner 2. The procedure simulates the actions of a line-of-business system that automatically creates these sites when certain business events occur.
- [Creating an Order Exception Site](#). This procedure is similar to creating incident management sites.
- [Viewing the Product Catalog](#). This procedure shows how to browse the Contoso product catalog.
- [Viewing a Product Category and the Navigation Control](#). This procedure shows how to navigate through a product category in the catalog. It also shows the breadcrumb navigation control.
- [Viewing a Product Details Page](#). This procedure shows how to navigate to the details page for a particular product and how to check to see if parts are available.
- [Creating a Promotion](#). This procedure shows how to create and publish a new promotion page.
- [Attaching an Access Control List to the Promotion Page](#). This procedure shows how to control who can see the promotion page.
- [Searching Across Multiple Site Collections](#). This procedure shows how to aggregate information across multiple site collections. It first shows how to attach tasks to incidents and then how to search for those tasks.
- [Logging a Partner Discount Trace Message](#). This procedure shows how to find warnings in the event viewer and the Universal Logging System.

Configuring the Sites to Appear in the Left Navigation Bar

You must configure the incident and order exception sites to appear in the left navigation bar. By default, they are not shown. The following procedure shows how to do this.

To make the subsites visible

1. Navigate to <http://localhost:9001/sites/partner1/incidents/default.aspx>.
2. Click the **Site Actions** tab.
3. Click **Site Settings**.
4. On the Site Settings Page, click **Navigation**.
5. On the Site Navigation Settings page, select the **Show subsites** check box. Click **OK**.
6. Navigate to <http://localhost:9001/sites/partner1/orderexceptions/default.aspx>. Perform the same steps to make the order exception sites visible.
7. Perform the same steps for Partner2 incidents sites and order exception sites:
 - a. <http://localhost:9001/sites/partner2/incidents/default.aspx>
 - b. <http://localhost:9001/sites/partner2/orderexceptions/default.aspx>

Creating Incident Management Sites

This procedure shows how to create incident sites in the Partner 1 collaboration portal. This simulates the behavior of a line-of-business (LOB) system that automatically creates an incident subsite when particular business events occur. You can create two incidents for Partner 1. They will be located in the Partner 1 collaboration portal. You can repeat the procedure to create incident subsites for Partner 2.

To create incident sites for Partner 1

1. Open <http://localhost:8585/Contoso.LOB.Web/NewIncidentSite.aspx>.

Note:

This page may not open, depending on the security policy settings of your system. If this occurs, comment out the line `<trust lever="Full"/>` in the `\spgdropLocation\Source\ PartnerPortal\Contoso.LOB.Services\web.config` file.

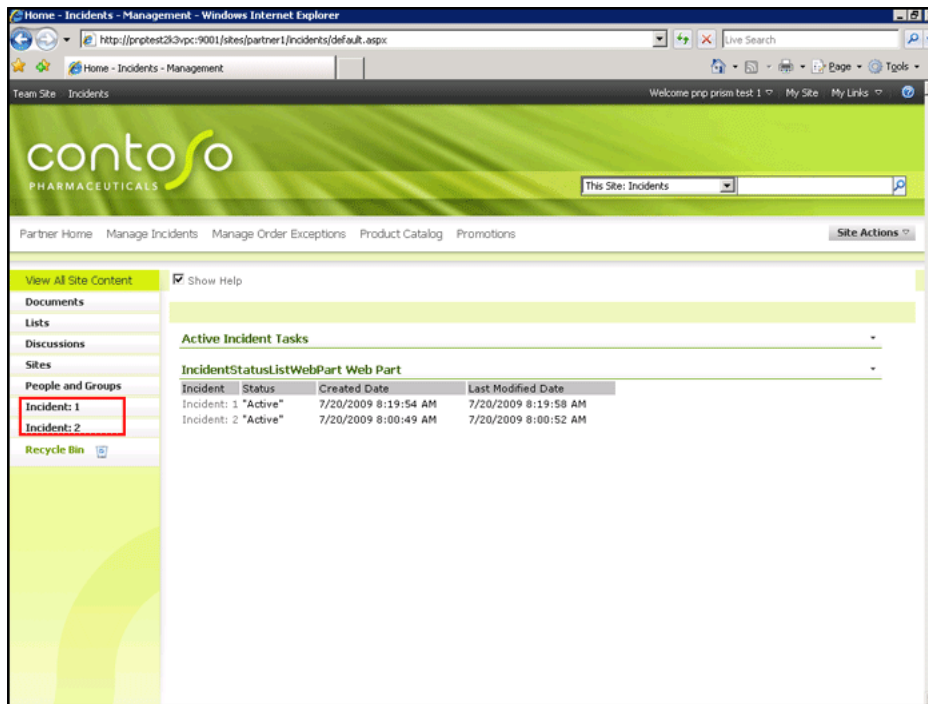
2. In the drop-down box, click **[Partner:ContosoPartner1][IncidentId:1] - X-ray machine: Installation issues**. Click **Create Site**.
3. In the drop-down box, click **[Partner:ContosoPartner1][IncidentId:2] - Contoso Blood Pressure Kit: Item damaged during delivery**. Click **Create Site**.
4. Repeat the procedure for the Partner 2 incidents.

Note:

Sometimes the workflow process may take several minutes. You can see workflow status by browsing to <http://localhost:9001/sites/PartnerCentral/SpgSubsite/Lists/SubSiteCreationRequests/AllItems.aspx>.

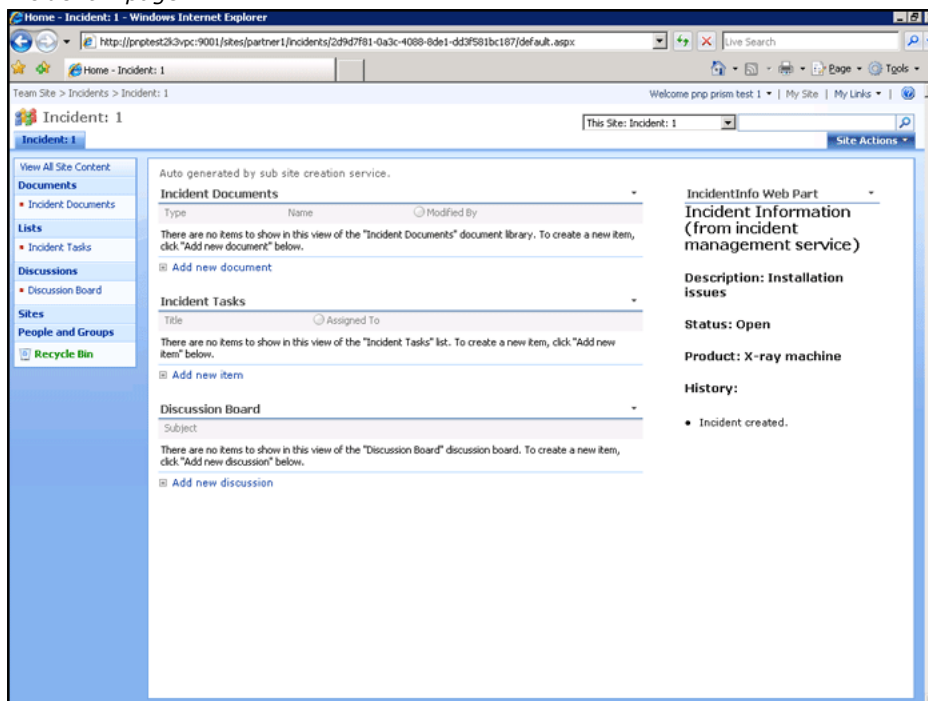
5. Verify that the incident sites were created and are visible to Partner 1. Open <http://localhost:9001/sites/partner1/> or log on to <http://servername:9002/sites/partner1/> as ContosoPartner1User1. Click **Manage Incidents**. The following illustration shows the incidents on the Partner 1 incidents site.

Partner 1 incident site



- Verify that the incident sites are in the Partner 1 site collection. Click either the **Incident 1** or **Incident 2** link. The Incident page opens. The following illustration shows the Incident 1 page.

Incident 1 page



- An alternative way to validate that the incident subsites exist is to view the All Site Content page. On the Partner 1 Incidents page, click **Sites** in the vertical navigation bar. The All Site Content page opens. It should display all of the incidents that you created.
- Repeat the procedure for Partner 2. Log on to <http://localhost:9001/sites/partner2/incidents/default.aspx> or to <http://servername:9002/sites/partner2/incidents/default.aspx> as ContosoPartner2User2 to verify that the sites were created.

Creating an Order Exception Site

The following procedure shows how to create an order exception site in the Partner 1 site collection. It is similar to the procedure in [Creating Incident Management Sites](#).

To create an order exception site

- Open <http://localhost:8585/Contoso.LOB.Web/CreateOrderExceptionSite.aspx>.
- In the **Partner** text box, type **ContosoPartner1**.
- In the **Order Exception Id** text box, type an alphabetical string. Click **Create Site**.

Note:

Workflow processes may take several minutes. You can see the workflow status by browsing to <http://localhost:9001/sites/PartnerCentral/SpqSubsite/Lists/SubSiteCreationRequests/AllItems.aspx>.

- Verify that the order exception site is in the Partner 1 site collection. Open <http://localhost:9001/sites/partner1/> or log on to <http://servername:9002/sites/partner1/> as ContosoPartner1User1. If you do not see the order exception, click **Sites**. The All Site Content page appears. It displays the incidents and the order exceptions. Click **Order Exceptions**.

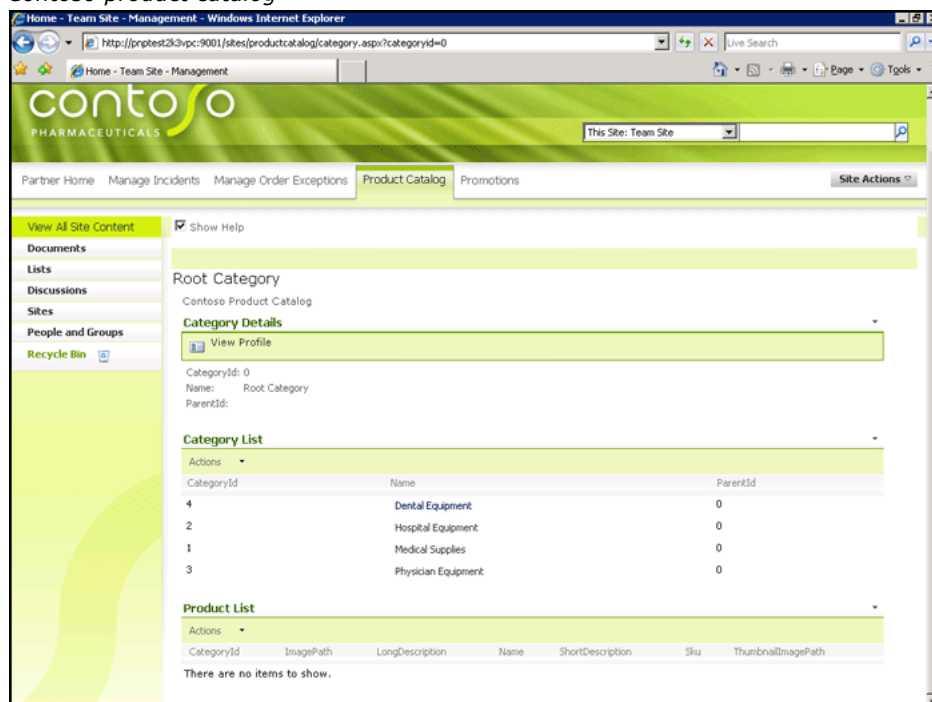
Viewing the Product Catalog

This procedure shows how to view the Contoso product catalog.

To view the product catalog

- Open <http://localhost:9001> or log on to <http://servername:9002/> as ContosoPartner1User1.
- Click **Product Catalog**. The product catalog page opens. The following illustration shows the catalog.

Contoso product catalog



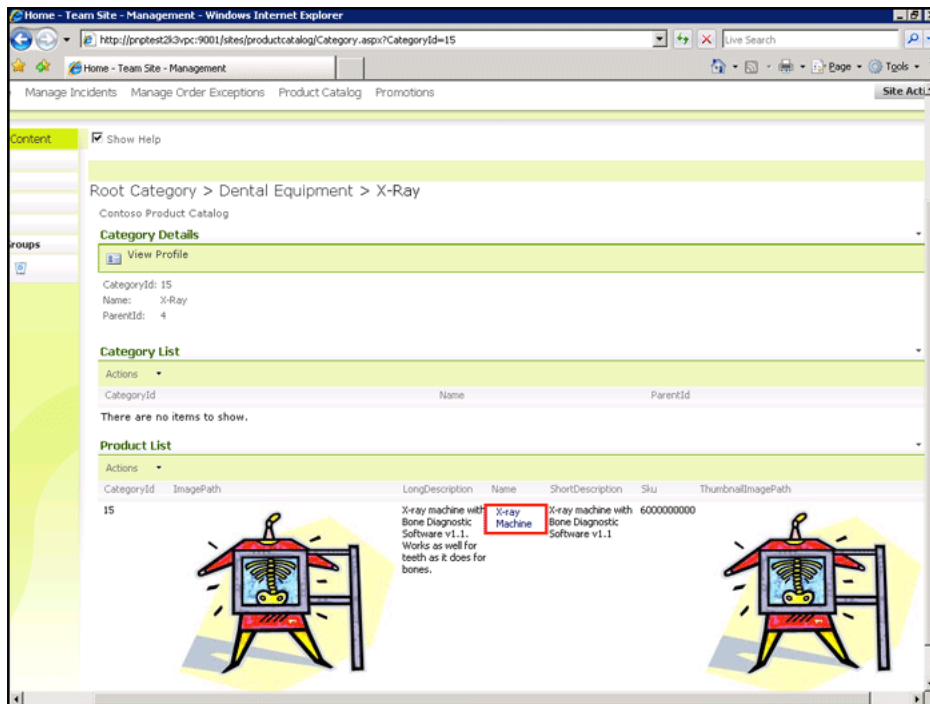
Viewing a Product Category and the Navigation Control

This procedure shows how to view a product category and displays the breadcrumb navigation control. The product catalog page should be open. (See [Viewing the Product Catalog](#).)

To view a product category and the breadcrumb navigation control

- In the Category List, click **Dental Equipment**.
- In Category List, right-click **X-Ray**. The page in the following illustration appears.

X-Ray catalog page



The bread crumb navigation shows the path from the root category to the X-Ray page.

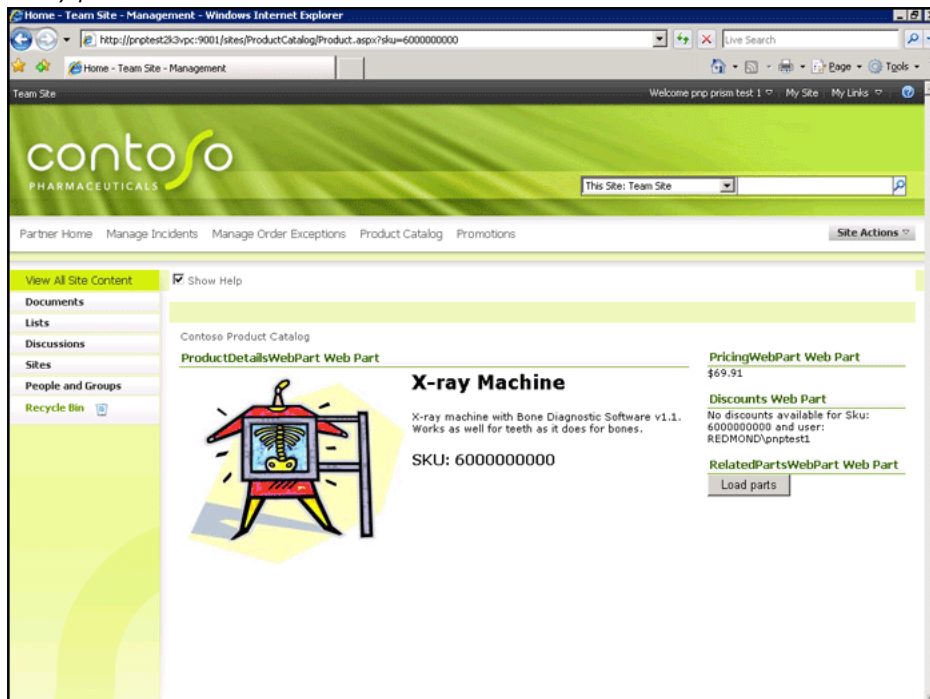
Viewing a Product Details Page

This procedure shows how to view a product details page. The X-Ray page should be open. (See [Viewing a Product Category and the Navigation Control.](#))

To view a product details page

1. In Product List, under **Name**, click **X-Ray Machine**. The Product Detail page appears. The following illustration shows this.

X-Ray product details



2. Click **Load Parts**. The message, "No parts found" appears.
3. Navigate to <http://localhost:9001/sites/ProductCatalog/Product.aspx?sku=1000000000>. The Blood Pressure Kit page appears. Click **Load Parts**. The list of parts shows "1 Part 1" and "2 Part 2".

Creating a Promotion

The following procedure shows how to create a promotion for a product.

To create a promotion

1. Navigate to <http://localhost:9001>. In the navigation bar, click **Promotions**.
2. Click **Site Actions**, and then click **Create Page**. The Create Page page appears. The following illustration shows this.

Page for creating promotions

3. In the combo box, verify that the following two layout pages are available:
 - (PromotionPage) PromotionLayout.aspx
 - (PromotionPage) PromotionLayoutVerticalStack.aspx
4. In the **Title** text box, type the name for the promotion. This example uses the name MyPromo1.
5. In the combo box, click **(PromotionPage) PromotionLayout.aspx**.
6. Click **Create**. The MyPromo1 design page opens. The following illustration shows this.

Design page for promotions

7. In the **Product SKU Field**, enter a valid SKU, such as 6000000000.
8. In the **Promotion Name Field**, click **Click here to add new content**, and then enter a name for the promotion, such as Contoso Promotion.
9. In **Promotion Image Field**, click **Click to add a new picture**. The **Edit Image Properties** dialog box appears. In this dialog box, do the following:

- Click the **Browse** button that is next to the **Selected Image** text box.
 - Click the **WWHO8_Doctors_around_System2** image. Click **OK**.
 - Click **OK**.
10. The MyPromo1 page opens with the new image.
 11. In the Zone 1 section, click the **edit** drop-down box. Point to **Connections**. Click **GetProduct SKU Connection Consumer From**. Click **Page Field Filter**. The Configure Connection dialog box appears. Select the filtered parameter **Sku**.

Note:

If the Configure Connection dialog box appears blank, follow these steps:

1. Close the Configure Connection dialog box.
2. Save the current promotion page. Click the **Check In to Share Draft** button.
3. Change the URL of the current page, replacing "localhost" with the machine name.
4. Click **Site Actions, Edit Page**.
5. Repeat step 11.
12. Perform the previous step for the Zone 2 Product Pricing and Promotion Discounts Web Parts.
13. In the **Filtered Parameter** drop-down box, click **Sku**. Click **Finish**. This retrieves the product details.
14. Add the Silverlight control. On the MyPromo1 page, in the **Windows Media Field** text box, enter /sites/Promotions/Movie Libraries/DemoMedia/intro.wmv.
15. On the navigation bar near the top of the page, click **Publish**. The finished promotional page opens. Play the Silverlight control.
16. Navigate to http://localhost:9001/sites/partner1/. The new promotion should be listed under **My Promotions**.

Attaching an Access Control List to the Promotion Page

You can attach an access control list (ACL) to a promotion page to limit the partners who can see it. The following procedure prevents Partner 2 from seeing the promotion.

To attach an ACL to a promotion page

1. Navigate to http://localhost:9001/sites/Promotions/Pages/default.aspx. The Promotions page opens.
2. Right-click on the MyPromo1 drop-down box (or whatever name you gave the page in [Creating a Promotion](#)), and then click **Manage Permissions**.
3. Click the **Actions** drop-down box, and then click **Edit Permissions**. The **Message from webpage** dialog box appears. Click **OK**.
4. The Permissions: MyPromo1 page appears. Select the **ContosoFbaPartner2** check box and the **ContosoWinPartner2** check box. Click the **Actions** drop-down box, and then click **Remove User Permissions**.
5. The **Message from webpage** dialog box appears. Click **OK**. The Permissions: MyPromo1 page appears. Note that ContosoFbaPartner2 and ContosoWinPartner2 no longer appear on the page.

Searching Across Multiple Site Collections

These procedures show how to search for incidents. Contoso employees perform this action to see a list of all tasks for all incidents. The first procedure shows how to associate tasks with incidents. The second procedure shows how to search for those tasks.

Creating an Incident Task

The following procedure adds a task to Partner 1: Incident 1.

To create an incident task

1. Navigate to http://localhost:9001/sites/partner1/. In the left navigation bar, click **Sites**. On the All Site Content page, click **Incidents**. Click **Incident: 1**.
2. On the Incident: 1 page, click **Incident Tasks**. Click the **New** drop-down box, and then click **Incident Task**. The Incident Tasks: New Item page opens.
3. In the **Title** text box, enter a name for the incident. Click **OK**.
4. Repeat the procedure for other incidents that you have created. To create tasks for Partner 2 incidents, navigate to http://localhost:9001/sites/partner2/.

Searching the Site Collections for Incidents

The following procedure shows how to search in different site collections for incidents. All partners have their own site collections.

To search the site collections

1. Click **Start**, point to **Administrative Tools**, and then click **SharePoint 3.0 Central Administration**.
2. In the left navigation bar, click **ContosoSSP**. The Home page opens.
3. Under **Search**, click **Search administration**. The Search Administration page opens.
4. In the left navigation bar, click **Content Sources**. The Manage Content Sources page opens.
5. In the **Name** drop-down box, click **Local Office SharePoint Server Sites**, and then click **Start Full Crawl**. Note that a full crawl can take several minutes, depending how the size of the site.

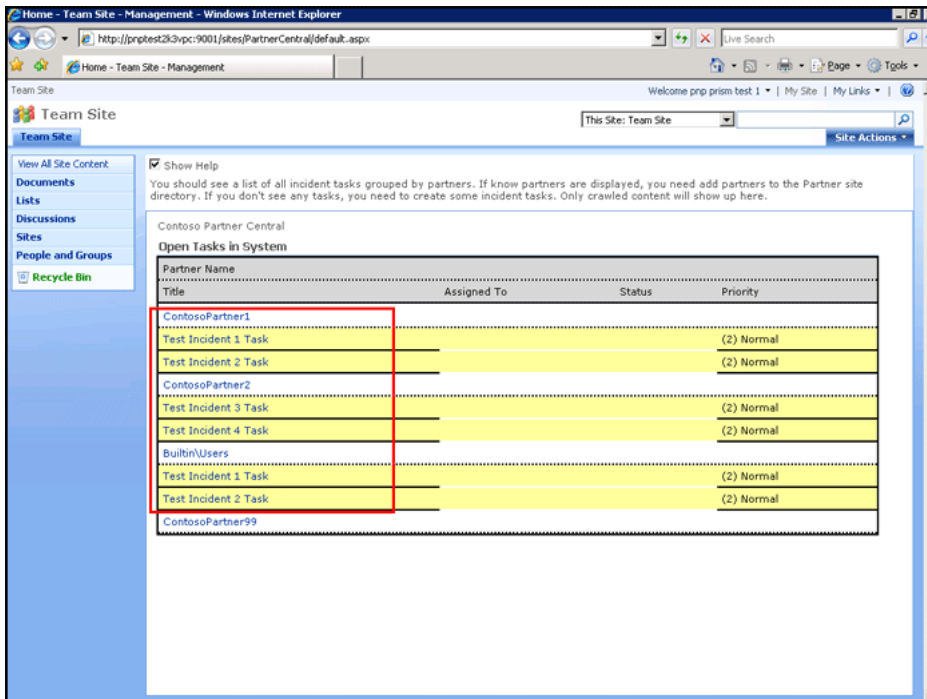
Note:

If the site has already been through a full crawl, you can select the **Start Incremental Crawl**. This only checks

for changes since the last crawl, so it is much faster than a full crawl.

- To view the results of the crawl, navigate to <http://localhost:9001/sites/PartnerCentral/default.aspx>. The following illustration shows a sample result.

Sample crawl results



The above search results do not include incident task status values. To include the status values in the search results, you must associate the status crawled property with a managed metadata property.

To include status values in the search results

- Click **Start**, point to **Administrative Tools**, and then click **SharePoint 3.0 Central Administration**.
- In the left navigation bar, click **ContosoSSP**. The Home page opens.
- Under **Search**, click **Search settings**.
- Under **Crawl Settings**, click **Metadata property mappings**.
- Navigate to the next page of managed properties and click the property **Status**.
- In the section **Mappings to crawled properties**, click the **Add Mapping** button.
- In the **Crawled property name** field, type "status" and click **Find**.
- Select **ows_Status(Text)** and click **OK**.
- Click **OK** on the **Edit Managed Property – Status** page.
- Follow the above instructions to start a full crawl and view the results of the crawl.

Logging a Partner Discount Trace Message

There are two logging operations and two locations that contain logging messages. The following table shows the operations and locations.

Operation	Location
LogToOperations	Event viewer and Unified Logging System
TraceToDeveloper	Unified Logging System

The following procedure shows how to write a trace message and then view it.

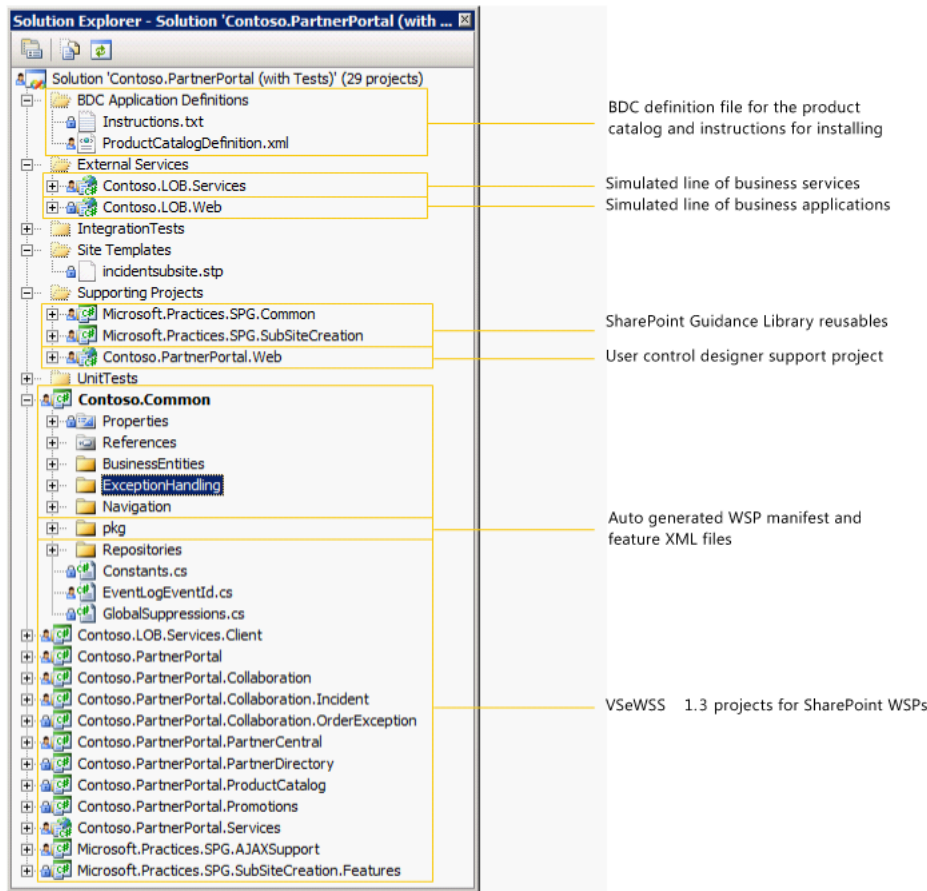
To create and search for a logging event

- Navigate to <http://localhost:9001/sites/ProductCatalog/Product.aspx?sku=6000000000>. This is the product details page for the X-ray machine. This creates a trace message that there are no discounts available.
- To see the log entry in the Unified Logging System, navigate to `C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\LOGS`.
- Open the most recent text file and search for "No discounts" to find the event.

Implementation View

The Partner Portal application uses the SharePoint Guidance Library and is based on the patterns and practices described in this guidance. The following illustration shows the structure of the Partner Portal application.

Solution Explorer view of the Partner Portal application



Contoso.PartnerPortal Solution

The Contoso.PartnerPortal solution contains approximately 30 projects. The following sections briefly describe each project and solution.

Web Solution Package Projects

This solution contains root-level projects that build Web solution packages (WSPs) for SharePoint. Each of these projects implements the functionality that is needed by a single associated SharePoint site. These projects use the Visual Studio extensions for Windows SharePoint Services version 1.3.

Each project includes a special auto-generated folder named pkg. This folder contains a file named Manifest.xml that lists WSP and Feature.xml files that are created by the project.

Note:

Projects that use Visual Studio extensions for Windows SharePoint Services 1.3 must not be in any solution folder. If it is, Visual Studio will throw an exception during compilation.

The WSP projects in the Partner Portal solution are the following:

- Contoso.Common.** This project includes classes that are used throughout the Partner Portal application. For example, the project defines business entities and interfaces for repositories. The Contoso.Common project also deploys and installs commonly used assemblies, such as Microsoft.Practices.SPG.Common and Microsoft.Practices.ServiceLocation.
- Contoso.LOB.Services.Client.** The repositories for the line-of-business (LOB) services are defined in this project. The proxy classes for the LOB services are also generated here. Other components use the Contoso.LOB.Services.Client component to connect to LOB services.
- Contoso.PartnerPortal.** Features intended for the root site collection of the Partner Portal application are implemented in this project.
- Contoso.PartnerPortal.Collaboration.** Features intended for the root Web of a partner's site collection are implemented in this project.
- Contoso.PartnerPortal.Collaboration.Incident.** Features for managing incidents are implemented in this project.

- **Contoso.PartnerPortal.Collaboration.OrderException.** Features for managing order exceptions are implemented in this project.
- **Contoso.PartnerPortal.PartnerCentral.** Features for the central partner management site are implemented in this project.
- **Contoso.PartnerPortal.ProductCatalog.** Features for the partner product catalog are implemented in this project.
- **Contoso.PartnerPortal.Promotions.** Features for authoring and managing promotions are implemented in this project.
- **Microsoft.Practices.SPG.AJAXSupport.** A feature for enabling AJAX on a SharePoint Web application is implemented in this project.
- **Microsoft.Practices.SPG.SubSiteCreation.Features.** Features for managing automated subsite creation are implemented in this project.

SharePoint Services Project

This solution contains a root-level project named `Contoso.PartnerPortal.Services`. This project implements a Windows Communication Foundation (WCF) service that integrates the Partner Portal application with LOB systems.

Projects for External Services and Applications

This solution has a folder named `External Services` that contains two projects. These projects define the simulated LOB applications that are used by the Partner Portal application. They are the following:

- **Contoso.LOB.Services.** This project implements the simulated product catalog, pricing, and incident management systems.
- **Contoso.LOB.Web.** This project implements the simulated incident and order exception management client applications. It also defines the Web pages that initiate the creation of the incident and order exception management subsites.

Integration Tests Project

This solution's `Integration Tests` folder contains a project named `Contoso.PartnerPortal.IntegrationTests`. This project implements integration tests that verify the behavior of Partner Portal components that run in a SharePoint instance.

Supporting Projects

This solution's `Supporting Projects` folder contains libraries and other components that are used by the Partner Portal application. These are the following:

- **Contoso.PartnerPortal.Web.** Visual Studio extensions for Windows SharePoint Services 1.3 project templates do not provide designer support for any of the ASP.NET user controls and page items. The `Contoso.PartnerPortal.Web` project uses the Visual Studio designer surface to create user controls that will be used in the Partner Portal application. The items in this project are linked as files into the Partner Portal WSP projects. This project is not directly used by the Partner Portal.
- **Microsoft.Practices.SPG.Common.** This project contains the Microsoft SharePoint Guidance Library.
- **Microsoft.Practices.SPG.SubSiteCreation.** This project implements the subsite creation workflow components.

Unit Testing Projects

This solution includes a folder named `UnitTests`. This folder contains unit test projects for the Partner Portal application and the SharePoint Guidance Library. The name of each unit testing project is the name of the project being tested plus the suffix `.Test`. For example, the unit tests for the `Contoso.Common` project are found in the `Contoso.Common.Tests` project.

Using Visual Studio extensions for Windows SharePoint Services When Implementations Are Separated from Interfaces and Base Classes

There are situations where an implementation class extends a base class or implements an interface that is not located in the current assembly. For example, the SharePoint Guidance Library is a separate assembly that defines interfaces that applications can implement. If you want to create your own logging class that uses the interface defined by the SharePoint Guidance Library, your new class will not be in the same assembly as the **ILogger** interface.

Test-driven development is another case where there are separate assemblies for interfaces and the classes that implement them. The separation allows you to substitute the dependencies of the object you want to test with mock objects at run time.

Some of the projects in Partner Portal application implement interfaces that are located in other assemblies.

When Visual Studio extensions for Windows SharePoint Services 1.3 creates WSPs, it assumes that any interface or base class is either in the same assembly as its derived classes or is in an assembly that is located in the global assembly cache.

If you implement an interface that is located in another assembly (and that assembly is not in the global

assembly cache), you will receive the following error message if you use Visual Studio extensions for Windows SharePoint Services 1.3 to create a WSP:

"Error 1VSeWSS Service Error: Unable to load one or more of the requested types. Retrieve the **LoaderExceptions** property for more information."

To resolve this problem, place the assembly that contains the interfaces and base classes into the global assembly cache.

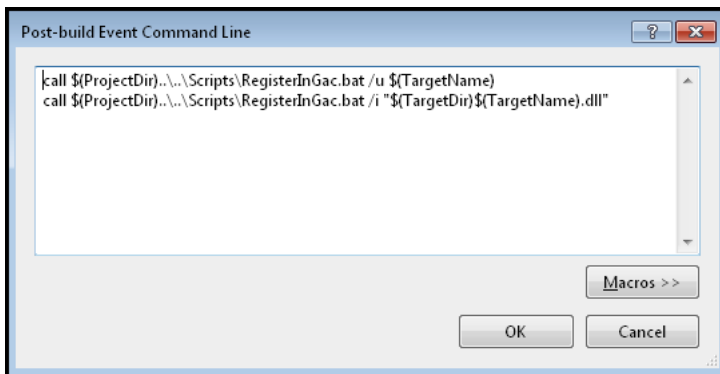
Creating a Post-Build Script

The Partner Portal application has a post-build script that uses the Gacutil.exe program to place the assemblies into the global assembly cache. The script accounts for the fact that the Gacutil.exe program is not in the same location on every system. The following batch file, which is named RegisterInGAC, attempts to locate the Gacutil.exe program. If it finds the program, it displays the program's location and then uses the program to install the assembly into the global assembly cache.

```
set gacutil=%programfiles%\Microsoft SDKs\Windows\v6.0A\Bin\gacutil.exe
echo *** RegisterInGac.bat %1 %2 echo * Trying to find GacUtil.exe:
if not exist "%gacutil%" echo * Could not find GACUtil at: %gacutil% if not exist
"%gacutil%" set gacutil=%ProgramW6432%\Microsoft SDKs\Windows\v6.0A\Bin\gacutil.exe if not
exist "%gacutil%" echo * Could not find GACUtil at: %gacutil% if not exist "%gacutil%" goto
error
echo * Using gacutil.exe from %gacutil%
call "%gacutil%" %1 %2 goto done
:error echo * Warning: Could not (un)register %2 in the gac.
:done
```

To run the batch file, you need to add a post-build event to your project. The following illustration shows the post-build event command line.

Post-build event commands



Note:

Assemblies in the global assembly cache always take precedence over assemblies that are located in the file system. Use the batch file as part of a post-build step to ensure that the assembly in the global assembly cache is always the latest version.

Partner Portal Application Implementation Details

This section includes explanations of how various aspects of the Partner Portal application was implemented. These topics are the following:

- [Repositories in the Partner Portal](#). This topic explains what a repository is and how it is designed and implemented in the Partner Portal application.
- [Product Catalog](#). This topic shows how the Partner Portal application uses the Business Data Catalog (BDC) to expose product information from a back-end catalog management system to partners.
- [Incident Dashboard](#). This topic explains how the Partner Portal uses the Content Query Web Part to perform a cross-site query across lists that are found in subsites of the current partner's site collection.
- [Logging and Tracing in the Partner Portal](#). This topic shows how the Partner Portal application uses the SharePoint logger that is provided by the SharePoint Guidance Library.
- [Product Catalog Search Configuration](#). This topic shows how to configure the SharePoint Search account and how to search the Contoso product catalog.
- [Subsite Creation Configuration](#). This topic explains how The Partner Portal application uses a workflow to programmatically create collaboration sites for partner incidents.
- [New Partner Provisioning](#). This topic explains how to add a new partner to the Partner Portal application.

Repositories in the Partner Portal

This section describes the implementation of the Repository pattern in the Partner Portal application. It also includes an overview of the business entities used by the Partner Portal application.

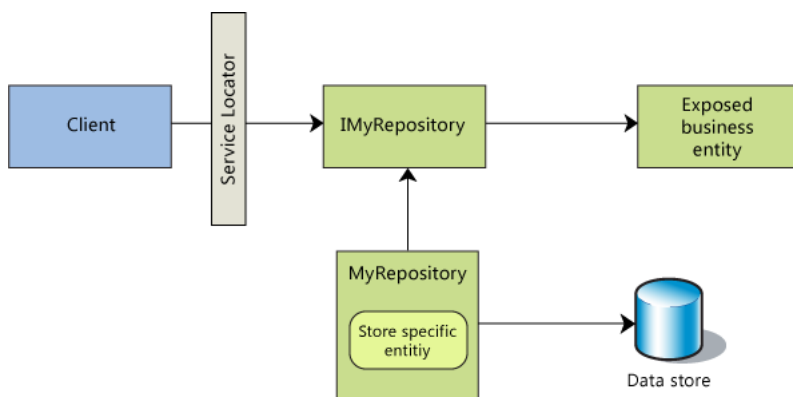
A repository is a design pattern that separates a data source from its associated business logic by encapsulating the data access details. Repositories translate the underlying data representation into an entity model that fits the problem domain. The Partner Portal application uses the Repository pattern to encapsulate business entities used by the application. To do this, it makes use of the helper classes provided by the SharePoint Guidance Library.

For more information about the Repository pattern, see [The Repository Pattern](#). For more information about the repository-related classes provided by the SharePoint Guidance Library, see [List-Based Repositories](#).

Repositories in the Partner Portal Application

The following illustration shows the high-level design of the repositories that are implemented by the Partner Portal application:

High-level design of repositories



The client, such as a Web Part, requests a repository from the service locator. The client relies solely on the interface of the repository. This makes it possible to unit test the client in isolation. The test environment can provide a mock object that stands in for the repository.

The repository interface exposes business entities. Business entities are strongly typed classes that represent data records. Examples are the **Product** and **Customer** class. These classes do not contain any code about how the objects are stored and retrieved from the data store.

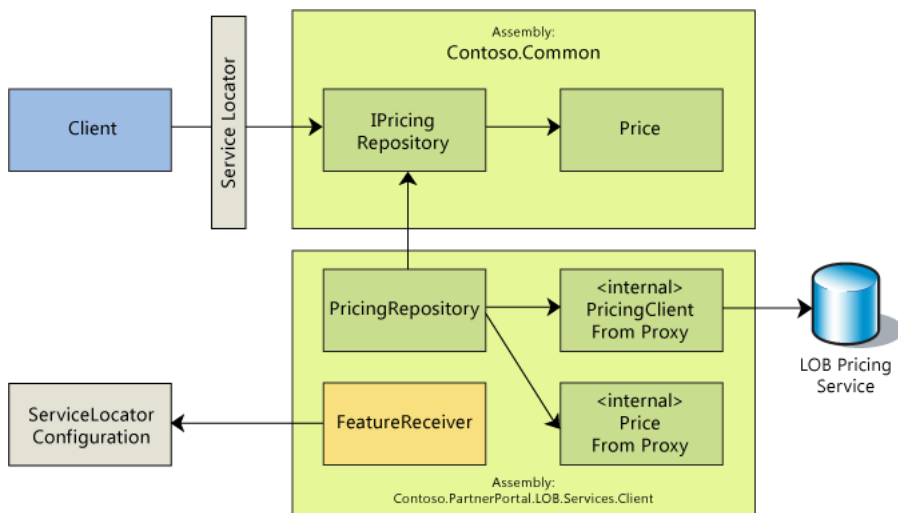
The repository's implementation interacts with the underlying data store. The repository can query a table in a database, update items in a SharePoint list or invoke Web services. The choice of data store determines the mode of data access. For example, to access items in a SharePoint list, you must interact with **SPLListItem** objects. The repository maps these store-specific entities to and from the strongly typed business entities.

Repositories reduce the complexity of accessing the data store. For example, when calling Web services, a repository can perform the authentication instead of having the client code do it. Repositories can also implement caching to improve performance.

Repositories are hard to unit test because they are tightly integrated with the underlying data store. It is possible to use mock objects that simulate behavior of the underlying database or SharePoint list, but this requires a large investment in test code. For this reason, many developers choose to test repositories with integration tests.

The following illustration shows an example of a repository implementation from the Partner Portal application.

Design of the Partner Portal's pricing repository



In this example, the interface and business entities of the repository are defined in the **Contoso.Common** assembly. The implementation is defined in the **Contoso.PartnerPortal.LOB.Services.Client** feature. When this feature is activated in SharePoint, the feature receiver calls the **ServiceLocatorConfig** class and maps the **IPricingRepository** interface to the **PricingRepository** class.

Splitting the repository's interface and the implementation of the repository into different assemblies is not required. However, if you do this, the client of the repository no longer requires a reference to the assembly that implements the repository. This is useful in the following situations:

- You want to develop, version, and deploy the feature that holds the repositories separately from the client. For example, if the repository accesses a Web service, you do not need to recompile the client code if the Web service changes.
- You want to be able to easily replace the implementation in the repositories. For example, in one environment you want to retrieve information from a database, but in another environment you want to retrieve the same information from a Web service.

Note:

Separating the repository implementation from its interface adds complexity. There are many situations where this is not required.

The Partner Portal Data Model

The **Contoso.Common** namespace contains classes, interfaces, and enumerations that provide a conceptual data model that is used by the Partner Portal application. This section discusses the repository interfaces and the business entity classes.

The **Contoso.Common.BusinessEntities** namespace contains business entity classes that represent many of the data elements that are used by the Partner Portal application. Repository interface methods return instances of Contoso business entity classes. Repository interfaces are found in the **Contoso.Common.Repositories** namespace.

Partner Portal Business Entity Classes

Business entities are declared in the **PartnerPortal\Contoso.Common\BusinessEntities** directory. It contains the following classes:

- **Product** class. Each Contoso product has a unique stock-keeping unit (SKU) in addition to other properties, such as a name and a description.
- **Category** class. Products are organized by a hierarchy of categories. Each product has a category. Each category, except for a special root category, has a (more general) parent category.
- **Price** class. Product pricing occurs on a per-partner basis. A product's price depends on the partner ID.
- **Discount** class. Discounts are part of promotions that occur from time to time. Discounts are specific to a particular partner and product. There can be more than one discount for each partner and product pair.
- **Incident** class. Incidents represent issues, such as the need for a replacement part, that require a high level of interaction between Contoso customer support and a partner. A SharePoint collaboration Web site is created for each active incident.
- **IncidentTask** class. Incidents can contain tasks. An example of a task is to send a replacement part to a customer.
- **Part** class. Some products are composed of parts.

Note:

There is no business entity class for partners and for the product/part relationship. The current partner ID is deduced from the run-time security context of the current user. The product/part relationship is an internal detail of the product catalog.

Partner Portal Repository Interfaces

There are four repository interfaces defined by the Partner Portal application. They are defined in the `PartnerPortal\Contoso.Common\Repositories` directory. These interfaces are the following:

- **IProductCatalogRepository** interface. The Partner Portal application uses the product catalog repository to look up a product using a product SKU and to enumerate products in a specified category. The application can enumerate subcategories of a category and parts that are used by a product.
- **IPricingRepository** interface. The Partner Portal application uses the pricing repository to look up a price that applies to a product in the current partner context. The repository can also return an enumeration of discounts that apply to a product in considering the current partner context.
- **IIncidentManagementRepository** interface. The Partner Portal application uses the incident management repository to look up an incident using an incident ID, write information to the incident history log and close an incident.
- **IIncidentTaskRepository** interface. The Partner Portal application uses the incident task repository to retrieve a list of all open incident tasks, for all partners.

Repository Implementations

The Partner Portal application uses the SharePoint service locator to map repository interfaces to their corresponding implementations. This is shown in the following code.

C#

```
IProductCatalogRepository productCatalogRepository =  
    SharePointServiceLocator.Current.GetInstance<IProductCatalogRepository>();
```

The SharePoint service locator's mapping between interface and implementation is configured during the execution of a feature receiver that installs LOB services. This is shown in the following code, which is located in the `PartnerPortal\Contoso.LOB.Services.Client\WebAppFeatureReceiver.cs` file.

C#

```
[CLSCompliant(false)]  
[Guid("8b0f085e-72a0-4d9f-ac74-0038dc0f6dd5")]  
public class UpdateServiceModelFeatureReceiver : SPFeatureReceiver  
{  
    // ...  
  
    [SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]  
    public override void FeatureInstalled(SPFeatureReceiverProperties properties)  
    {  
        ServiceLocatorConfig typeMappings = new ServiceLocatorConfig();  
  
        // ...  
  
        typeMappings.RegisterTypeMapping<IProductCatalogRepository,  
            CachedBdcProductCatalogRepository>();  
    }  
}
```

This code saves an association between the **IProductCatalogRepository** interface and an implementation class that is named **CachedBdcProductCatalogRepository**. The type mapping is saved in the content database using a property of the local SharePoint farm's property bag.

The SharePoint service locator allows the Partner Portal's components to be tested in isolation. Unit tests can replace the configured repository with a test-provided stand-in component. The Partner Portal unit tests use the **MockProductCatalogRepository** class for this purpose. The following code, taken from `PartnerPortal\Contoso.PartnerPortal.ProductCatalog.Tests\ProductDetailsPresenterFixture.cs`, demonstrates how the unit test replaces the configured component with a test-provided component.

C#

```
SharePointServiceLocator.ReplaceCurrentServiceLocator(new ActivatingServiceLocator()  
    .RegisterTypeMapping<IProductCatalogRepository, MockProductCatalogRepository>());
```

Note:

It is a recommended practice to place repository interfaces and their implementations in separate assemblies. This makes it easier to replace implementations without affecting client components. Separate assemblies also make unit testing easier.

You should be aware that the Visual Studio extensions for Windows SharePoint Services version 1.3 requires that dependent assemblies be located in the global assembly cache. The reason for this is that Visual Studio extensions for Windows SharePoint Services 1.3 individually sends each assembly to the Web service to package. The packaging service iterates over all the types in the assembly. If one of the types has a base class or interface in a different assembly, this packaging fails because it cannot find them. When the assembly with the base class is in the global assembly cache, the packaging can continue normally.

The standard build process does not automatically install assemblies into the global assembly cache. Therefore, if you place interfaces or base classes in a separate assembly used by a Visual Studio extensions for Windows SharePoint Services project, you should add a post-build action to install your assembly into the global assembly cache.

For example, the Contoso.Common project includes the following post-build command:

call \$(SolutionDir)..\Scripts\RegisterInGac.bat /u \$(TargetName)

The RegisterInGac batch file locates the gacUtil.exe command-line utility program on the local system and uses this utility program to install the newly built assembly into the global assembly cache.

More Information

For more information about the SharePoint service locator, see [The SharePoint Service Locator](#). For a walkthrough of the product data catalog, see [Product Catalog](#).

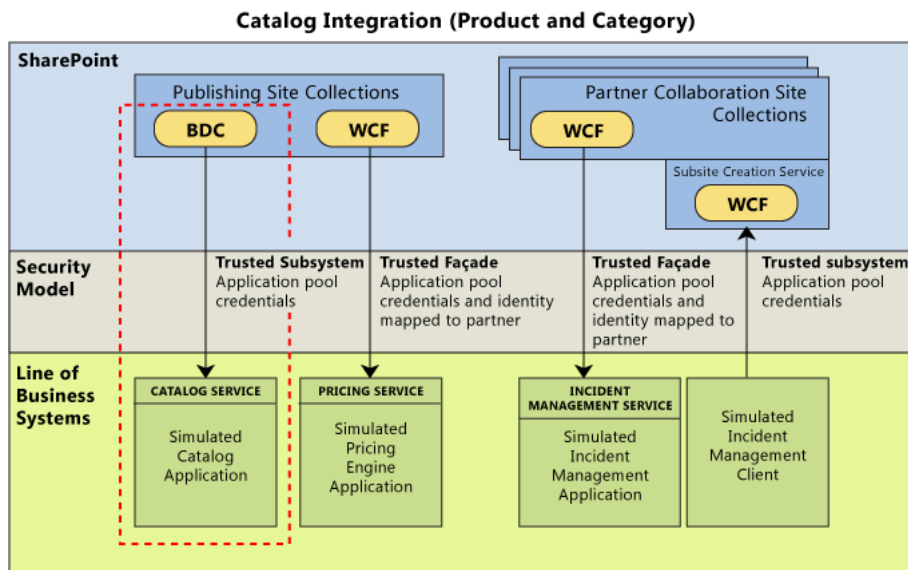
Product Catalog

This walkthrough demonstrates how the Partner Portal application uses the Business Data Catalog (BDC) to expose product information from a back-end catalog management system to partners. The BDC allows you to expose line-of-business (LOB) data from back-end systems simply by creating an XML file. You do not need to write additional code. This topic assumes that you have a basic understanding of the BDC. If you are new to the BDC, see [Integrating Line-of-Business Systems](#).

The Product Catalog Web service exposes read-only data for product categories and products. It is part of a simulated external product catalog application. Information in the product catalog is not partner specific. The same information is accessible to any authenticated partner. The Partner Portal application uses a trusted subsystem model to secure the Web service. This means that the Web service trusts the application pool identity that is used by the SharePoint Web application.

The following diagram illustrates how the data that is exposed by the BDC is integrated into the Partner Portal application.

Catalog integration

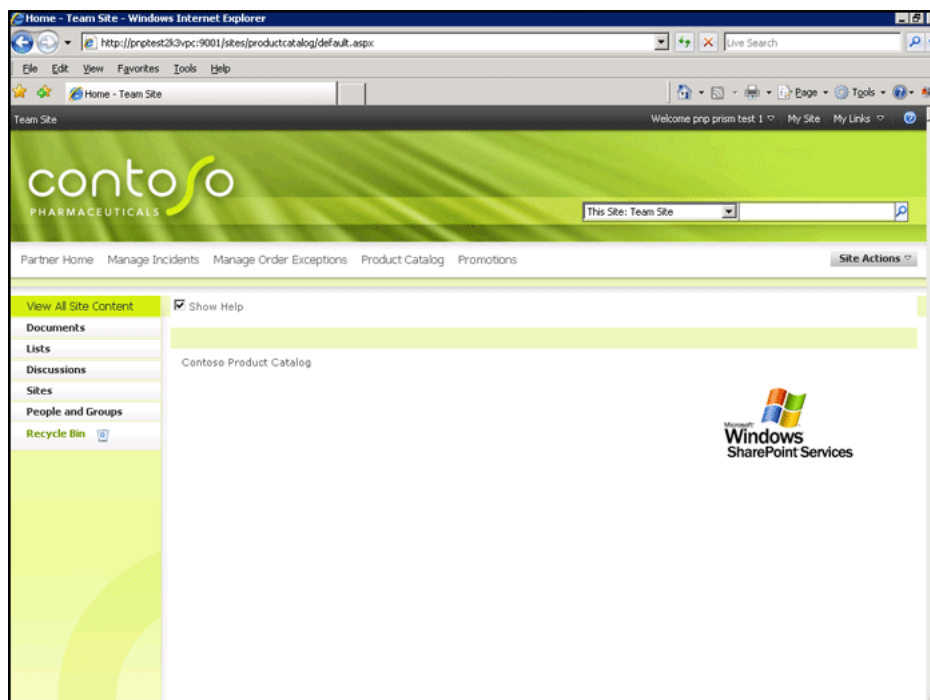


Using the Product Catalog

To see how the product catalog uses the standard BDC Web parts, open the URL http://<<Server_Name>>:9001/sites/productcatalog/default.aspx.

The following figure shows the Product Catalog page.

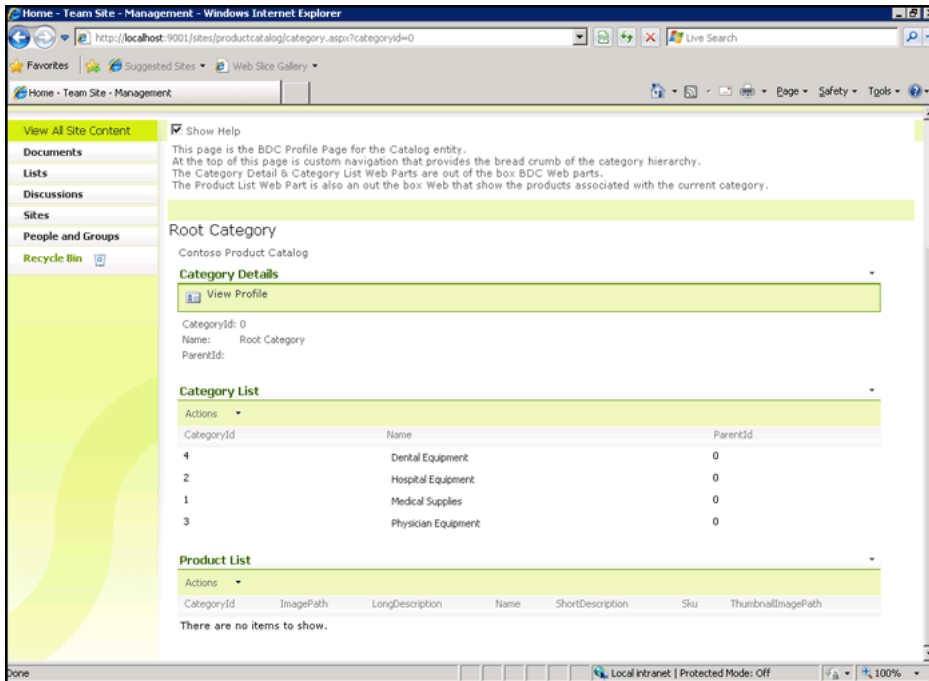
Product Catalog page



You can see the root category with a list of subcategories, and select categories to see their subcategories. The

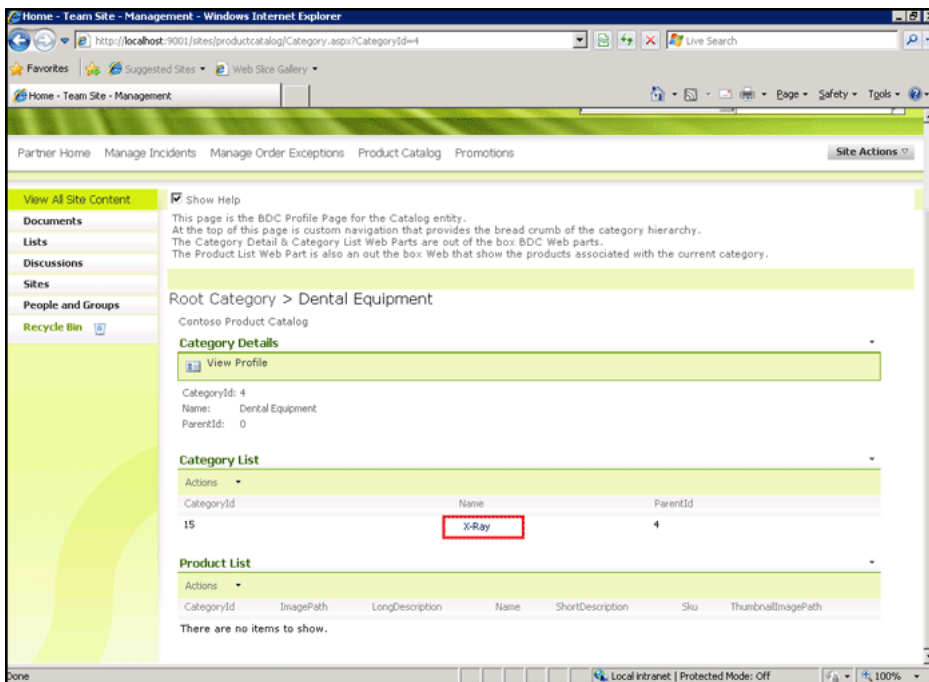
following figure shows the Root Category page.

Root Category page



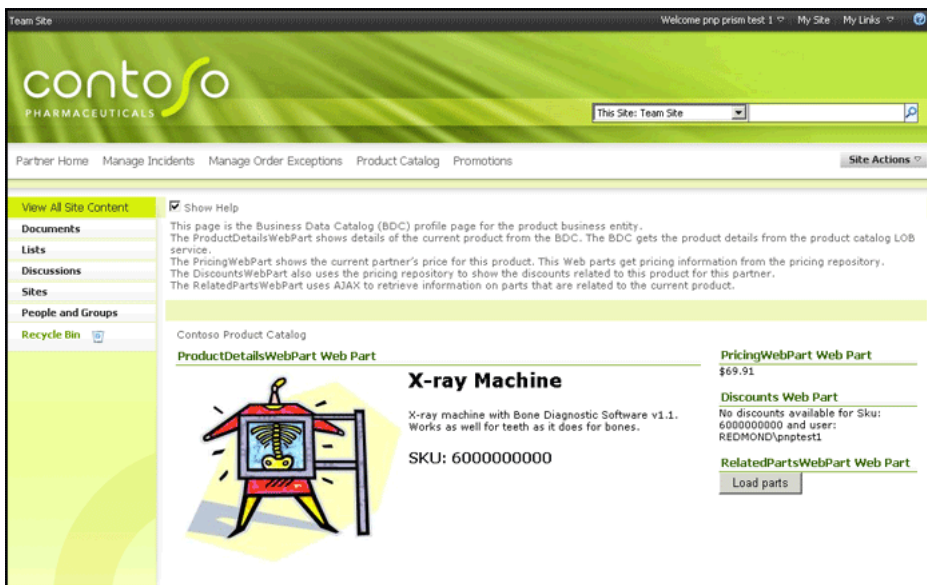
Click the **Dental Equipment** category. Click X-ray. The following figure shows the Category Profile page for Dental Equipment.

Category Profile page



By clicking **X-Ray**, you will see the product catalog page for that product. The Partner Portal application has overridden the default hyperlinks so that the Product Profile page links to the application's publishing page for that product. The following picture shows the Product Details page for the X-ray machine.

Product Details page



Code Walkthrough

The product catalog is implemented with the following components:

- The Product Catalog Web service.
- The BDC application definition file.
- Web parts that are provided by the BDC.
- A custom Web part for the product detail view.

The Product Catalog Web Service

Many enterprises use Web services to access back-end data and operations. The Product Catalog Web service exposes the operations of a simulated product catalog system. This is a minimal implementation that uses a fixed, read-only data set. The **Contoso.LOB.Services.ProductCatalog** Web service is implemented by the Contoso.LOB.Services project. The data is provided by the **ProductCatalogDataSet** class. The data set is initialized at load time by reading XML data stored in the Contoso.LOB.Services\App_Data\ProductCatalogDataSet.xml file.

The Web service is implemented with Windows Communication Foundation (WCF) and uses the **basicHttpBinding**. For more information on WCF see [Windows Communication Foundation](#) on MSDN. The BDC can access basic Web services such as those that are exposed by ASMX, but it does not support many of the configuration options that are available in WCF. By using the **basicHttpBinding**, the exposed services are compatible with the BDC's basic Web service client. For more information on using the **basicHttpBinding**, see [Interoperability with ASP.NET Web Services](#) and [How to: Configure WCF Service to Interoperate with ASP.NET Web Service Clients](#).

WCF uses interfaces to define service operations. The catalog service uses the **IProductCatalog** interface that is located in the Contoso.LOB.Services\Contracts\IProductCatalog.cs file to define its service interface. The **ProductCatalog** class implements this interface and supports the following Web service operations:

- **GetProductSkus**. This operation returns a list of string-valued stock keeping unit IDs (SKUs) for all products in the dataset.
- **GetProductBySku**. This operation returns a product for the SKU specified, or null if no match found.
- **GetProductsByCategory**. This operation returns the list of products contained in the category specified.
- **GetProducts**. This operation returns a list of all products in the product catalog data set.
- **GetCategoryById**. This operation returns the category for the ID specified, or null if no match found.
- **GetCategories**. This operation returns a list of all categories in the Product Catalog data set.
- **GetChildCategoriesByCategory**. This operation returns a list of subcategories of the given category for the ID specified.
- **GetPartsByProductSku**. This operation returns a list of parts for the product for the SKU specified.

Note:

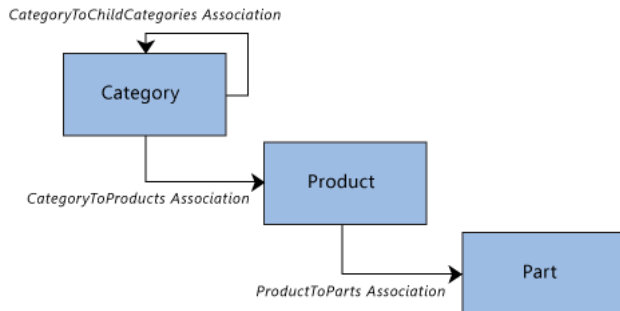
In order to simulate the latency that is typically inherent in database lookups, the Product Details Web service inserts timed delays into each invocation of its Web service methods. The duration of this delay is configured in the Contoso.LOB.Services\Web.config file with the **sleepAmount** application setting. The default delay is 501 milliseconds. The duration of this delay is designed to make the application caching behavior noticeable.

The Product Catalog Metadata Model

Applications define BDC metadata about the services that are provided by the LOB systems. The metadata model represents, for example, LOB systems, LOB system instances, entities, operations, and associations. More

concretely, it describes the information and relationships that are exposed by the Product Catalog service to SharePoint through the BDC. For more information about the BDC metadata model, see [Integrating Line-of-Business Systems](#). The following diagram represents the metadata model that is defined for the catalog application.

BDC metadata model



All entities and associations are exposed by the Product Catalog Web service. Associations describe how the BDC should navigate from one entity to another. A **Category** entity represents the product category. Because categories are hierarchical, categories can be children of other categories. For example, hospital equipment is a child of the root category, and emergency room is a child category of hospital equipment. The **CategoryToChildCategories** association describes this relationship. A **Product** entity represents a product that is offered by Contoso for sale to its partners. Each **Product** entity is assigned to a category. A category can have zero or more products. The **CategoryToProducts** association describes this relationship. A **Part** is an entity that represents a replacement part for a product. A product can have zero or more replacement parts. The **ProductToParts** association describes this relationship.

The BDC Application Definition File

The BDC uses an application definition file for specifying the metadata model. The application definition file `ProductCatalogDefinition.xml` contains XML descriptions for the Product Catalog Web service described in the Product Catalog Web Service section. This file was originally created with the Business Data Catalog Editor and then manually edited. See [Integrating Line-of-Business Systems](#) for more information.

Note:

In all of the XML code shown below for the `ProductDetailsDefinition.xml` file, the **Type** and **TypeName** attributes have been elided (...) whenever they reference the built-in string type. This is to make reading the code easier. Refer to the XML file for the full text.

The LobSystem Declaration

The **LobSystem** object contains all of the sections that are related to the definition of the service. The following XML code shows the **LobSystem** object declaration.

XML

```

<LobSystem
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.microsoft.com/office/2006/03/BusinessDataCatalog
BDCMetadata.xsd"
  Type="WebService"
  Version="1.0.0.0"
  Name="ContosoProductCatalogService"
  xmlns="http://schemas.microsoft.com/office/2006/03/BusinessDataCatalog">
  ...

```

The **LobSystem** object references standard BDC schemas. It also defines the type of the **LobSystem** object as a Web service. For more information, see [LobSystem](#) in the BDC documentation.

The Properties Section

The properties section is contained within the **LobSystem** object declaration and provides properties for the system. The following XML code defines the **Properties** object.

XML

```

...
<Properties>
  <Property Name="WsdlFetchUrl" Type="System.String">
    http://localhost:8585/Contoso.LOB.Services/ProductCatalog.svc
  </Property>

  <Property Name="WebServiceProxyNamespace">
    ContosoProductCatalogProxy
  </Property>
</Properties>
...

```

The **WsdlFetchUrl** property gives the URI of the Product Catalog Web service that the BDC runtime uses to get

the Web service definition. The BDC also uses the URI at run time as the endpoint to access the LOB service.

The **WebServiceProxyNamespace** provides the logical name for the Product Catalog Web service. This name is used in the remainder of the definitions within the **LobSystem** object declaration. For example, the logical name is used when defining operations that the Product Catalog Web service provides.

Note:

The application definition file can specify either a database system or a Web service system as its data source. The Partner Portal application uses a Web service.

The LobSystemInstances Section

The following code shows the **LobSystemInstances** section of the BDC application definition file.

XML

```
<LobSystemInstances>
  <LobSystemInstance Name="ContosoProductCatalogService">
    <Properties>
      <Property Name="LobSystemName" Type="System.String">
        ContosoProductCatalogService
      </Property>
      <Property Name="WebServiceAuthenticationMode"
Type="Microsoft.Office.Server.ApplicationRegistry.SystemSpecific.WebService.HttpAuthenticatio
nMode">
        RevertToSelf
      </Property>
    </Properties>
  </LobSystemInstance>
</LobSystemInstances>
```

This section sets the **Name** attribute to **ContosoProductCatalogService**. The name, which must be unique, is used when:

- Administering the BDC using SharePoint's shared services provider user interface.
- Using XML to configure the Microsoft Office SharePoint Services–provided Web Parts that will display this data.
- Invoking the **GetLobSystemInstanceByName** method of the **Microsoft.Office.Server.ApplicationRegistry.MetadataModel.ApplicationRegistry** static class, which is the programmatic entry point into the BDC data.

This section also sets the **WebServiceAuthenticationMode** property to **RevertToSelf**. This security mode implements the Trusted Subsystem pattern. When a user requests product information, the SharePoint application pool identity is used to authenticate to this instance of the Product Catalog Web service. For more information on the Trusted Subsystem approach see [Business Data Catalog Authentication](#), [Trusted Subsystem](#) in the WCF documentation, and the whitepaper [Trusted Subsystem Design](#).

The Entities Section

The **Entities** section provides the definitions of the business entities and related methods of the catalog service that are exposed through BDC. There are subsections for the category and product business entities.

The Category Business Entity

The **Entity** element declares the **Category** entity. It contains subsections for properties, identifiers and methods.

```
...
<Entity EstimatedInstanceCount="10000" Name="Category">
  <Properties>
    <Property Name="Title" Type="System.String">Name</Property>
  </Properties>

  <Identifiers>
    <Identifier ... Name="CategoryId" />
  </Identifiers>

  <Methods>
    <Method Name="GetCategoryById"> ... </Method>
    <Method Name="GetChildCategoriesByCategory"> ... </Method>
    <Method Name="GetCategories"> ... </Method>
  </Methods>
</Entity>
...
```

The subsections are:

- The **Properties** subsection gives the display title that is used for category lists.
- The **Identifier** element defines the unique key that is used to index into the table of categories.
- The **Methods** section declares each method that is available for this business entity.

Each method declaration includes a detailed declaration of the data types that are used by that method. For example, the following XML shows the type descriptors that are used within the declaration of the

GetCategoryById method.

XML

```
...
<Method Name="GetCategoryById"> ... </Method>
  <Parameters>

    <Parameter Direction="In" Name="categoryId">
      <TypeDescriptor ... IdentifierName="CategoryId" Name="categoryId" />
    </Parameter>

    <Parameter Direction="Return" Name="Return">
      <TypeDescriptor
        TypeName="ContosoProductCatalogProxy.Category, ContosoProductCatalogService"
        Name="Category">
        <TypeDescriptors>
          <TypeDescriptor ... IdentifierName="CategoryId" Name="CategoryId" />
          <TypeDescriptor ... Name="Name" />
          <TypeDescriptor ... Name="ParentId" />
        </TypeDescriptors>
      </TypeDescriptor>
    </Parameter>

  </Parameters>
...
</Method>
...
```

This XML code shows how the data provided to and returned from the **GetCategoryById** method is structured. The **Direction=In** attribute is used for input parameters. The **Direction=Return** attribute is used for the return value.

Note:

If you do not define **Finder** methods, your entity cannot be used in certain BDC Web Parts.

The following code shows the special XML markup that is required for the **Finder** method.

XML

```
...
  <Method Name="GetCategories">
    ...
    <MethodInstance Type="Finder" ReturnParameterName="Return"
      ReturnPropertyDescriptorName="Categories" ReturnPropertyDescriptorLevel="0"
      Name="FindCategoriesInstance" />
    ...
  </Method>
  ...
  <Method Name="GetCategoryById">
    ...

    <MethodInstance Type="SpecificFinder" ReturnParameterName="Return"
      ReturnPropertyDescriptorName="Category" ReturnPropertyDescriptorLevel="0"
      Name="FindSpecificCategoryInstance" />
  </Method>
...

```

The methods **FindCategoriesInstance** and **FindSpecificCategoryInstance** allow the caller to look up data that corresponds to product category records.

The Product Business Entity

The ProductCatalogDefinition.xml file also contains definitions for the **Product** entity. The following XML code shows these definitions.

XML

```
...
<Entity EstimatedInstanceCount="10000" Name="Product">
  <Properties>
    <Property Name="Title" Type="System.String" >Product</Property>
  </Properties>

  <Identifiers>
    <Identifier ... Name="Sku" />
  </Identifiers>

  <Methods>
    <Method Name="GetProductsByCategory"> ... </Method>
    <Method Name="GetProductBySku"> ... </Method>
    <Method Name="GetProducts"> ... </Method>
    <Method Name="GetProductSkus"> ... </Method>
  </Methods>

```

```
</Entity>
...
```

Like the **Category** business entity, the **Product** entity has special methods that are marked as **Finder** and **SpecificFinder**. The **GetProducts** method is the **Finder** method, and the **GetProductBySku** is the **SpecificFinder** method. Unlike the **Category** entity, the **Product** entity defines the **GetProductSkus** method which is used as the **IdEnumerator**. **IdEnumerator** methods are used specifically by SharePoint Search for crawling BDC entities.

The following XML shows a sample type descriptor for the **GetProduct** method.

XML

```

...
<Method Name="GetProducts">
  <Parameters>
    <Parameter Direction="Return" Name="Return">
      <TypeDescriptor
        TypeName="ContosoProductCatalogProxy.Product[],ContosoProductCatalogService"
        IsCollection="true" Name="Products">
          <TypeDescriptors>
            <TypeDescriptor
              TypeName="ContosoProductCatalogProxy.Product,ContosoProductCatalogService" Name="Product">
                <TypeDescriptors>
                  <TypeDescriptor ... Name="CategoryId" />
                  <TypeDescriptor ... Name="ImagePath" />
                  <TypeDescriptor ... Name="LongDescription" />
                  <TypeDescriptor ... Name="Name" />
                  <TypeDescriptor ... Name="ShortDescription" />
                  <TypeDescriptor ... IdentifierName="Sku" Name="Sku" />
                  <TypeDescriptor ... Name="ThumbnailImagePath" />
                </TypeDescriptors>
              </TypeDescriptor>
            </TypeDescriptors>
          </TypeDescriptor>
        </Parameter>
      </Parameters>
    ...
  </Method>
  ...

```

The **Product** business entity includes fields such as **CategoryID** and **ImagePath**. These are the data attributes of the product.

The Associations Section

Relationships between entities are declared with the **Associations** element.

XML

```

<Associations>
  <Association Name="CategoryToChildCategories"
    AssociationMethodEntityName="Category"
    AssociationMethodName="GetChildCategoriesByCategory"
    AssociationMethodReturnParameterName="Return"
    AssociationMethodReturnPropertyDescriptorName="Categories"
    AssociationMethodReturnPropertyDescriptorLevel="0">
    <SourceEntity Name="Category" />
    <DestinationEntity Name="Category" />
  </Association>
  <Association Name="CategoryToProducts" AssociationMethodEntityName="Product"
    AssociationMethodName="GetProductsByCategory"
    AssociationMethodReturnParameterName="Return"
    AssociationMethodReturnPropertyDescriptorName="Products"
    AssociationMethodReturnPropertyDescriptorLevel="0">
    <SourceEntity Name="Category" />
    <DestinationEntity Name="Product"/>
  </Association>
</Associations>
...

```

The **CategoryToChildCategories** association is implemented by invoking the **GetChildCategoriesByCategory** method. The **CategoryToProducts** association is implemented with the **GetProductsByCategory** method.

For more information about association, see [Association](#) on MSDN.

Incident Dashboard

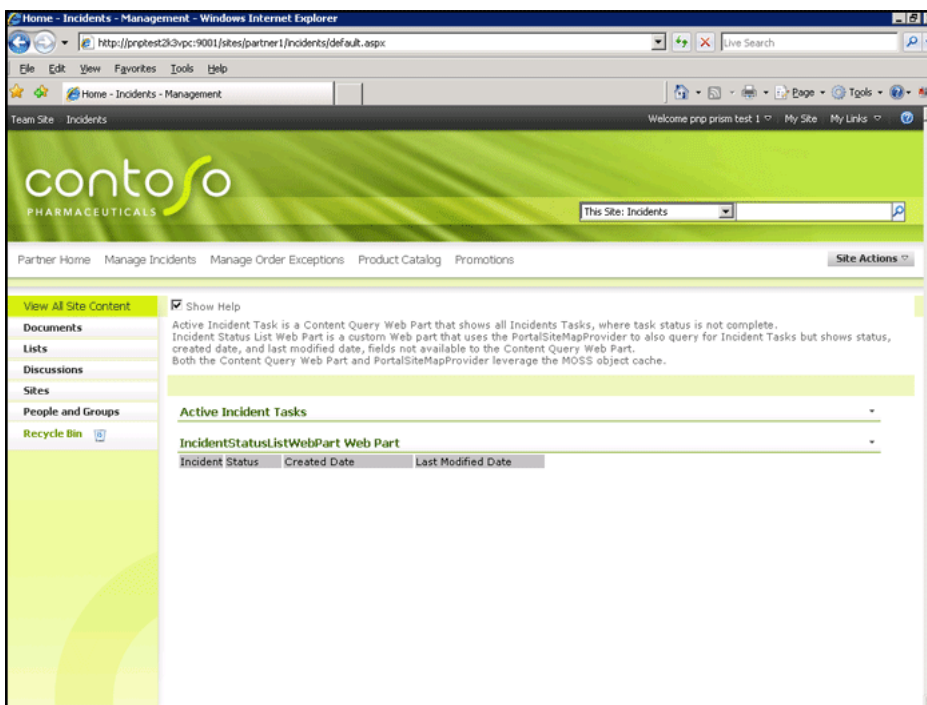
The Contoso Corporation uses an incident management system that tracks any issues that Contoso partners report. An example of an incident is a broken MRI machine that a partner needs to troubleshoot and repair. The incident management system includes a workflow that assigns the incident to higher levels of support as an issue escalates. If an incident reaches level 3, the system invokes a WCF Web service that creates a collaborative SharePoint site. This site allows partners and Contoso engineers to share documents, review status and work together to resolve the problem. The incident Web site is a subsite that exists within the partner's site collection.

The Incident Dashboard is located within a partner's collaboration site collection. It lists the incidents that a partner has reported as well as the active tasks that are associated with each incident. It is similar to a table of contents because it summarizes the information that is contained in other pages. The Incident Dashboard demonstrates a cross-site query with a Content Query Web Part.

To view the Incident Dashboard go to the homepage of the partner's collaboration site collection and click **Manage Incidents** in the global navigation bar.

The following figure illustrates the Incident Dashboard.

Partner Portal Incident Dashboard



The Incident Dashboard contains the following information:

- The active tasks for all incidents
- The active incidents
- The closed incidents

Active Incident Tasks

The Active Incident Tasks pane is a summary of all the active tasks for all incidents. A task is active if its status field is anything other than **Completed**. The summary is the result of a cross-site query.

The Active Incident Tasks pane is implemented with a Content Query Web Part. This is a standard Web Part that is provided by Microsoft Office SharePoint Services. You can use the Content Query Web Part to create custom views of data that is obtained from many lists within a site collection, and to present that data on a single Web page. After you add the Content Query Web Part to a Web page, you can customize the Web Part's querying behavior, custom lists, and content types by setting custom properties.

The Incident Dashboard uses the Content Query Web Part to perform a cross-site query across lists that are found in subsites of the current partner's site collection. Each level 3 incident corresponds to a subsite. The Content Query Web Part presents the results of the query, which are then displayed on the dashboard. The Web Part is configured to include a customized query that is field based. You can also configure the Web Part to search for particular content types and list templates.

The Custom Query Web Part is configured in the file `PartnerPortal\Contoso.PartnerPortal.Collaboration.Incident\IncidentsLandingPage\Module.xml`, found in the `Contoso.PartnerPortal.Collaboration.Incident` project. The following code shows the configuration.

XML

```
<AllUsersWebPart WebPartOrder="1" WebPartZoneID="Left">
  <![CDATA[
```



```

    <webParts>
      <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
        <metaData>
          <type
name="Microsoft.SharePoint.Publishing.WebControls.ContentByQueryWebPart,
Microsoft.SharePoint.Publishing, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" />
          <importErrorMessage>Cannot import this Web Part.</importErrorMessage>
        </metaData>
        <data>
          <properties>
            <! ... !>
          </properties>
        </data>
      </webPart>
    </webParts>
  </>
</AllUsersWebPart>

```

The configuration determines the Web Part's order and zone. It also identifies the Web Part class **ContentByQueryWebPart**. This class is located in the Microsoft.SharePoint.Publishing.WebControls namespace.

The following code shows how to create the custom query. This code is located in PartnerPortal\Contoso.PartnerPortal.Collaboration.Incident\IncidentsLandingPage\Module.xml.

XML

```

<property name="ContentTypeBeginsWithId"
type="string">0x0108009BBBA7168ED64455B16ECDBD00BA7997</property>

```

The **ContentTypeBeginsWithId** property sets the content type ID for the custom query. The string 0x0108009BBBA7168ED64455B16ECDBD00BA7997 is the content type identifier for the Incident Task content type. This content type is defined in the Content Types/IncidentTask/IncidentTask.xml file of the Contoso.PartnerPortal.Collaboration.Incident project.

By default, the Content Query Web Part searches the current site and all of its subsites. In the Partner Portal application, the partner collaboration site that contains the incident dashboard also contains a subsite for each incident that is associated with that partner. As a consequence, the Web Part's default scope searches all incident tasks for the current partner.

Note:

It is possible to customize the scope of the query. For more information, see [How to: Customize the Content Query Web Part by using Custom Properties](#) on MSDN.

The query is further restricted to include only incident tasks that have not been completed. This field-based query is expressed by overriding properties with names that begin with **Filter**. The query for the Active Incident Tasks is shown in the following code.

XML

```

<properties>
  <!-- ... -->
  <property name="FilterField1"
    type="string">{c15b34c3-ce7d-490a-b133-3f4de8801b76}</property>
  <!-- ... -->
  <property name="FilterType1" type="string">Choice</property>
  <!-- ... -->
  <property name="FilterOperator1"
type="Microsoft.SharePoint.Publishing.WebControls.ContentByQueryWebPart+FilterFieldQueryOpera
tor, Microsoft.SharePoint.Publishing, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c">Neq</property>
  <!-- ... -->
  <property name="FilterValue1" type="string">Completed</property>
  <!-- ... -->
</properties>

```

Together, the four properties, **FilterField1**, **FilterType1**, **FilterOperator1** and **FilterValue1** express the constraint that the status field must not equal **Completed**.

The **FilterField1** property identifies the field that is used for comparison. In this case, the property is set to the field ID of the **Status** field. This ID is defined in the Lists\Tasks\schema.xml file of the Contoso.PartnerPortal.Collaboration.OrderException project.

The **FilterType1** property is set to the value **Choice** to indicate that the comparison is performed on enumerated values. The **Status** field is declared with the **Choice** type.

The **FilterOperator1** property specifies that the basis of selection is the **Neq** operator, which means not equal.

The **FilterValue1** property is the value of the **Status** field that is used for comparison. In this case, the value is **Completed**.

More Information

For more information on the Content Query Web Part, see [How to: Customize the Content Query Web Part by using Custom Properties](#) on MSDN.

Logging and Tracing in the Partner Portal

This topic shows how the Partner Portal application uses the SharePoint logger that is provided by the SharePoint Guidance Library.

Locating the SharePoint Logger Service

The Partner Portal's **ProductDetailsPresenter** class demonstrates the following:

- It demonstrates how to use the SharePoint logger to record information to the unified logging service (ULS) and the Windows event log.
- It demonstrates how the SharePoint service locator decouples the logging consumer from the implementation.

The following code shows the **ProductDetailsPresenter** class's constructor. The product details presenter implementation is located in `Contoso.PartnerPortal.ProductCatalog\WebParts\ProductDetails\ProductDetailsPresenter.cs`.

```
C#
public class ProductDetailsPresenter
{
    // ...

    private IProductDetailsView view;
    private ILogger logger;

    public ProductDetailsPresenter(IProductDetailsView view)
    {
        this.view = view;
        this.logger = SharePointServiceLocator.Current.GetInstance<ILogger>();
    }
    // ...
}
```

The constructor uses the **SharePointServiceLocator** class to get a logger instance and save it in an instance field named **logger**. The **ProductDetailsPresenter** class is now decoupled from a specific logger implementation. The choice of logger implementation is configurable. The default logger can be replaced without modifying the **ProductDetailsPresenter** class.

For more information about the SharePoint service locator, see [The SharePoint Service Locator](#).

Methods of the **ProductDetailsPresenter** class write diagnostic information to the logger that was initialized in the constructor. For example, the following code shows the **LoadProduct** method.

```
C#
public void LoadProduct(string sku)
{
    try
    {
        this.logger.TraceToDeveloper("Start LoadProduct.");

        // ... perform application logic ...

        this.logger.TraceToDeveloper("End LoadProduct.");
    }

    catch(Exception ex)
    {
        // If something goes wrong, make sure the error gets logged
        new ViewExceptionHandler().HandleViewException(ex, this.ErrorVisualizer,
            "Due to a technical problem, the product details cannot be displayed.");
    }
}
```

The logger is invoked three times. There are **TraceToDeveloper** calls at the beginning and end of the **LoadProduct** method that create entries in the ULS trace log. There is also a call to the **HandleViewException** method of the **ViewExceptionHandler** class. The **HandleViewException** method also calls the logger's **LogToOperations** method. This ensures that if the **LoadProduct** method throws an exception, the details are preserved in both the Windows event log and the ULS trace log.

Product Catalog Search Configuration

This topic describes how to configure the SharePoint Search account and how to search the Contoso product catalog. You must edit the settings for the Farm Search Service account and for the Default Content Access account. Configuring search can be quite complex and this topic only covers the essentials. For more information, see [Deploy in a simple server farm](#) (Office SharePoint Server), [Configure the Office SharePoint Server Search service](#) (Office SharePoint Server), and [Plan search](#) (Office SharePoint Server) on TechNet.

Configuring the Farm Search Service Account

The search service runs under the farm service account. In a production farm, you must set this account as a domain account for security. To review the details of setting up accounts, see [Plan for administrative and service accounts](#) on TechNet. Because different deployment scenarios can affect the account requirements, review this material before you implement a production deployment. The default settings that are installed by the Partner Portal application will not work because the Local Service account does not have access to the Contoso Product Catalog Business Data Catalog (BDC). You will need to create an account for the search service. This account must belong to the WSS_ADMIN_WPG and Administrators group (for more information, see [Office SharePoint Server Search and Windows SharePoint Server Search may not work as expected after you update Office SharePoint Server 2007 Search](#)). Additionally, this account must be added to the local group name ContosoTrustedAccounts. The catalog service will only permit members of the ContosoTrustedAccounts group to access the service. For information about how to change the search service account, see "To change credentials for the Office SharePoint Server Search service by using the user interface" in [Change passwords for the Search service accounts \(Search Server 2008\)](#) on TechNet.

Configuring the Default Content Access Account

You must specify a default account that has read permissions in order to crawl content. To avoid crawling unpublished versions of documents, it is recommended that this account not have administrative privileges on the target server. For more information, see [Plan for administrative and service accounts](#) and [Configure default content access account](#) on TechNet. For simplicity, you can use ContosoAppPoolUser (Password: P2ssw0rd\$) account credentials with the Partner Portal application, which does not have administrative privileges. Limit the account's access to the content to read-only.

Configuring BDC Search

The default account crawls business data that is exposed through the BDC. For more information, see the section "Create content source for business data" in [Configure the Office SharePoint Server Search service](#) on TechNet. In order for search to crawl a business data source that is defined by the BDC, you must implement a specific finder and an **IDEnumerator** interface. For more information, see [MethodInstance](#) in the BDC documentation. The following procedure configures the BDC search for the Partner Portal application.

To configure the BDC search for the Partner Portal

1. On the taskbar, click **Start**, point to **Administrative Tools**, and then click **SharePoint 3.0 Central Administration**.
2. In the Quick Launch, click **ContosoSSP**. The Home page opens.
3. In the **Search** area, click **Search settings**. The Configure Search Settings page appears.
4. Click **Content sources and crawl schedules**. The Manage Content Sources page appears.
5. On the toolbar, click **New Content Source**. The Add Content Source page appears.
6. In the **Name** field, enter **ContosoBDCSource**.
7. In the **Content Source Type** area, select **Business Data**.
8. In the **Applications** area, select **Crawl selected applications**, and then select the **ContosoProductCatalogService** check box.
9. In the **Crawl Schedules** area, in the **Full Crawl** drop-down box, click **None**. In the **Incremental Crawl** drop-down box, click **None**.
10. In the **Start Full Crawl** area, clear the **Start full crawl of this content source** check box if it is selected. You will manually start the crawl.
11. Click **OK**.

This procedure does not cover every possible way to configure the BDC. For more information, see [Walkthrough: Configuring Search for the AdventureWorks Business Data Application](#) on MSDN,

Configuring a Search Scope

Search scopes narrow the search results of a search query. For more information about how to create scopes and scope rules, see [Configure the Office SharePoint Server Search service \(Office SharePoint Server\)](#) on TechNet. The following procedure is for the Partner Portal application. It sets a scope that limits the search to the Contoso product catalog.

To set the search scope

1. On the taskbar, click **Start**, point to **Administrative Tools**, and then click **SharePoint 3.0 Central Administration**.
2. In the Quick Launch, click **ContosoSSP**. The Home page opens.
3. In the **Search** area, click **Search settings**. The Configure Search Settings page appears.
4. In the **Scopes** area, click **View Scopes**. The View Scopes page appears.

5. On the toolbar, click **New Scope**. The Create Scope page appears.
6. In the **Title** field, enter **ContosoBDCScope**.
7. Click **OK**.
8. Under **Update Status**, click **Add Rules** where it says "Empty - Add Rules" on the newly added scope.
9. Select the **Content Source** radio button.
10. In the **Content Source** drop-down box, click **ContosoBDCSource**.
11. Click **OK**.

The following procedure updates the scope.

To force the scope to update

1. On the Central Administration page, click **ContosoSSP** in the Quick Launch.
2. In the **Search** section, click **Search settings**.
3. In the **Scopes** section, click **Start update now**.
4. Click **View scopes**.

After the update completes, you will see the status of the ContosoBDCScope change to **Ready**.

Crawling the BDC Source

To return results from a search query, you must crawl the BDC source. For information about setting up crawl rules, see [Configure the Office SharePoint Server Search service](#). You can also initiate a manual crawl of the catalog. The following procedure shows how to do this.

To initiate a manual crawl

1. On the Central Administration page, click **ContosoSSP** in the Quick Launch.
2. In the **Search** section, click **Search settings**.
3. Click **Content sources and crawl schedules**.
4. Right-click **ContosoBDCSource**, and then click **Start Full Crawl**.
5. Refresh the page, until the status changes from **Crawling Full** to **Idle**.
6. Right-click **ContosoBDCSource**, and then click **View Crawl Log**. You should see a list of product pages.

Create a Search Site

To search the BDC, you must create a search site. The following procedure shows how to do this.

To create a search site

1. Navigate to the Product Catalog site at <http://localhost:9001/sites/productcatalog/category.aspx?categoryid=0>.
2. Click the **Site Action** tab, and then click **Create**.
3. Under **Web Pages**, click **Sites and Workspaces**.
4. In the **Title** text box, type **Contoso Search Center**.
5. In the **URL name** text box, type **Search**.
6. Under **Select a template**, click the **Enterprise** tab, and then click **Search Center**.
7. Click **Create**.

After the Contoso Search Center is created, you can type **X-Ray** in the search box, and start the search. You should see one result that is named **Product.aspx**. Click the result link. It should open the page located at <http://localhost:9001/sites/ProductCatalog/Product.aspx?sku=6000000000>.

More Information

For more information about searching the BDC, see [Searching Business Data](#) on MSDN. Also see [Configure the Office SharePoint Server Search service](#) and [SharePoint 2007: BDC - Enabling Search on business data](#) on the Winsmarts.com site.

Subsite Creation Configuration

The Partner Portal application uses the workflow that is defined in the Microsoft.Practices.SPG.SubSiteCreation.Features project to programmatically create collaboration sites for partner incidents. This workflow assumes that the Web-scoped feature named Microsoft patterns & practices Sub Site Creation Infrastructure (Web) is activated. Activating this feature creates a list that is named Business Event Type Configuration. This list maps a business event to its corresponding site template or site definition. The Uniform Resource Identifier (URI) of the list is `http://<siteName>/Lists/BusinessEventTypeConfiguration`.

When the **SubSiteCreation** workflow starts, the **ResolveSiteTemplateActivity** workflow activity runs. This workflow activity looks up the site template name that is associated with the current workflow instance's business event name.

The Microsoft patterns & practices Sub Site Creation Infrastructure (Web) feature is activated on the **SpgSubsite** site in the **PartnerCentral** site collection. The Partner Portal application's installation script populates the Business Event Type Configuration list with two list items. The following table lists these items.

Title	Business Event	Site Template	Business Event ID Key	Top-Level Site Relative URL
Incident	Incident	SPGSubsiteTemplate	IncidentId	Incidents
Order Exception	OrderException	ORDEREXCEPTION#0	OrderExceptionId	OrderExceptions

Note:

You can use the SharePoint Guidance Library's subsite creation workflow for your own SharePoint applications. To do this, associate a new business event name with a site template or site definition by adding a new item to the Business Event Type Configuration list.

New Partner Provisioning

The Partner Portal application includes a configuration script that is named ContosoSetup.bat. This script provisions the Partner1 and Partner2 sites. This topic includes two procedures. The first procedure shows you how to add additional partner sites. The second procedure shows you how to add new members and a new role to the forms-based authentication database.

Note:

Wherever you see PartnerN in the procedures, substitute Partner3, Partner4, and so on.

To provision a new partner site

1. Run the following script to create a PartnerN site collection, incidents site, and order exceptions site. The script also activates the features that are required by these sites.

MS-DOS

```
REM Create PartnerN site, incidents web and orderexceptions web
REM Replace PartnerN with respective Partner ex:PartnerN
REM update ownerlogin with respective login

set ownerlogin=domain\username
set stsadm=%CommonProgramFiles%\Microsoft Shared\web server extensions\12\BIN\stsadm.exe

if exist "%stsadm%" goto STSEnd
stsadm=C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN\stsadm.exe

if exist "%stsadm%" goto STSEnd
stsadm=C:\Program Files (x86)\Common Files\Microsoft Shared\web server
extensions\12\BIN\stsadm.exe
:STSEnd

"%stsadm%" -o Createsite -url http://localhost:9001/sites/PartnerN -owneremail
myname@mydomain -ownerlogin %ownerlogin% -sitetemplate STS#0 -description "PartnerN"

"%stsadm%" -o createweb -url http://localhost:9001/sites/PartnerN/incidents -sitetemplate
sts#1 -title Incidents
"%stsadm%" -o createweb -url http://localhost:9001/sites/PartnerN/orderexceptions
-sitetemplate sts#1 -title "Order Exceptions"
REM Activating feature PublishingSite
"%stsadm%" -o activatefeature -id F6924D36-2FA8-4f0b-B16D-06B7250180FA -url
http://localhost:9001/sites/PartnerN
REM Activating feature PublishingWeb
"%stsadm%" -o activatefeature -id 94C94CA6-B32F-4da9-A9E3-1F3D343D7ECB -url
http://localhost:9001/sites/PartnerN
REM Activating feature ContosoPartnerPortalPromotionsSite
"%stsadm%" -o activatefeature -id 6fd12cb8-f910-4cf4-b7d0-6b74769483ae -url
http://localhost:9001/sites/PartnerN
REM Activating feature PartnerCollaborationSite
"%stsadm%" -o activatefeature -id f15ec5c3-d97f-4838-b874-0224dbbee676 -url
http://localhost:9001/sites/PartnerN
REM Activating feature PartnerCollaborationWeb
"%stsadm%" -o activatefeature -id 0d4249b5-6c67-466a-8c5a-a49cb7aff73e -url
http://localhost:9001/sites/PartnerN
REM Activating feature IncidentSubSiteResources
"%stsadm%" -o activatefeature -id 1c28f48f-6544-45b3-8db0-e87977c09e46 -url
http://localhost:9001/sites/PartnerN
REM Activating feature IncidentDashboard
"%stsadm%" -o activatefeature -id 86dfa338-3643-49c3-a935-26b043547326 -url
http://localhost:9001/sites/PartnerN/incidents
REM Activating feature ContosoPSSOrderExceptionSite
"%stsadm%" -o activatefeature -id c5decae2-d3e2-4048-a25a-42c5fe6eac29 -url
http://localhost:9001/sites/PartnerN
REM Activating feature ContosoTheme
"%stsadm%" -o activatefeature -id 7a576729-6fb8-4597-bbad-59c2aa7d7d0e -url
http://localhost:9001/sites/PartnerN
"%stsadm%" -o activatefeature -id 7a576729-6fb8-4597-bbad-59c2aa7d7d0e -url
http://localhost:9001/sites/PartnerN/incidents
"%stsadm%" -o activatefeature -id 7a576729-6fb8-4597-bbad-59c2aa7d7d0e -url
http://localhost:9001/sites/PartnerN/orderexceptions
REM Activating feature ContosoContextualHelp
"%stsadm%" -o activatefeature -id 92a99385-aefc-4718-98c2-1ac541c55a6e -url
http://localhost:9001/sites/PartnerN
"%stsadm%" -o activatefeature -id 92a99385-aefc-4718-98c2-1ac541c55a6e -url
http://localhost:9001/sites/PartnerN/incidents
"%stsadm%" -o activatefeature -id 92a99385-aefc-4718-98c2-1ac541c55a6e -url
http://localhost:9001/sites/PartnerN/orderexceptions
REM Activating feature ContosoPSSGlobalNav
"%stsadm%" -o activatefeature -id be8c9c0c-3201-45e7-a433-806d9dfc35eb -url
http://localhost:9001/sites/PartnerN
"%stsadm%" -o activatefeature -id be8c9c0c-3201-45e7-a433-806d9dfc35eb -url
```

```
http://localhost:9001/sites/PartnerN/incidents
"%stsadm%" -o activatefeature -id be8c9c0c-3201-45e7-a433-806d9dfc35eb -url
http://localhost:9001/sites/PartnerN/orderexceptions
```

2. Add entries to the partner directory. To do this, do the following:
 - a. Browse to <http://localhost:9001/sites/PartnerCentral>.
 - b. In the Quick Launch, click **Sites**.
 - c. Click **PartnerDirectory**. The Site Directory page appears.
 - d. Click the **Partners** tab.
 - e. Click **New**.
 - f. In the **Sites: New Item** form, enter the following information:
 - a. In the **Title** field, enter **PartnerN**.
 - b. In the **Partner** field, enter **ContosoPartnerN**.
 - c. In the URL field, enter <http://localhost:9001/sites/partnerN>.
3. Click **OK**.
4. Add the Windows Users and Group. To do this, do the following:
 - a. On the taskbar, click **Start**, point to **Administrative Tools**, and then click **Computer Management**. Expand the **Local Users and Groups** node.
 - b. Right-click **Users**, and then click **New User**. Enter the following information:
 - a. In the User name field, enter ContosoPartnerNUser6.
 - b. Fill in the other text fields as you see fit.
 - c. Clear the User must change password in next login check box.
 - d. Click **Create**.
 - e. Repeat step 4 for users 7, 8, and 9.
 - c. Right-click **Groups**, and then click **New Group**. Enter the following information:
 - a. In the Group name field, enter ContosoWinPartnerN.
 - b. Click **ADD**. In the **Members** field, click each user that you created in the previous step.
 - c. Click **Create**.
 - d. Click **Close**.
5. Run the following script to add the ContosoWinPartnerN permissions to the site collections, incidents site, and order exception site.

MS-DOS

```
REM Create PartnerN site, incidents web and orderexceptions web
REM Replace PartnerN with respective Partner ex:PartnerN

set domainName=%ComputerName%

set stsadm=%CommonProgramFiles%\Microsoft Shared\web server extensions\12\BIN\stsadm.exe

if exist "%stsadm%" goto STSEnd
stsadm=C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN\stsadm.exe

if exist "%stsadm%" goto STSEnd
stsadm=C:\Program Files (x86)\Common Files\Microsoft Shared\web server
extensions\12\BIN\stsadm.exe
:STSEnd

REM Add ContosoWinPartnerN to partnerN site http://localhost:9001/sites/partnerN
"%stsadm%" -o adduser -url http://localhost:9001/sites/partnerN -userlogin
%domainName%\ContosoWinPartnerN -useremail partnerN@SPG.com -role Design -username
ContosoWinPartnerN
REM Add ContosoFbaPartners to Default sites http://localhost:9001
"%stsadm%" -o adduser -url http://localhost:9001 -userlogin %domainName%\ContosoWinPartnerN
-useremail partnerN@SPG.com -role Design -username ContosoWinPartnerN
REM Add ContosoFbaPartners to Portal sites http://localhost:9001/sites/promotions
"%stsadm%" -o adduser -url http://localhost:9001/sites/promotions -userlogin
%domainName%\ContosoWinPartnerN -useremail partnerN@SPG.com -role Design -username
ContosoWinPartnerN
REM Add ContosoFbaPartners to Portal sites http://localhost:9001/sites/productcatalog
"%stsadm%" -o adduser -url http://localhost:9001/sites/productcatalog -userlogin
%domainName%\ContosoWinPartnerN -useremail partnerN@SPG.com -role Design -username
ContosoWinPartnerN
REM Add ContosoWinPartnerN into http://localhost:9001/sites/PartnerCentral
"%stsadm%" -o adduser -url http://localhost:9001/sites/PartnerCentral -userlogin
%domainName%\ContosoWinPartnerN -useremail partnerN@SPG.com -role Read -username
ContosoWinPartnerN
```

6. Configure the Business Data Catalog access for ContosoWinPartnerN. To do this, do the following:
 - a. Start Sharepoint3.0 Central Administration.
 - b. In the Quick Launch, click **ContosoSSP**.
 - c. Under **Business Data Catalog**, click **View applications**.
 - d. Click **ContosoProductCatalogService**.
 - e. Click **Manage Permissions**.

- f. Click **Add Users/Groups**.
 - g. In the **Users/Groups** field, enter **ContosoWinPartnerN**. In the **Choose Permissions** area, select the check boxes for all four permissions (**Edit**, **Execute**, **Select in Clients**, and **Set Permissions**).
 - h. Click **Save**.
 - i. Select the **ContosoWinPartnerN** check box. Click **Copy all permissions to descendants**. In the **Message from webpage** dialog box, click **OK**.
7. Add entries in SharePoint for the new Windows users. To do this, do the following:
 - a. Start Sharepoint3.0 Central Administration.
 - b. In the Quick Launch, click **ContosoSSP**.
 - c. Click **User profiles and properties**.
 - d. Click **Add user profile**.
 - e. Enter the following information in the Add User Profile form:
 - a. In the Account name field, enter either **mydomain\ContosoPartnerNUser6** or **myPCname \ContosoPartnerNUser6**.
 - b. In the **Name** field, enter either **mydomain\ContosoPartnerNUser6** or **myPCname \ContosoPartnerNUser6**.
 - c. In the **Partner Id** field, enter **ContosoPartnerN**.
 - f. Click **SaveandClose**. The User Profiles and Properties page appears.
 - g. Click **View user profiles**.
 - h. In the ContosoPartnerNUser6 drop-down box, click **Edit**. Make sure that the **Partner Id** field contains the partner ID. If it does not, enter it. Click **Save and Close**.
 - i. Repeat these steps for ContosoPartnerNUser7, 8, and 9.
 8. Modify the Contoso.LOB.Services data files to add sample data for PartnerN. Add new incident data in the \Source\PartnerPortal\Contoso.LOB.Services\App_Data\IncidentDataSet.xml file. Add new pricing data in the \Source\PartnerPortal\Contoso.LOB.Services\App_Data\PricingDataSet.xml file.
 9. Recompile the Contos.LOB.Services project.
 10. Navigate to <http://localhost:9001/sites/PartnerN> to see the new partner site.

The following procedure shows how to add the new partners to the forms-based authentication (FBA) membership database. This procedure is optional. Use it only if you want new partners to be able to use FBA. To do this, you must edit the membership database to create the new members and role. This requires that you use the [Membership Seeder](#) program, which is on CodePlex. This procedure does not explain how to use Membership Seeder. For more information, see [Forms Authentication in SharePoint Products and Technologies \(Part 1\): Introduction](#) on MSDN.

To use Membership Seeder to add the FBA members and role

1. Add new data to the ContosoFBAdb database. (The ContosoFBAdb database is installed by the Partner Portal application's installation script. It is in the local\OFFICESERVERS SQL Server instance.) To do this, do the following:
 - a. Open Membership Seeder, and then click **Configure**. In the dialog box, enter **local\OFFICESERVERS**. This is where the membership database is hosted. Save your changes, and then close Membership Seeder.
 - b. Reopen Membership Seeder.
 - c. In the **Role** field, enter **ContosoFbaPartnerN**.
 - d. Add ContosoPartnerNUser6, 7,8, and 9 to the **dbo.aspnet_Users** table.
 - e. Add the entry that maps the users to the role in the **dbo.aspnet_UsersInRoles** table.
2. Run the following script to add ContosoFbaPartnerN permissions to the site collections and sites.

MS-DOS

```
REM Create PartnerN site, incidents web and orderexceptions web
REM Replace PartnerN with respective Partner Number ex:Partner99

set stsadm=%CommonProgramFiles%\Microsoft Shared\web server extensions\12\BIN\stsadm.exe

if exist "%stsadm%" goto STSEnd
stsadm=C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN\stsadm.exe

if exist "%stsadm%" goto STSEnd
stsadm=C:\Program Files (x86)\Common Files\Microsoft Shared\web server
extensions\12\BIN\stsadm.exe
:STSEnd

REM Add ContosoFbapartnerN to partnerN site http://localhost:9001/sites/partnerN
"%stsadm%" -o adduser -url http://localhost:9001/sites/partnerN -userlogin
PartnerGroups:ContosoFbapartnerN -useremail partnerN@SPG.com -role Design -username
ContosoFbapartnerN
REM Add ContosoFbaPartners to Default sites http://localhost:9001
"%stsadm%" -o adduser -url http://localhost:9001 -userlogin PartnerGroups:ContosoFbapartnerN
-useremail partnerN@SPG.com -role Design -username ContosoFbapartnerN
REM Add ContosoFbaPartners to Portal sites http://localhost:9001/sites/promotions
"%stsadm%" -o adduser -url http://localhost:9001/sites/promotions -userlogin
PartnerGroups:ContosoFbapartnerN -useremail partnerN@SPG.com -role Design -username
ContosoFbapartnerN
REM Add ContosoFbaPartners to Portal sites http://localhost:9001/sites/productcatalog
```



```
%stsadm% -o adduser -url http://localhost:9001/sites/productcatalog -userlogin
PartnerGroups:ContosoFbapartnerN -useremail partnerN@SPG.com -role Design -username
ContosoFbapartnerN
REM Add ContosoFbapartnerN into http://localhost:9001/sites/PartnerCentral
"%stsadm%" -o adduser -url http://localhost:9001/sites/PartnerCentral -userlogin
PartnerGroups:ContosoFbapartnerN -useremail partnerN@SPG.com -role Read -username
ContosoFbapartnerN
```

3. Configure the Business Data Catalog access for ContosoFbaPartnerN. To do this, do the following:
 - a. Start Sharepoint 3.0 Central Administration.
 - b. Click **ContosoSSP**.
 - c. Under **Business Data Catalog**, click **View applications**.
 - d. Click **ContosoProductCatalogService**.
 - e. Click **Manage Permissions**.
 - f. Click **Add Users/Groups**.
 - g. In the **Users/Groups** field, type **ContosoFbapartnerN**. In the **Choose Permissions** area, select the check boxes for all four permissions (**Edit**, **Execute**, **Select in Clients**, and **Set Permissions**).
 - h. Click **Save**.
 - i. Select the **ContosofbaPartnerN** check box. Click **Copy all permissions to descendants**. In the **Message from webpage** dialog box, click **OK**.
4. Add entries to User Profiles for new FBA partner users:
 - a. Start Sharepoint3.0 Central Administration.
 - b. Click **ContosoSSP**.
 - c. Click **User profiles and properties**.
 - d. Click **Add user profile**.
 - e. Enter the following information in the Add User Profile form:
 - a. In the **Account name** field, enter **Partners:ContosoPartnerNUser6**.
 - b. In the **Name** field, enter **Partners:ContosoPartnerNUser6**.
 - f. Click **SaveandClose**. The User Profiles and Properties page appears.
 - g. Click **View user profiles**.
 - h. In the **ContosoPartnerNUser6** drop-down box, click **Edit**. In the **Partner Id** field, enter **ContosoPartnerN**.
 - i. Click **Save and Close**.
 - j. Repeat these steps for ContosoPartnerNUser7, 8, and 9.

Naming Conventions for the Procedures

The following naming conventions are used in the two procedures:

- **ContosoWinPartnerN**. This is the local Windows group or the Active Directory domain group.
- **ContosoFbaPartnerN**. This is the role in the ContosoFBadb database.
- **Partners**. This is the membership provider that is defined in the **PeoplePickerWildcards** element in the Web.config file that is used by SharePoint Central Administration.
- **PartnerN**. This is used in three places. They are the following.
 - It is the site collection name and it is used in the site collection URL. For example, http://localhost:9001/sites/PartnerN.
 - It is used in the user profile in the **Partner Id** field.
 - It is used in the PartnerCentral\PartnerDirectory in the **Title** field.
- **ContosoPartnerN**. This is used in four places. They are the following:
 - It is used in the **Partner** field in the page: http://localhost:8585/contoso.lob.web/CreateOrderExceptionSite.aspx.
 - It is used in the PartnerCentral\PartnerDirectory for the **Partner** field.
 - It is used in Source\PartnerPortal\Contoso.LOB.Services\App_Data\IncidentDataSet.xml file.
 - It is used in Source\PartnerPortal\Contoso.LOB.Services\App_Data\PricingDataSet.xml file.

Training Management Reference Implementation

The Contoso Training Management application demonstrates how the Human Resources group of Contoso Pharmaceuticals uses SharePoint to manage its training course offerings. (In the rest of this documentation, the Contoso Training Management Reference Implementation is referred to as the Training Management application.) Employees can perform activities such as enrolling in training courses and seeing the status of their registrations. For example, training managers can see any pending registrations that they must either approve or reject and then perform those tasks. To install the Training Management application, see [Installing the Training Management Application](#).

To learn about the design of the application, see [Design of the Training Management Application](#). This topic explains the operating environment, the structure of the application, and how the application uses a service locator.

To learn about how each of the use cases is implemented, see [Training Management Application Walkthrough](#). This topic explains the code in detail.

After you install the Training Management application, you can upgrade it to add new functionality. You can also brand the site with the Contoso theme. To learn more about upgrading the application and applying the theme, see [Training Management Application Upgrade](#).

Design of the Training Management Application

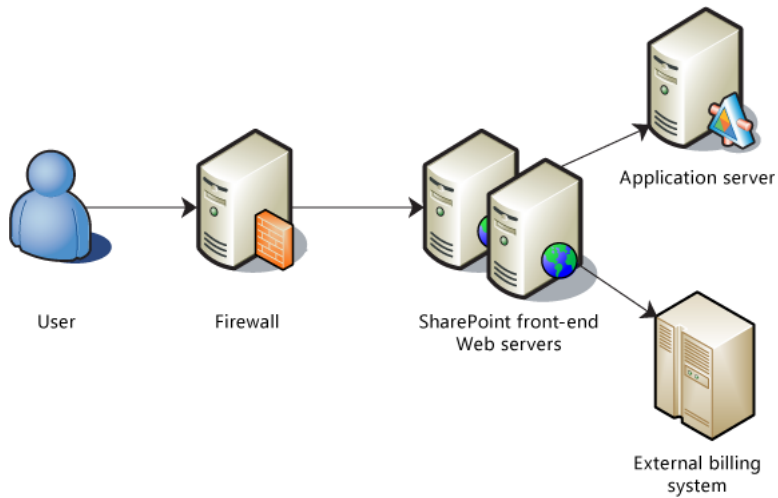
This section describes the design of the Training Management application. It includes the following topics:

- [The Operating Environment](#). This topic discusses the environment that is used by the Training Management application.
- [The Application Structure](#). This topic discusses the architecture of the Training Management application.
- [Service Locator Assembly](#). This topic discusses the service locator. This class instantiates services that are used in the application.

The Operating Environment

The following figure illustrates an example of a SharePoint operating environment. It includes a firewall, a Web front-end cluster, an external billing system, and the application server.

Example of SharePoint operating environment



A possible deployment scenario is to locate the default Web page on the Web front-end server. The application server runs the central administration. The application uses an external billing system to charge the appropriate cost centers when employees enroll in courses. The business logic also runs on the billing system. The SharePoint front-end servers present the data from the billing system.

The Application Structure

The Training Management application is a SharePoint application. This topic discusses its structure, the interactions between components, and the application's classes.

Assemblies

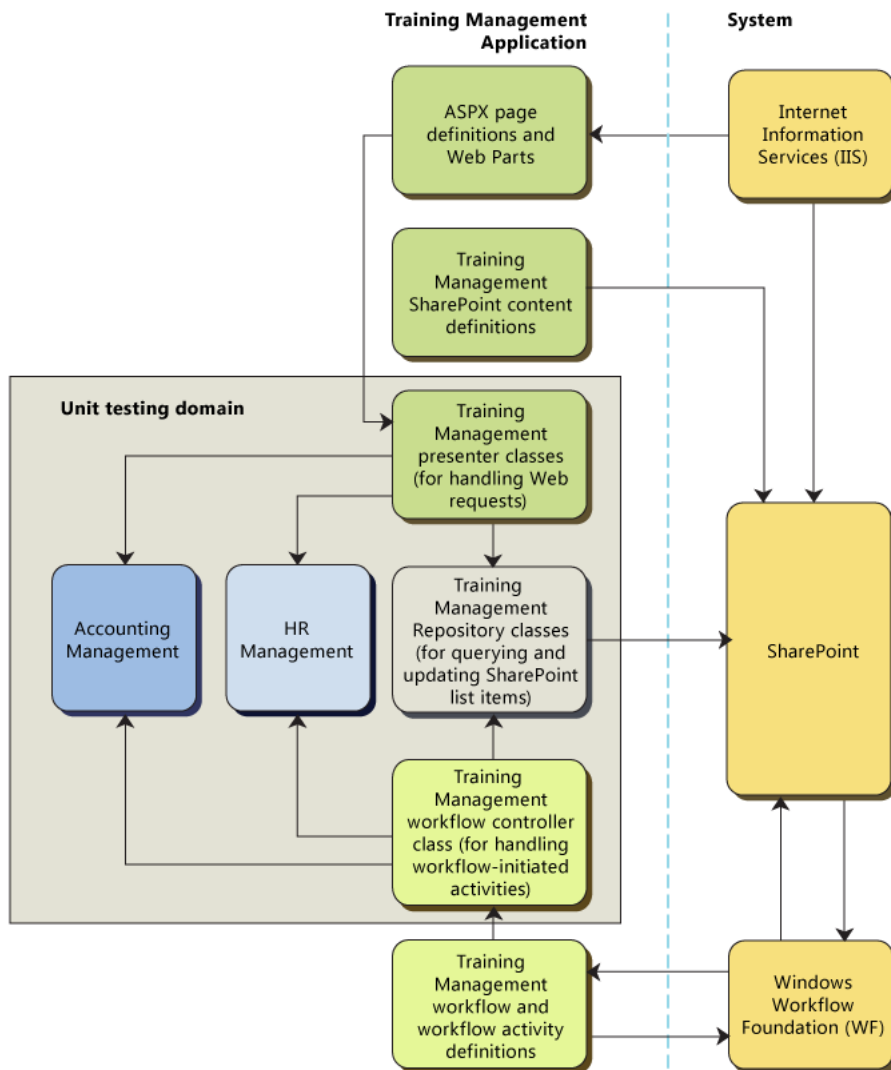
The Training Management application consists of nine .NET Framework assemblies that interact with SharePoint and with Windows Server components. Each assembly corresponds to a Visual Studio project that is found in the Contoso.TrainingManagement.RI.sln file. The assemblies are the following:

- **Contoso.TrainingManagement.Common.** This assembly contains constants that are used throughout the Training Management application.
- **Contoso.TrainingManagement.ServiceLocator.** This assembly implements the Service Locator pattern for the components that are provided by Training Management application. For more information about the service locator assembly, see [Service Locator Assembly](#) in this guidance. For more information about the Service Locator pattern, see [Service Locator](#) on MSDN.
- **Contoso.TrainingManagement.Web.** This project defines the ASPX pages that are used by the Training Management application. It does not have an assembly. The files in this project are included in the Contoso.TrainingManagement project as links.
- **Contoso.TrainingManagement.** This assembly defines the data structures and lists that are used by the SharePoint Web site. It also includes presenter helper classes that process the application's ASPX pages. ASPX pages handle HTTP requests by invoking methods of Web presenter classes that validate the input and format the responses.
- **Contoso.TrainingManagement.Repository.** This assembly provides business entity and repository classes that act as an interface to the list items stored in SharePoint.
- **Contoso.AccountingManagement.** This assembly contains the interface to the financial operations.
- **Contoso.HRManagement.** This assembly contains the interface to the human resources (HR) information.
- **Contoso.TrainingManagement.Workflows.** This project defines the workflows that are used by the Training Management application. It does not have an assembly. The files in this project are included in the Contoso.TrainingManagement.Workflows.RegistrationApproval project as links.
- **Contoso.TrainingManagement.Workflows.RegistrationApproval.** This project is used to package the registration approval workflow as a SharePoint feature in a SharePoint solution.

Structure of the Application

The Training Management application interacts with SharePoint in several ways. The following figure illustrates the structure of the Training Management application.

Training management application assemblies



The preceding diagram is color coded so that you can determine which assembly contains each element of the Training Management application. The **Contoso.TrainingManagement.Common** and **Contoso.TrainingManagement.ServiceLocator** assemblies are not shown. They are utilities that are used by all parts of the application. For more information about the service locator, see [Service Locator Assembly](#).

Note:

The diagram shows the unit testing domain. This is the set of components for which unit tests have been written. One of the design goals of the Training Management application is to demonstrate how unit testing methodologies can be incorporated into applications that are based on the SharePoint platform. For more information, see [Unit Testing with Mock Objects](#) in this guidance.

Interactions Between Components

The arrows in the preceding diagram represent method invocations or the initiation of an activity. These interactions are the following:

- **IIS to SharePoint.** SharePoint pages are loaded as a result of HTTP requests that are received by Microsoft Internet Information Services (IIS).
- **IIS to Training Management ASPX pages and Web Parts.** The Training Management application's ASPX pages and Web Parts are loaded as a result of HTTP requests that are received by IIS.
- **Training Management content definitions to SharePoint.** The Training Management application includes content type and list definitions for SharePoint.
- **ASPX pages and Web Parts to Training Management Web presenter classes.** The ASPX pages and Web Parts provided by the Training Management application invoke methods of helper classes named presenters. When there is an HTTP request, these methods validate the user input and calculate the HTTP response.

- **Presenter classes to repository classes.** Web presenters call methods of the Training Management repository classes. These methods query and update the Training Management application's list of course registrations.
- **Presenter classes to HR management.** The application's presenters invoke methods of the **HRManagement** class to query for human resources data.
- **Presenter classes to Accounting Management.** The application's presenters invoke methods of the **AccountingManagement** class. These methods query and update the accounting database.
- **Repository classes to SharePoint.** The repository layer interacts with SharePoint's data store by invoking methods of classes provided by SharePoint.
- **SharePoint to Windows Workflow Foundation.** When updates occur in SharePoint, such as the addition of a new registration list item, they trigger event activities in the Windows Workflow Foundation workflow engine.
- **Windows Workflow Foundation to SharePoint.** The workflow engine can initiate SharePoint activities. For example, the workflow engine can initiate the creation or deletion of SharePoint tasks.
- **Training Management workflow to Windows Workflow Foundation.** The Training Management workflow component provides the definition of the workflow rules that are used by the workflow engine. To learn more about the workflow, see [Designing the Registration Approval Workflow](#).
- **Windows Workflow Foundation to Training Management workflow.** The workflow engine invokes methods of the Training Management workflow classes when events or workflow-initiated activities occur.
- **Training Management workflow to workflow controller class.** When processing events or workflow-initiated activities, the application's workflow activities invoke methods of a helper class known as a workflow controller. This division of responsibility is motivated by the unit testing architecture chosen for the Training Management application.
- **Workflow controller class to repository classes.** The workflow controller class invokes data access methods that are provided by the **Repository** classes.
- **Workflow controller class to HRManagement classes.** The workflow controller class invokes data access methods that are provided by the **HRManagement** classes.
- **Workflow controller class to AccountingManagement classes.** The workflow controller class invokes data access methods that are provided by the **AccountingManagement** classes.

Application Classes

This section summarizes the classes that are defined in the Training Management application. To determine the assembly that contains each class, see the diagram of the application structure earlier in this topic.

ASPX Page Classes

The ASPX page classes are the following:

- **CourseRegistration.** This class derives from **System.Web.UI.Page**. It was created with a Visual Studio designer. It handles ASPX processing for course registrations.
- **RegistrationApproval.** This class derives from **System.Web.UI.Page**. It was created with a Visual Studio designer. It handles ASPX processing for registration approval.

Training Management Web Presenter Classes

The Training Management Web presenter classes are the following:

- **CourseRegistrationPresenter.** This class is used when processing a course registration HTTP request. It interacts with the **TrainingCourseRepository** and **RegistrationRepository** classes to validate user input and format the HTTP response.
- **RegistrationApprovalPresenter.** This class is used when processing a registration approval HTTP request. It interacts with the **TrainingCourseRepository** and **RegistrationRepository** classes to validate user input and format the HTTP response.
- **DirectReportsPresenter.** This class presents a manager's direct reports using information from the **HRManagement** component. The **DirectReportsPresenter** class is used by the **DirectReports** Web Part that is on the dashboard.
- **TrainingBudgetPresenter.** This class presents the training budget for each department based on information from the **AccountingManagement** component. It is used by the **TrainingBudget** Web Part that is on the dashboard.

Training Management Repository Classes

The Training Management repository classes are the following:

- **TrainingCourseRepository.** This class is the data interface to the list of training courses.
- **RegistrationRepository.** This class is the data interface to the list of registration requests.
- **ListItemRepository.** This class is a helper class that simplifies the calls that modify SharePoint list items.

Training Management Workflow Controller Class

The **RegistrationApprovalWorkflowController** class provides the business logic that controls the workflow for registration approvals.

Training Management Workflow Class

The **RegistrationApprovalWorkflow** class is a subclass of the **SequentialWorkflowActivity** class, which is provided by Windows Workflow Foundation. This class contains methods created by the workflow designer hosted in Visual Studio and programmer-written code that invokes methods of the **RegistrationApprovalWorkflowController** class to handle the execution of workflow activities.

HR Management Class

The **HRManager** class is the interface to the human resources information.

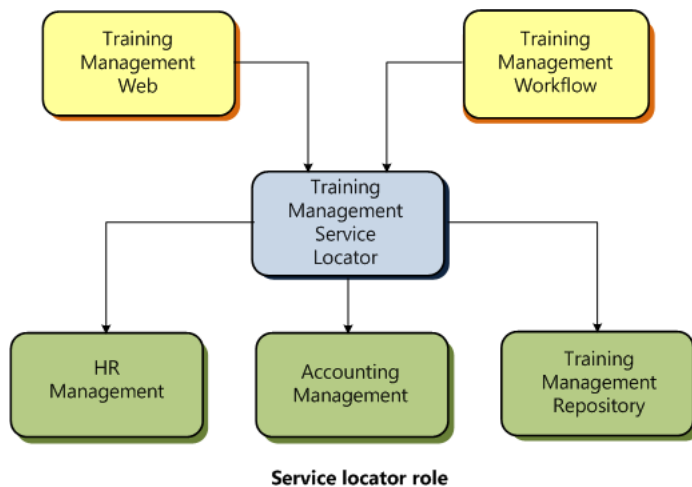
Accounting Manager Class

The **AccountingManager** class is the interface to the accounting database.

Service Locator Assembly

The service locator assembly includes a service locator class. This class instantiates services that are used in the application. The following figure illustrates this.

Service locator



This assembly implements the Service Locator pattern for the components that are provided by the Training Management application. For more information about the Service Locator pattern, see [The Service Locator Pattern](#) in this guidance and [Service Locator](#) on MSDN.

The Training Management application's **ServiceLocator** class locates services and also functions as an object factory. In the released version of the Training Management application, the **Get<T>** method constructs and returns a new object instance each time it is called. This behavior is intentional. In earlier implementations, the **ServiceLocator** class maintained a set of object instances that were shared across the application. However, this increased the risk of threading issues because the object instances that were accessed through the **ServiceLocator** class could be simultaneously used by multiple users. If correct thread locking or transaction management is not in place, the object can be in an inconsistent state when it is accessed. Constructing and returning new object instances for each call to the **Get<T>** method discourages storing shared state in instance-scoped member variables. If you must use shared state, carefully plan where the shared state is stored and how it is accessed.

Training Management Application Walkthrough

This topic describes the implementation of the Training Management application. It includes the following, as applicable:

- A description of how a use case or the workflow is implemented
- Examples of code that show relevant parts of the implementation
- Other useful information, such as where to obtain more information

The Training Management application can be thought of as a combination of use cases and a workflow. This guidance includes the following topics:

- [Create a New Course Use Case](#). This topic describes how to create a training course and add it to the list of training courses.
- [Register for a Course Use Case](#). This topic describes how to register for a training course.
- [Registration Approval Use Case](#). This topic describes how to approve or reject a course registration.
- [Registration Workflow Implementation](#). This topic describes how the workflow adds a new registrant to the list of employees that are registered for courses.
- [View the Manager Dashboard Use Case](#). This topic describes the manager dashboard.
- [View the Training Dashboard Use Case](#). This topic describes the training dashboard.

Create a New Course Use Case

Users create new courses by selecting **Add New Item** on the Training Management SharePoint Web site. They are directed to a form that requires the following information:

- A course title
- An eight-character course code
- A course description
- An enrollment deadline
- The course's start date
- The course's end date
- The cost of the course
- The difficulty level of the course

The following illustration shows the form.

Create a course entry form

root > training > Training Courses > New Item

Training Courses: New Item

Attach File * indicates a required field

Title * Beginning C#

Code * ProgC101

Description
This is an introductory course in C#. No programming experience is required.

Enrollment Deadline * 10/20/2008

Start Date * 11/10/2008

End Date * 11/13/2008

Cost * 200

Difficulty Level 1

After filling out the form, the user submits it by clicking **OK**. The application applies the appropriate business rules to validate the user input. If valid, a new course is created by adding it to the training course repository, which is a higher-level abstraction of the Training Course SharePoint list. For information about this list, see [The Training Course List](#).

After the course is created, it appears on the training dashboard page of the Training Management application with information from the form. The following illustration shows the new entry.

New course entry

Training Courses						
Title	Code	Enrollment Deadline	Start Date	End Date	Cost	Difficulty Level
Beginning C# NEW	ProgC101	10/20/2008	11/10/2008	11/13/2008	\$200.00	1
Office 2007 Training NEW	OFFICE07	12/31/2010	1/1/2011	1/6/2011	\$100.00	2
Add new item						

The column headings, such as Title and Code, are fields that are defined in a SharePoint content type named **TrainingCourseContentType**. For more information, see [The Training Course Content Type](#).

The Training Course Content Type

A content type defines various elements that characterize your data. For example, a content type can include the metadata that is associated with the data, the workflows, and the event receivers. The Create a Course use case includes one content type that is named **TrainingCourseContentType**. The content type is defined in the TrainingCourseContentType.xml file located in the ContentTypes\TrainingCourseContentType folder of the Contoso.TrainingManagement project.

The **TrainingCourseContentType** includes seven fields and two event receivers. The following are the **TrainingCourseContentType** fields:

- **TrainingCourseCode**. This field is for the code that identifies the course.
- **TrainingCourseDescription**. This field is for the description of the course.
- **TrainingCourseEnrollmentDate**. This field is for the last day that an employee can enroll in the course.
- **TrainingCourseStartDate**. This field is for the date the course begins.
- **TrainingCourseEndDate**. This field is for the date the course ends.
- **TrainingCourseCost**. This field is for the cost of the course.
- **TrainingCourseDifficultyLevel**. This field is for the course's level of difficulty.

The following XML from the TrainingCourseContentType file shows two of the fields.

XML

```
<Field ID="{e5509750-cb71-4de3-873d-171ba6448fa5}" Type="Text" Name="TrainingCourseCode"
DisplayName="Code" Hidden="FALSE" Required="TRUE" Sealed="FALSE" MaxLength="8"
DisplaceOnUpgrade="TRUE" />
<Field ID="{8E39DAD4-65FA-4395-BA0C-43BF52586B3E}" Type="Note"
Name="TrainingCourseDescription" DisplayName="Description" Hidden="FALSE" Required="FALSE"
Sealed="FALSE" RichText="TRUE" DisplaceOnUpgrade="TRUE" />
```

The form that SharePoint generates to enter the course information is based on the field type. For example, the **TrainingCourseDescription** field's type is **Note**, so SharePoint includes a text box. For more information about each attribute, see [Field Element \(List – Definition\)](#) on MSDN. Each field corresponds to a column in the Training Courses list. The **DisplayName** attribute defines the name of the column.

The content type also has two list item event receivers named **ItemAdding** and **ItemUpdating**. For more information about them, see [The List Item Event Receivers](#).

For more information about content types, see [Content Types](#) on MSDN.

The List Item Event Receivers

The Training Management application includes two list item event receivers; they are named **ItemAdding** and **ItemUpdating**. When managers create a course, the **ItemAdding** event receiver first validates the information before the course is added to the list. When you create a content type and select **Add with Event Receiver** in the **Content Type Settings** dialog box, Visual Studio automatically generates a file named *name* ItemEventReceiver.cs and a file named *name*ListEventReceiver.cs.

The **ItemAdding** and **ItemUpdating** event receivers are defined in the **TrainingCourseItemEventReceiver** class. This class must inherit from the **SPItemEventReceiver** class to access the event receivers.

The **SPItemEventProperties** class is provided by the SharePoint platform. It contains the following properties that are used by the list item event receivers:

- **ErrorMessage**. Set this property to a text string that explains the validation failure.
- **Cancel**. Set this property to **true** if you want to fail the validation; if you do this, SharePoint produces an error message and does not add the item to the list.
- **AfterProperties**. Use this property to retrieve the values you want to validate.

For more information, see [SPItemEventProperties](#) on MSDN.

The following is the code for the business rules.

```
C#
public override void ItemAdding(SPItemEventProperties properties)
{
    bool isValid = true;
    StringBuilder errorMessage = new StringBuilder();

    string title = string.Empty;
    string code = string.Empty;
    DateTime enrollmentDate = DateTime.MinValue;
    DateTime startDate = DateTime.MinValue;
    DateTime endDate = DateTime.MinValue;
    float cost = 0;

    // A Web is required to retrieve Field names for iterating through the
    // ListItem collection in the SPItemEventProperties and for
    // retrieving existing items in the TrainingCourses list.
    using (SPWeb web = properties.OpenWeb())
    {
        ITrainingCourseRepository repository =
        ServiceLocator.GetInstance().Get<ITrainingCourseRepository>();
        Initialize(properties.AfterProperties, repository, web, out title, out code, out
        enrollmentDate, out startDate, out endDate, out cost);

        if (this.ValidateCourseCodeExists(repository, code, errorMessage, web))
        {
            isValid = false;
        }

        isValid = isValid & this.ValidateCourseCode(code, errorMessage);
        isValid = isValid & this.ValidateEnrollmentDate(enrollmentDate, errorMessage);
        isValid = isValid & this.ValidateStartDate(enrollmentDate, startDate, errorMessage);
        isValid = isValid & this.ValidateEndDate(startDate, endDate, errorMessage);
        isValid = isValid & this.ValidateCourseCost(cost, errorMessage);

        // If any of the rules fail, set an error message and set the
        // Cancel property to true to instruct SharePoint to redirect to
        // a standard error page.
        if (!isValid)
        {
            properties.ErrorMessage = errorMessage.ToString();
            properties.Cancel = true;
        }
    }
}
```

If the course information violates any of these rules, the user receives an error message that explains the problem. This message is displayed on the standard SharePoint error page.

The **ItemUpdating** list item event receiver is invoked when a user edits information about a course that already exists. The only difference between it and the **ItemAdding** event receiver is that it does not validate the enrollment date.

If there is an unhandled exception, SharePoint catches it and displays a standard error page with a generic message. If you catch exceptions in your own code, you can create your own error messages and cancel the operation to the list item.

The Training Course List

After you define the content types, you can create lists that display those content types. Lists are containers of rows and columns; they are similar to database tables. The SharePoint platform provides interfaces to manage lists that are very easy to use. In many situations, lists are preferable to databases because, unlike many relational database management systems, they require no specialized knowledge. Because of the simplicity of lists, the Training Management application uses them as the primary method of storing data. For more information about when to use lists and databases, see [Using SharePoint Lists vs. Database Tables](#). SharePoint lists are instances of the **SPList** class. For more information, see [SPList Class \(Microsoft.SharePoint\)](#) on MSDN.

The Training Management application uses a list to display the available training courses. This list is based on the **TrainingCourse** content type. If you use Visual Studio extensions for Windows SharePoint Services, you supply only a pre-existing content type to create this list. The following procedure demonstrates how to create a list that is based on a content type.

To create a list definition from a content type

1. In Visual Studio, right-click the **Contoso.TrainingManagement** project, point to **Add**, and then click **New Item**.
2. In the Categories pane, click **SharePoint**. In the Templates pane, click **List Definition from Content Type**.
3. In the **Name** text box, type the name of the list, and then click **Add**.
4. In the **List Definition Type** drop-down box, click the content type. If you want to create an instance of the list, select the check box. Click **OK**.

Visual Studio generates three XML files when it creates a list. For the **TrainingCoursesListDefinition** in the Training Management application, the following are the names of the XML files:

- Schema.xml
- TrainingCourseListInstance.xml
- TrainingCourseListTemplate.xml

The next sections describe these files.

The Schema File

The Schema.xml file defines the views, forms, toolbar, and special fields for lists that are created with the List Definition template. The schema is used wherever the associated **SPList** instance occurs. It includes definitions of two views that are identified as **BaseViewID=0** and **BaseViewID=1**. The Trainingdashboard.aspx page uses **BaseViewID=0**. The page where users can create a course uses **BaseViewID=1**. The **ViewFields** element defines the field references that appear on the list page. This is shown in the following XML.

XML

```
<ViewFields>
  ...
  <FieldRef Name="TrainingCourseCode">
  </FieldRef>
  <FieldRef Name="TrainingCourseEnrollmentDate">
  </FieldRef>
  <FieldRef Name="TrainingCourseStartDate">
  </FieldRef>
  <FieldRef Name="TrainingCourseEndDate">
  </FieldRef>
  <FieldRef Name="TrainingCourseCost">
  </FieldRef>
  <FieldRef Name="TrainingCourseDifficultyLevel">
  </FieldRef>
</ViewFields>
```

Each field reference is associated with a field that is also included in the schema file. Each field has a variety of attributes associated with it, including **Name**, which is the name that appears as one of the categories on the Training Courses list view. For example, the **TrainingCourseCode** field has a display name of Code.

The List Template File

The TrainingCourseListTemplate.xml file is the template for the training course list instances.

The List Instance File

The TrainingCourseListInstance.xml file defines a feature ID that establishes a deployment dependency. This means that an instance of the Training Course list appears when you create a site based on a site definition that has the feature referenced by the feature ID.

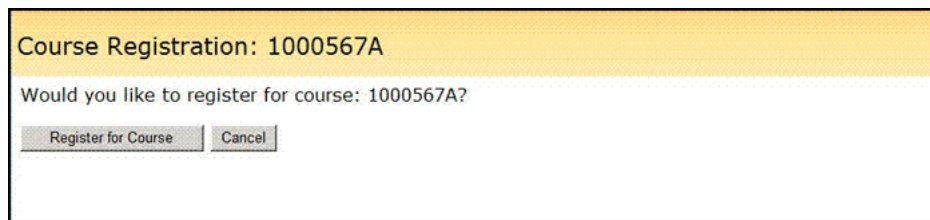
Register for a Course Use Case

The Register for a Course use case demonstrates how employees can register for a training course. The code that implements this use case demonstrates how an application can query SharePoint list items and add a new item to a list.

Using Custom Action

To begin the registration process, you can use either of two custom actions. One option is to navigate to the DisplayForm.aspx page for a specific training course and select the **Register for Course** action. The other option is to select the **Register for Course** action from the course's context menu that is located on the training dashboard page or the Training Course list page. You can also make a direct HTTP request for the CourseRegistration.aspx page (<http://myserver/sites/training/CourseRegistration.aspx?ID=1>). This request initiates the **Page_Load** handler that validates the request and calculates the fields that are used in the ASPX page. If the request passes this process, the employee sees the course registration form. The following illustration shows the form.

Training Management registration form



Clicking the **Register for Course** button adds a new item to the SharePoint registration list. This list contains items of the application-provided **Registration** content type.

To define a custom action, use the **CustomAction** element. The following XML (CourseRegistrationElement.xml) adds the Register for a Course action to the DisplayFormToolbar and to the EditControlBlock. It is located in Contoso.TrainingManagement\Forms\CourseRegistration.

XML

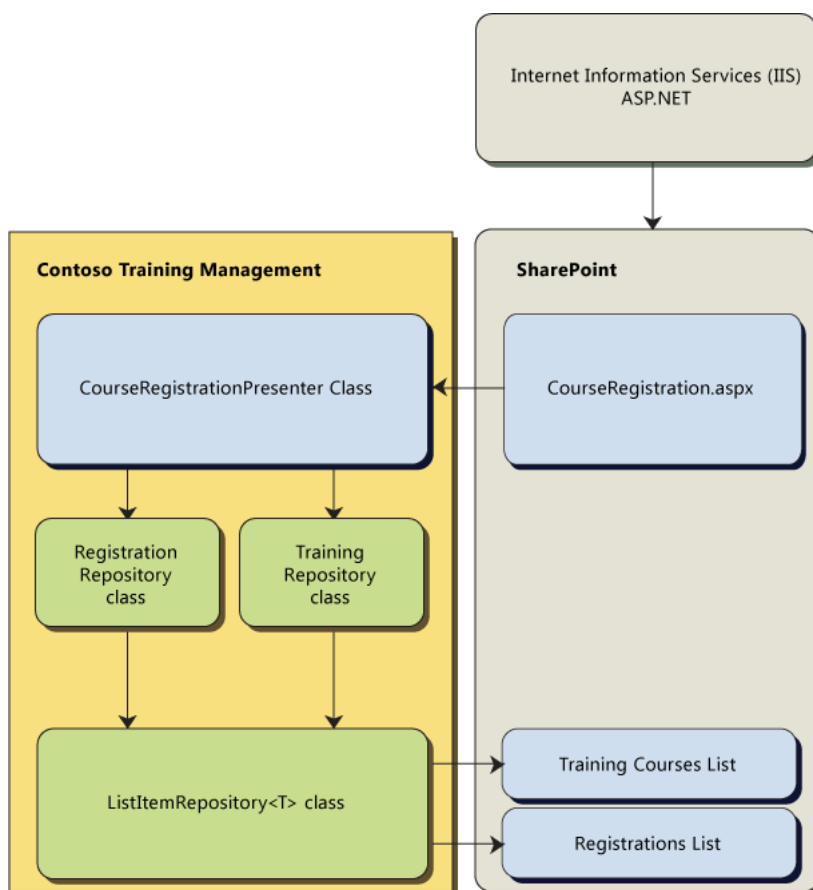
```
<CustomAction
  Id="695F3959-79C5-48b0-AACC-30DF5A731C81"
  Location="DisplayFormToolbar"
  Title="Register for Course"
  RegistrationType="ContentType"
  RegistrationId="0x01000CEE433CD7484DC5ADBEC636A5DD0C07">
  <UrlAction Url="~site/CourseRegistration.aspx?ID={ItemId}" />
</CustomAction>
<CustomAction
  Id="0F446E21-839F-4938-9F1C-914237F15694"
  Location="EditControlBlock"
  Title="Register for Course"
  RegistrationType="ContentType"
  RegistrationId="0x01000CEE433CD7484DC5ADBEC636A5DD0C07">
  <UrlAction Url="~site/CourseRegistration.aspx?ID={ItemId}" />
</CustomAction>
```

The **Id** attribute is a programmer-generated GUID. The **RegistrationType** attribute associates the action with a content type and the **RegistrationId** attribute is the ID of the training course content type. The **UrlAction** attribute specifies the HTTP request that is generated when the user selects the action. The **{ItemId}** expression is replaced at run time with the ID of the selected list item.

Classes Used by the Course Registration Use Case

The following illustration shows the control flow for handling HTTP requests for the Course Registration use case.

Course registration control flow



Defined in Contoso.TrainingManagement project



Defined in Contoso.TrainingManagement.Repository project

The arrows in the preceding diagram represent the direction of the control flow. The following five classes are involved in the control flow:

- **CourseRegistration**. This class is the ASPX page definition produced by the Visual Studio designer. Internet Information Services (IIS) invokes handler methods of the **CourseRegistration** class when it receives HTTP requests.
- **CourseRegistrationPresenter**. This is the presenter for the CourseRegistration.aspx page. For more information about the presenter, see [The Model-View-Presenter \(MVP\) Pattern](#) and [Unit Testing with Mock Objects](#) in this guidance.
- **TrainingCourseRepository**. This class provides Create/Update/Delete operations for the Training Courses SharePoint list.
- **RegistrationRepository**. This class provides Create/Update/Delete operations for the SharePoint registration list.
- **ListItemRepository**. This class encapsulates some of the details of interacting with SharePoint lists.

Registration Phases

The Registration use case occurs in two phases:

- **Page_Load** phase
- **Submit_Click** phase

These phases correspond to rendering the initial registration page in response to the HTTP request from an employee and to processing the button click when the employee confirms the request for a new registration.

The Registration use case includes the following topics:

- [Course Registration Page Load](#). This topic explains how a Web page is loaded when an employee submits an HTTP request.
- [Adding an Item to the Registration List](#). This topic explains how a new item is added to the registration list. The registration list contains course registration requests.
- [The Registration Content Type](#). This topic explains the content type that defines an item in the registration list.

- [The Registration List](#). This topic explains the structure of the registration list.

Course Registration Page Load

The page load process begins when an employee submits an HTTP request, such as `http://myserver/sites/training/CourseRegistration.aspx?ID=1`, to begin the registration process. The HTTP request initiates the **Page_Load** handler. It validates the request and calculates the fields that are used in the ASPX page. The page load process is an example of how to programmatically query for items in a SharePoint list. The diagram shown in the previous topic, [Register for a Course Use Case](#), illustrates the control flow of this use case.

The following sections explain the page load process by examining the methods that are called. The calls occur in the following order:

1. [CourseRegistration.Page_Load](#). Internet Information Services (IIS) invokes the **Page_Load** method of the **CourseRegistration** class when it receives the initial HTTP request.
2. [CourseRegistrationPresenter.RenderCourseRegistrationView](#). This method validates the registration context and calculates the values needed to display the registration form. To see an example of the registration form, see [Register for a Course Use Case](#).
3. [CourseRegistrationPresenter.GetCourse](#). This method confirms whether the course ID that is provided as an argument corresponds to one of the Training Course list items in SharePoint. If the course ID corresponds, the course information is returned through an out parameter.
4. [TrainingCourseRepository.Get](#). This method queries for data in the Training Courses list.
5. [BaseEntityRepository<T>.GetListItem](#). This code demonstrates how to create and configure an instance of the **SPQuery** class.
6. [ListItemRepository.Get](#). This code locates an item within a SharePoint list.

The following sections describe the work that is performed by each of these methods during the registration page load.

The Page_Load Method

This is the code for the **Page_Load** method that is invoked by ASP.NET. The code is located in `CourseRegistration.aspx.cs` located in the `Contoso.TrainingManagement.Web` project.

```
C#
protected void Page_Load(object sender, EventArgs e)
{
    if ( !Page.IsPostBack )
    {
        CourseRegistrationPresenter courseRegistrationPresenter = new
CourseRegistrationPresenter(this);
        courseRegistrationPresenter.RenderCourseRegistrationView(SPContext.Current.Web,
SPContext.Current.Web.CurrentUser.LoginName);

        this.CourseList.DataValueField = "Id";
        this.CourseList.DataTextField = "Code";
        this.CourseList.DataBind();
    }
}
```

The `CourseRegistration` page's **Page_Load** handler gets its data from the following two sources that together represent the current execution context:

- The ASP.NET **Request** class. The **Request** class gives access to the parameters passed in as part of the HTTP request.
- The SharePoint **SPContext** class. The **SPContext** class is a SharePoint class that provides information about the run-time context of the SharePoint application. The **SPContext.Current** property contains an object that provides information about the SharePoint context that exists when the page loads. The **Web** property of this context object gives the current SharePoint Web site. The type of the Web site is **SPWeb**.

Note:

The **SPWeb** object returned by the **SPContext.Current.Web** property returns an object that represents the current SharePoint site. This object contains methods and properties to access the users, lists, and other content elements of the SharePoint application. For more information, see [SPWeb Class \(Microsoft.SharePoint\)](#) on MSDN.

The **Page_Load** method constructs a presenter (**CourseRegistrationPresenter**) and invokes the **CourseRegistrationPresenter.RenderCourseRegistrationView** method.

The **CourseRegistrationPresenter** class validates the registration data and sets the page properties that are required to present the registration form to the user. For example, there is an error message if the user has already registered for the course. This next section describes this class.

Rendering the Course Registration View

The **RenderCourseRegistrationView** method of the **CourseRegistrationPresenter** class validates the registration context and calculates the values needed to display the registration form. This form is illustrated at the beginning of the [Register for a Course Use Case](#). The **RenderCourseRegistrationView** is invoked by the **CourseRegistration** class as part of the page load process.

The **RenderCourseRegistrationView** method calls the following two private helper methods through the **GetCourseUserRegistration** helper method:

- **GetCourse**. This method checks to see whether the course ID provided as an argument corresponds to one of

the Training Course list items in SharePoint.

- **GetUserRegistration.** This method checks that the registration request does not already exist. This method is not shown in the following code because it is similar to the **GetCourse** method.

If all the conditions are not met, an error page is displayed.

The following is the code for the **GetCourse** method.

```
C#
private bool GetCourse(SPWeb web, ICourseRegistrationView view,
                      int courseId, out TrainingCourse course)
{
    bool success = false;

    ITrainingCourseRepository trainingCourseRepository =
ServiceLocator.GetInstance().Get<ITrainingCourseRepository>();

    course = trainingCourseRepository.Get(courseId, web);

    if ( course != null )
    {
        success = true;
    }
    else
    {
        view.ContentMessage = "The course selected was not a valid.";
    }

    return success;
}
```

The preceding code demonstrates how the Service Locator pattern is used to access the Training Management application's **TrainingCourseRepository** component. The next section explains how the **TrainingCourseRepository** component's **Get** method queries SharePoint data. For more information about the Service Locator pattern, see [The Service Locator Pattern](#) in this guidance and [Service Locator](#) on MSDN.

Constructing a TrainingCourse Query Using CAML

The **TrainingCourseRepository** class includes a **Get** method that queries SharePoint for the Training Courses list item that corresponds to an integer ID. The code is located in the TrainingCourseRepository.cs file of the Contoso.TrainingManagement.Repository project.

```
C#
public TrainingCourse Get(int id, SPWeb web)
{
    StringBuilder queryBuilder = new StringBuilder("<Where>");
    queryBuilder.Append("<Eq><FieldRef Name='ID' />");
    queryBuilder.Append(string.Format("<Value Type='Integer'>{0}</Value></Eq>", id));
    queryBuilder.Append("</Where>");

    return GetListItem(queryBuilder.ToString(), web);
}
```

This method constructs an XML query string and passes it to the **GetListItem** method that is provided by its base class, **BaseEntityRepository<T>**. The XML query string restricts the search to list items that match the course ID. SharePoint queries use the Collaborative Application Markup Language (CAML). For more information about the SharePoint query syntax, see [Query Element \(Query\)](#) on MSDN. For tools to help you write CAML queries, see [CAML Query Tools](#).

Querying SharePoint via the ListItemRepository

The **GetListItem** method is defined in the **BaseEntityRepository<T>** class. The code is located in the BaseEntityRepository.cs file in the Contoso.TrainingManagement.Repository project.

```
C#
protected T GetListItem(string caml, SPWeb web)
{
    T entity = null;

    SPQuery query = new SPQuery();
    query.Query = caml;

    SPLListItem item = null;

    item = listItemRepository.Get(web, this.ListName, query);

    if (item != null)
    {
        entity = PopulateEntity(item);
    }
}
```

```
    return entity;
}
```

The preceding code demonstrates how to create and configure an instance of the **SPQuery** class. **SPQuery** objects use the CAML query language to execute data queries in SharePoint. The code creates an instance of the class and sets its **Query** property to a string in the CAML query syntax.

Next, the code invokes the **Get** method of an instance of a helper class named **ListItemRepository**. This class is provided by the application's **Contoso.TrainingManagement.Repository** component to encapsulate some aspects of interacting with SharePoint lists.

The following code is the **Get** method. It is located in the `ListItemRepository.cs` file in the `Contoso.TrainingManagement.Repository` project.

```
C#
public SPListItem Get(SPWeb web, string listName, SPQuery query)
{
    SPListItem item = null;
    SPListItemCollection collection = null;

    collection = web.Lists[listName].GetItems(query);

    if ( collection != null && collection.Count > 0 )
    {
        item = collection[0];
    }

    return item;
}
```

The collection indexer (using `[]` syntax) locates the list with the name provided by the **listName** parameter. In this case, it will be the string "Training Courses".

Next, the code invokes the **GetItems** method of the **SPList** class provided by SharePoint. This executes the CAML query against the list.

The result of the query is an **SPListItemCollection** object. In the Training Management application, queries may return either a collection with zero values or a single value. If the requested item is found, the collection will be nonempty and the item found will be returned; otherwise, the value **null** will be returned.

Adding an Item to the Registration List

After the [page loads](#), the registration form appears. The following illustration shows this form.

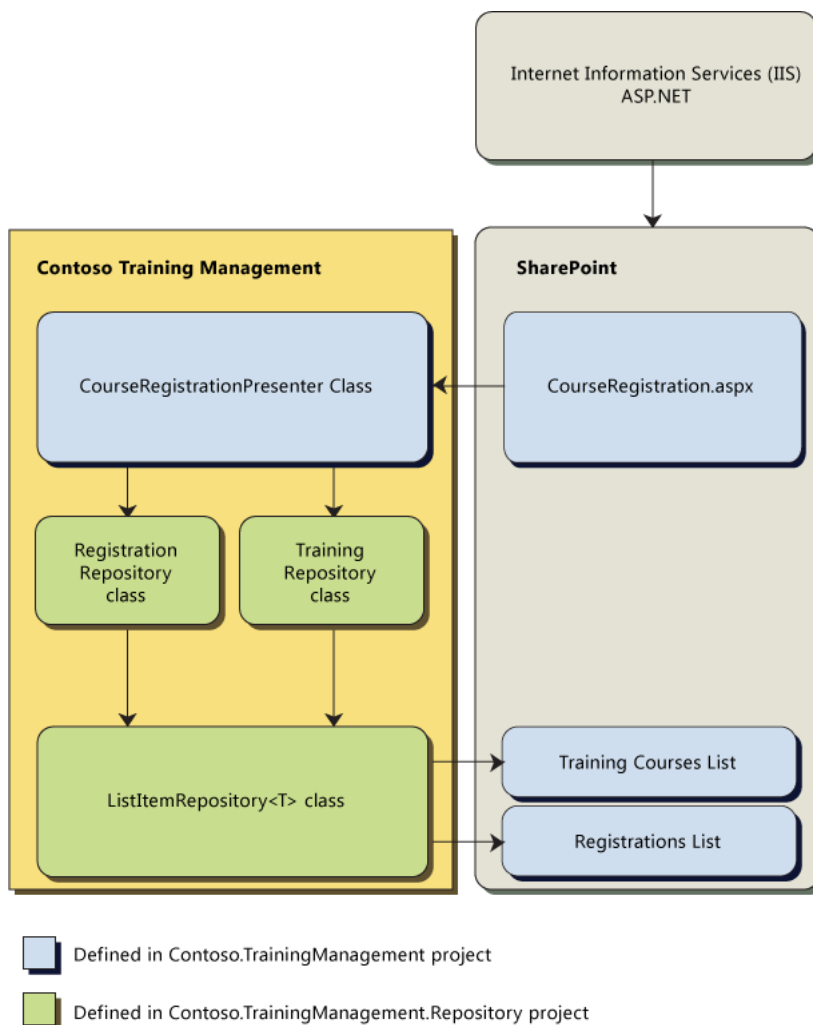
Training Management registration form

Course Registration: 1000567A

Would you like to register for course: 1000567A?

The employee clicks the **Register for Course** button to process the registration. The code associated with the button click is an example of how to programmatically add a new item to a SharePoint list. The following illustration shows the control flow of the use case.

Course registration control flow



Selecting the **Register for Course** button causes the **CourseRegistrationSubmit_Click** method to execute. The following sections explain the course registration process by examining the methods that are called. The calls occur in the following order:

1. [CourseRegistration.Submit_Click](#). This method is invoked by Internet Information Services (IIS) when the user clicks the **Register for Course** button.
2. [CourseRegistrationPresenter.Register](#). This method validates the arguments and invokes the repository layer to add a new list item.
3. [CourseRegistrationPresenter.PerformRegistration](#). This is a helper method of the presentation layer.
4. [RegistrationRepository.Add](#). This method is provided by the repository layer for adding a new registration list item.
5. [BaseEntityRepository<T>.AddListItem](#). This is a helper method provided by the Registration Repository's base class.
6. [ListItemRepository.Add](#). This is an application utility method for adding items to SharePoint lists.

The Submit_Click Handler

The following code shows the **CourseRegistrationSubmit_Click** method that handles the registration button click. It is located in the CourseRegistration.aspx.cs file in the Contoso.TrainingManagement.Web project.

```
C#
protected void Submit_Click(object sender, EventArgs e)
{
    CourseRegistrationPresenter presenter = new CourseRegistrationPresenter(this);
    presenter.Register(SPContext.Current.Web, SPContext.Current.Web.CurrentUser.LoginName);
}
```

The **Submit_Click** method is registered as the button's handler in the CourseRegistration.aspx file.

The **Register** method then calls SharePoint methods and ASP.NET methods to register the user for the course. The **PerformRegistration** method executes as part of the **Register** method. This code is in CourseRegistrationPresenter.cs.

Performing the Registration – Presentation Layer

The **CourseRegistrationPresenter.Register** method is the entry point into the presenter when adding a new registration. The method invokes a private helper method named **PerformRegistration**. The **PerformRegistration** method is located in the CourseRegistrationPresenter.cs file of the Contoso.TrainingManagement.Web project.

```
C#
private void PerformRegistration(SPWeb web, TrainingCourse course, SPUser user)
{
    Registration registration = new Registration();
    registration.Title = String.Format("{0} - {1}", course.Code, user.Name);
    registration.CourseId = course.Id;
    registration.UserId = user.ID;
    registration.RegistrationStatus = "Pending";

    IRegistrationRepository registrationRepository =
        ServiceLocator.GetInstance().Get<IRegistrationRepository>();

    int id = registrationRepository.Add(registration, web);
}
```

This code constructs a new **Registration** object and populates its properties with values that define the course registration to be added to the SharePoint list. The status of the new registration is **Pending**. The **Registration** class contains the same fields as [The Registration Content Type](#) in SharePoint.

The code demonstrates how the Service Locator pattern is used to access the Training Management application's **RegistrationRepository** class. This class is exposed through an interface named **IRegistrationRepository**. For more information about the Service Locator pattern, see [The Service Locator Pattern](#) in this guidance and [Service Locator](#) on MSDN.

Finally, the code invokes the **RegistrationRepository** class's **Add** method to add the item to the Registrations list stored in SharePoint.

Adding the List Item – Repository Layer

In the Registration Repository component of the Training Management application, a constructor and the following three methods are involved in processing the addition of a new Registrations list item:

- **RegistrationRepository.Add**
- **BaseEntityRepository<T>.AddListItem**
- **ListItemRepository.Add**

The following code is for the **RegistrationRepository.Add** method. This code is located in the RegistrationRepository.cs class of the Contoso.TrainingManagement.Repository project.

```
C#
public int Add(Registration registration, SPWeb web)
{
    return AddListItem(registration, web);
}
```

The **Add** method uses the functionality that is provided by the base class **BaseEntityRepository<T>**. The following code is the **BaseEntityRepository<T>.AddListItem** method. This code is located in the BaseEntityRepository.cs file of the Contoso.TrainingManagement.Repository project.

```
C#
// This is in class BaseEntityRepository<T>
protected int AddListItem(T entity, SPWeb web)
{
    Dictionary<Guid, object> fields = GatherParameters(entity, web);
}
```

```

        SPListItem item = null;

        item = listItemRepository.Add(web, this.ListName, fields);

        return (int)item[new Guid(Fields.Id)];
    }

    // This is in class RegistrationRepository
    protected override Dictionary<Guid, object>
        GatherParameters(Registration entity, SPWeb web)
    {
        Dictionary<Guid, object> fields = new Dictionary<Guid, object>();
        fields.Add(new Guid(Fields.Title), entity.Title);
        fields.Add(new Guid(Fields.CourseId), entity.CourseId);
        fields.Add(new Guid(Fields.UserId), entity.UserId);
        fields.Add(new Guid(Fields.User), web.SiteUsers.GetByID(entity.UserId));
        fields.Add(new Guid(Fields.RegistrationStatus), entity.RegistrationStatus);

        return fields;
    }

```

The preceding code creates a **Dictionary** object that contains values for each of the fields. It then creates an instance of the helper class named **ListItemRepository**. The **AddListItem** method calls the **ListItemRepository.Add** method to process the addition. It returns the ID of the newly created item.

The following code is for the **ListItemRepository.Add** method. This code is found in the `ListItemRepository.cs` file in the `Contoso.TrainingManagement.Repository` project.

```

C#
public SPListItem Add(SPWeb web, string listName, Dictionary<Guid, object> fields)
{
    SPListItem newItem = null;

    newItem = web.Lists[listName].Items.Add();

    foreach ( Guid key in fields.Keys )
    {
        newItem[key] = fields[key];
    }

    newItem.Update();

    return newItem;
}

```

In summary, creating a new list item is a three-step process:

1. Use the **SPListItemCollection** class's **Add** method to create a new empty list item in the Registrations list.
2. Set each field of the new item. The code uses the **Dictionary** object that was provided as an argument to the method invocation. The dictionary contains key/value pairs that specify each of the list item's field values.
3. Use the **SPListItem.Update** method to commit the changes.

The Registration Content Type

The [Register for a Course Use Case](#) relies on a registration list that contains registration list items.

The Registration content type is named **RegistrationContentType**. It is defined in Contoso.TrainingManagement\Content\Types\RegistrationContentType.xml. It inherits from the base content type, which is **Item**. The data elements of the content type are defined with the **Field Ref** element. Each **Field Ref** element refers to a type definition that is provided by a **Field** element. The **Field** elements are not nested within the **Content Type** element scope. This allows the **Field** type definitions to be used in more than one content type. The following is an example of a **Field** element.

XML

```
<Field ID="{11b6eba7-d1a1-4d15-9770-645052681e40}" Type="Integer" Name="CourseId"
  DisplayName="Course Id" Hidden="TRUE" Required="TRUE" Sealed="TRUE" ShowInEditForm="FALSE"
  DisplaceOnUpgrade="TRUE" />
```

The following is the corresponding **Field Ref** element.

XML

```
<FieldRef ID="{11b6eba7-d1a1-4d15-9770-645052681e40}" Name="CourseId" />
```

For more information about content types, see [Content Type](#) on MSDN. For information about how to create a custom content type, see [How to: Create a Custom Content Type with Event Receivers](#).

The Registration List

The [Register for a Course Use Case](#) relies on a registration list. The registration list contains course registration requests. It is defined by the following XML files:

- [Schema.xml](#)
- [RegistrationListTemplate.xml](#)
- [RegistrationListInstance.xml](#)

The Schema File

The Schema.xml file is located in the ListDefinitions\RegistrationListDefinition directory of the Contoso.TrainingManagement project; it defines the views, forms, toolbar, and special fields for lists that are created with the List Definition template. The schema is used wherever the associated **SPList** instance occurs. It includes definitions of four views, identified as **BaseViewID=0**, **BaseViewID=1**, **BaseViewID=2**, and **BaseViewID=3**. The default.aspx page is **BaseViewID=0**. The **BaseViewID=1** is the All Items page.

The **BaseViewID=2** view is used when the Registrations List View Web Part is provisioned on the training dashboard. This view uses the **ViewFields** element to display the registration status of the employee who is logged on. The registration status is one of the fields defined in the content type. The following is the **ViewFields** element.

XML

```
<ViewFields>
  ...
  <FieldRef Name="RegistrationStatus">
  </FieldRef>
</ViewFields>
```

A query retrieves the course registration. The following code shows the query.

XML

```
<Query>
  <Where>
    <Eq>
      <FieldRef Name="User" />
      <Value Type="Integer">
        <UserID />
      </Value>
    </Eq>
  </Where>
  <OrderBy>
    <FieldRef Name="RegistrationStatus" />
  </OrderBy>
</Query>
```

For information about the **Value** element, see [Value Element \(Query\)](#) on MSDN.

The **BaseViewID=3** view is used when the Registration List View Web Part is provisioned on the manager dashboard. This view uses the **ViewFields** element to display registrants whose registration status is **Pending**. The following is the **ViewFields** element.

XML

```
<ViewFields>
  ...
  <FieldRef Name="User">
  </FieldRef>
  <FieldRef Name="RegistrationStatus">
  </FieldRef>
</ViewFields>
```

A query retrieves the employee's course registrations that are pending. The following code shows the query.

XML

```
<Query>
  <Where>
    <Eq>
      <FieldRef Name="RegistrationStatus" />
      <Value Type="Text">Pending</Value>
    </Eq>
  </Where>
  <OrderBy>
    <FieldRef Name="Modified" Ascending="FALSE">
    </FieldRef>
  </OrderBy>
</Query>
```

For more information about SharePoint lists, see [SPList Class \(Microsoft.SharePoint\)](#) on MSDN. For more information about the SharePoint query syntax, see [Query Element \(Query\)](#) on MSDN.

The Registration List Template

The ListDefinitions\RegistrationListDefinition\RegistrationListTemplate.xml file in the Contoso.TrainingManagement project is the template for the Registrations list.

The Registration List Instance File

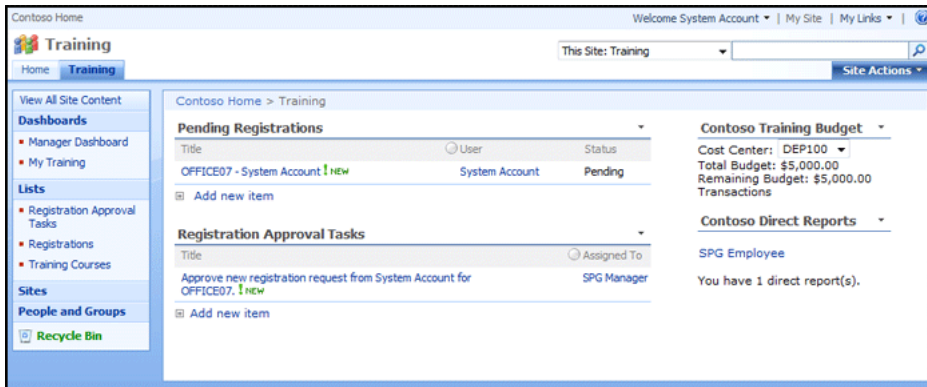
The ListDefinitions\RegistrationListDefinition\RegistrationListInstance file in the Contoso.TrainingManagement project defines a feature ID that establishes a deployment dependency. This means that an instance of the registration list appears when you deploy the application.

Registration Approval Use Case

The Registration Approval use case occurs when a manager approves or rejects a registration approval request that is submitted by an employee. The code that implements this use case demonstrates how to update a field of a list item and how to query for tasks.

Managers begin by selecting an approval task on their dashboards. The following illustration shows the manager dashboard.

The manager dashboard



Selecting an item from the Registration Approval Tasks Web Part generates an HTTP request. This request initiates the **Page_Load** handler that validates the request and calculates the fields that are used in the ASPX page. If the request passes this process, the manager sees the approval form that is shown in the following illustration.

Manager approval form

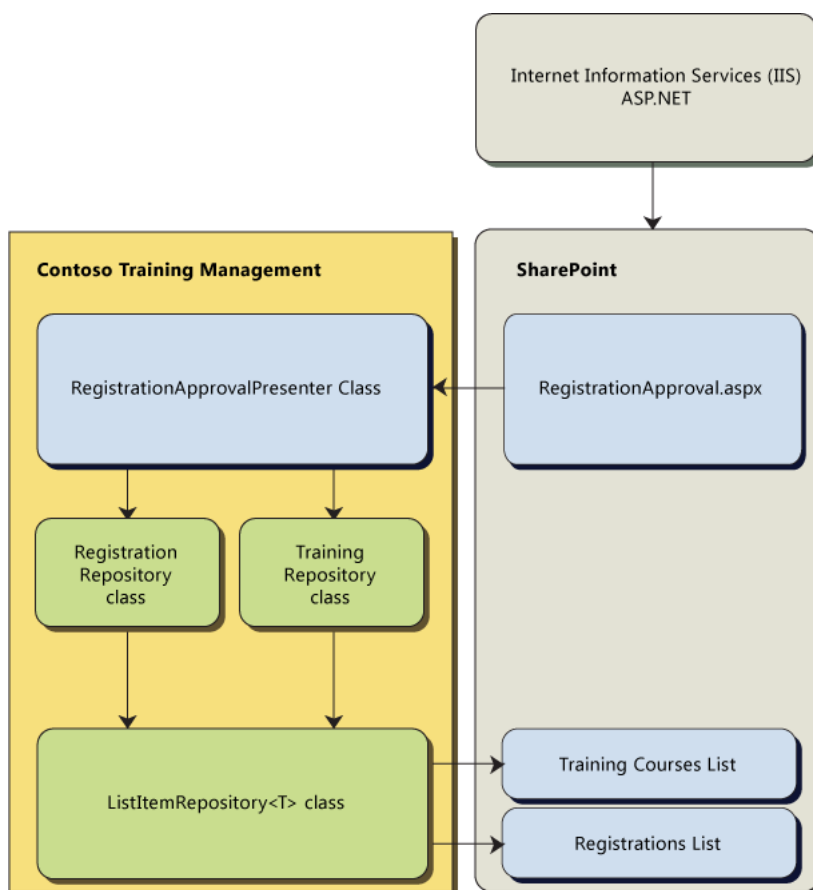


Selecting either **Approved** or **Rejected** from the drop-down list and clicking **Submit** causes the registration to be updated in the SharePoint list. The workflow process automatically generates the appropriate accounting entries. For more information, see [Registration Workflow Implementation](#).

Classes Used by the Registration Approval Use Case

The following illustration shows the control flow for handling HTTP requests for the Registration Approval use case.

Registration approval control flow



Defined in Contoso.TrainingManagement project



Defined in Contoso.TrainingManagement.Repository project

The arrows in the preceding diagram represent the direction of the control flow. The following five classes are involved:

- **RegistrationApproval**. This class is the ASPX page definition produced by the Visual Studio designer. Internet Information Services (IIS) invokes handler methods of the **RegistrationApproval** class when it receives HTTP requests.
- **RegistrationApprovalPresenter**. This class is the presenter for the RegistrationApproval.aspx.
- **TrainingCourseRepository**. This class is the interface to the SharePoint list of available courses.
- **RegistrationRepository**. This class provides an **Update** method that modifies the field values of an existing registration.
- **ListItemRepository**. This class encapsulates some of the details of interacting with SharePoint lists.

Registration Phases

The Registration Approval use case occurs in two phases:

- The **Page_Load** phase
- The **Submit_Click** phase

These phases correspond to rendering the initial approval form in response to the initial HTTP request from the manager and processing the button click when the manager confirms the request for a new registration. The second phase updates the registration list item.

The Registration use case discusses the following topics:

- [Registration Approval Page Load](#). This topic explains the page load process that begins when a manager submits an HTTP request to begin the registration process.
- [Updating a Registration List Item](#). This topic explains how to update an item in the Registrations list. In this case, the item's status is updated.

Registration Approval Page Load

The page load process begins when a manager submits an HTTP request to begin the registration approval process. (The registration approval workflow, discussed later, generates an approval task for each registration. Each approval task has a link to the Registration Approval page.) The HTTP request initiates the **Page_Load** handler. It validates the request and calculates the fields that are used in the ASPX page. The page load process is an example of how to programmatically query for items in a SharePoint list using a task list as an example.

The following sections explain the page load process by examining the methods that are called. The calls occur in the following order:

- [RegistrationApproval.Page_Load](#). IIS invokes the **Page_Load** method of the **RegistrationApproval** class when it receives the initial HTTP request.
- [RegistrationApproval.RenderView](#). This is a helper method that invokes the presentation layer.
- [RegistrationApprovalPresenter.RenderRegApprovalView](#). This method validates the registration approval context and calculates the values needed to display the page.
- [RegistrationApprovalPresenter.GetTaskAndRegistration](#). This method looks up the Registrations list item associated with the task ID provided as part of the request.
- [RegistrationApprovalTaskRepository.Get](#). This method retrieves a **RegistrationApprovalTask** item from the **RegistrationApprovalTasks** SharePoint list.
- [RegistrationRepository.Get](#). This method retrieves a Registration from the Registrations SharePoint list.

The following sections describe each of these methods.

The Page_Load Method

This is the code for the **Page_Load** method that is invoked by ASP.NET. The code is located in `RegistrationApproval.aspx.cs` located in the `Contoso.TrainingManagement.Web` project.

```
C#
[SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
protected void Page_Load(object sender, EventArgs e)
{
    Page.Validate();
    if ( !Page.IsPostBack || !Page.IsValid )
    {
        string sourceUrl = String.Empty;
        if ( Request.QueryString["Source"] != null )
        {
            sourceUrl = Request.QueryString["Source"];
        }

        RenderView(SPContext.Current.Web, Request.Params["ID"], sourceUrl);
    }
}

[SharePointPermission(SecurityAction.LinkDemand, ObjectModel = true)]
private void RenderView(SPWeb web, string taskID, string sourceUrl)
{
    RegistrationApprovalPresenter presenter =
        new RegistrationApprovalPresenter(this);
    bool success = presenter.RenderRegApprovalView(web, taskID);

    ContentMessage.Text = ContentMessage.Text.Replace("\r\n", "<br />");

    if ( success )
    {
        ViewState[taskIdViewStateKey] = taskID;
        if ( sourceUrl == String.Empty )
        {
            sourceUrl = String.Format("{0}/{1}", web.Url, managerDashboardPageFileName);
        }
        ViewState[sourceUrlViewStateKey] = sourceUrl;

        Status.DataBind();
    }
}
```

The `RegistrationApproval.aspx` page's **Page_Load** handler gets information about the execution context from the following two sources:

- The ASP.NET **Request** class. The **Request** class gives access to the parameters passed in as part of the HTTP request. The application uses the **QueryString** method to extract the ID of the task that is being processed.
- The SharePoint **SPContext** class. The **SPContext** class is a SharePoint class that provides information about the run-time context of the SharePoint application. The **SPContext.Current** property contains an object that provides information about the SharePoint context that exists when the page loads. The **Web** property of this context object gives the current SharePoint Web site. The type of the Web site is **SPWeb**.

 **Note:**

The **SPWeb** object that is returned by the **SPContext.Current** property represents the current SharePoint site. This object contains methods and properties to access the users, lists, and other content elements of the SharePoint application. For more information, see [SPWeb Class \(Microsoft.SharePoint\)](#) on MSDN.

The **Page_Load** method invokes a helper method named **RenderView**. It calls into the next layer of the Training Management application. This layer is implemented by the **RegistrationApprovalPresenter** class.

Rendering the Registration Approval View

The **RenderRegApprovalView** method of the **RegistrationApprovalPresenter** class validates the registration context and calculates the values needed to display the form shown at the beginning of the [Registration Approval use case](#). The **RenderRegApprovalView** method is invoked by the **RegistrationApproval** class as part of the page load process.

The **RenderRegApprovalView** method calls a private helper method named **GetTaskAndRegistration**. This method checks to see if the ID of the task that is one of the HTTP parameters is valid. It also validates the registration that is associated with this task. If all the conditions are met, the manager approval form displays.

The following is the code for the **GetTaskAndRegistration** method. It is located in the **RegistrationApprovalPresenter.cs** file.

```
C#
private bool GetTaskAndRegistration(SPWeb web, IRegistrationApprovalView view, string taskID,
out Registration registration)
{
    IRegistrationApprovalTaskRepository registrationApprovalTaskRepository =
ServiceLocator.GetInstance().Get<IRegistrationApprovalTaskRepository>();
    RegistrationApprovalTask registrationApprovalTask = null;
    if ( !String.IsNullOrEmpty(taskID) )
    {
        int queryID;
        if ( int.TryParse(taskID, out queryID) )
        {
            registrationApprovalTask = registrationApprovalTaskRepository.Get(queryID, web);
        }
    }

    if ( registrationApprovalTask == null )
    {
        view.Message = "The Approval Task selected is not valid.";
        registration = null;
        return false;
    }

    IRegistrationRepository registrationRepository =
ServiceLocator.GetInstance().Get<IRegistrationRepository>();
    int registrationID = registrationApprovalTask.WorkflowItemId;
    registration = registrationRepository.Get(registrationID, web);
    if ( registration == null )
    {
        view.Message = "The Registration associated with the selected Approval Task is not
valid.";
        return false;
    }

    return true;
}
```

This method uses the service locator to acquire instances of the **RegistrationApprovalTaskRepository** and the **RegistrationRepository**. Using the **taskID** parameter, the **RegistrationApprovalTaskRepository** instance is queried for a **RegistrationApprovalTask** item. If an item is found, the **RegistrationApprovalTask**'s **WorkflowItemId** property is then used to query the **RegistrationRepository**. The **WorkflowItemId** property provides the link between the **RegistrationApprovalTask** and the related registration.

Updating a Registration List Item

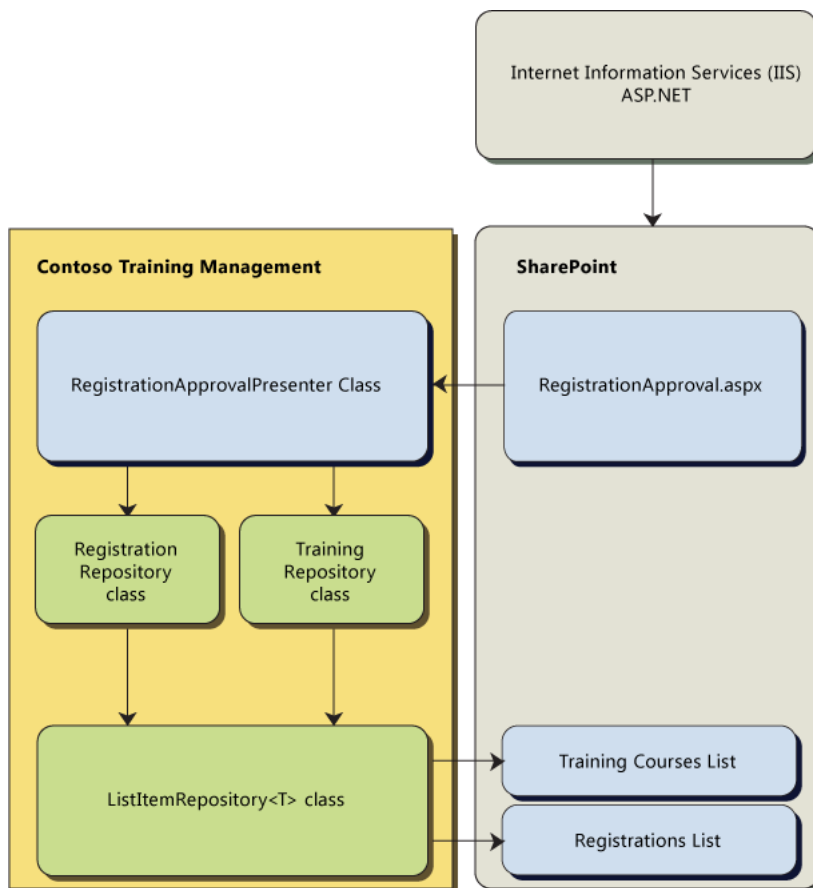
After the page loads, as discussed in [Registration Approval Page Load](#), the registration approval form appears. The following illustration shows this form.

Manager approval form



The manager clicks the **Submit** button to process the registration approval. The code associated with this button click is an example of how to programmatically update an item in a SharePoint list. The control flow is shown in the following illustration.

Registration approval control flow



 Defined in Contoso.TrainingManagement project

 Defined in Contoso.TrainingManagement.Repository project

Selecting the **Submit** button causes the **RegistrationApproval.Submit_Click** method to execute. The following sections explain the Registration Approval process by examining the methods that are called. The calls occur in the following order:

1. [RegistrationApproval.Submit_Click](#). This method is invoked by Internet Information Services (IIS) when the manager clicks the **Submit** button.
2. [RegistrationApproval.ProcessApproval](#). This is a helper method that invokes the presenter.

3. [RegistrationApprovalPresenter.ProcessApproval](#). This method has the business logic to process a registration approval.
4. [RegistrationRepository.Update](#). This method is provided by the repository layer for updating a registration list item.
5. [BaseEntityRepository<T>.UpdateListItem](#). This is a helper method provided by the Registration Repository's base class.
6. [ListItemRepository.Update](#). This method is an application utility method for updating items to SharePoint lists.

The Submit_Click Handler

The following code shows the **RegistrationApproval.Submit_Click** method that handles the registration approval button click. It is located in the Contoso.TrainingManagement.Web project in the RegistrationApproval\RegistrationApproval.aspx.cs file.

```
C#
protected void Submit_Click(object sender, EventArgs e)
{
    if ( Page.IsValid )
    {
        SPUtility.ValidateFormDigest();
        ProcessApproval(SPContext.Current.Web, (string)ViewState[taskIdViewStateKey],
        (string)ViewState[sourceUrlViewStateKey]);
    }
}

private void ProcessApproval(SPWeb web, string taskID, string sourceUrl)
{
    RegistrationApprovalPresenter presenter =
        new RegistrationApprovalPresenter(this);
    bool success = presenter.ProcessApproval(web, taskID, Status.SelectedValue);

    if ( success )
    {
        SPUtility.Redirect(sourceUrl, SPRedirectFlags.Default, HttpContext.Current);
    }
}
```

The **Submit_Click** method is registered as the button's handler in the RegistrationApproval.aspx file.

The **ProcessApproval** method continues the update by calling into the presenter through the **RegistrationApprovalPresenter.ProcessApproval** method. This method is described in [Performing the Registration – Presentation Layer](#). The **Status.SelectedValue** property gives the new status that will be used to update the registration associated with the current task ID.

Performing the Registration – RegistrationApprovalPresenter

The **RegistrationApprovalPresenter.ProcessApproval** method is the entry point into the presenter when approving or rejecting a registration. The **ProcessApproval** method is located in the RegistrationApprovalPresenter.cs file of the Contoso.TrainingManagement.Web project.

```
C#
public bool ProcessApproval(SPWeb web, string taskID, string registrationStatus)
{
    Registration registration = null;

    bool success = GetTaskAndRegistration(web, _view, taskID, out registration);

    if ( success )
    {
        registration.RegistrationStatus = registrationStatus;
        IRegistrationRepository registrationRepository =
            ServiceLocator.GetInstance().Get<IRegistrationRepository>();
        registrationRepository.Update(registration, web);
    }

    return success;
}
```

This code retrieves the **Registration** object that is associated with the current task ID. This is explained in more detail in [Rendering the Registration Approval View](#). The **Registration** class is described in more detail in [Register for a Course Use Case](#).

Next, the code uses the Service Locator pattern to access the Training Management application's **RegistrationRepository** class. This class is exposed through an interface named **IRegistrationRepository**. For more information about the Service Locator pattern, see [Service Locator](#) on MSDN.

Next, the code updates the **RegistrationStatus** field of the registration object to contain the new status provided by the **registrationStatus** field that is specified by the user. The new status will be **Approved** or **Rejected**.

Finally, the code invokes the **RegistrationRepository** class's **Update** method to update the item in the registration list stored in SharePoint.

Updating the Registration List Item – Repository Layer

In the Repository component of the Training Management application, the following three methods are involved in processing the updating a registration list item:

- **RegistrationRepository.Update**
- **BaseEntityRepository<T>.UpdateListItem**
- **ListItemRepository.Update**

The following code is for **RegistrationRepository.Update** method. This code is located in the **RegistrationRepository.cs** class of the **Contoso.TrainingManagement.Repository** project.

```
C#
public void Update(Registration registration, SPWeb web)
{
    UpdateListItem(registration, web);
}
```

The **Update** method uses the functionality provided by the base class **BaseEntityRepository<T>**.

The following code is the **BaseEntityRepository<T>.UpdateListItem** method. This code is located in the **BaseEntityRepository.cs** file of the **Contoso.TrainingManagement.Repository** project.

```
C#
// This is in class BaseEntityRepository<T>.
protected void UpdateListItem(T entity, SPWeb web)
{
    Dictionary<Guid, object> fields = GatherParameters(entity, web);

    listItemRepository.Update(web, this.ListName, entity.Id, fields);
}
```

The following code is the **RegistrationRepository.GatherParameters** method.

```
C#
// This is in class RegistrationRepository.
protected override Dictionary<Guid, object> GatherParameters(Registration entity, SPWeb web)
{
    Dictionary<Guid, object> fields = new Dictionary<Guid, object>();
    fields.Update(Fields.Title, entity.Title);
    fields.Update(Fields.CourseId, entity.CourseId);
    fields.Update(Fields.UserId, entity.UserId);
    fields.Update(Fields.User, web.SiteUsers.GetByID(entity.UserId));
    fields.Update(Fields.RegistrationStatus, entity.RegistrationStatus);

    return fields;
}
```

This code creates a **Dictionary** object that contains values for each of the fields.

The **UpdateListItem** method calls the **ListItemRepositoryUpdate** method to process the modification.

The following code is for the **ListItemRepository .Update** method. This code is found in the **ListItemRepository.cs** file in the **Contoso.TrainingManagement.Repository** project.

```
C#
public void Update(SPWeb web, string listName, int listItemId, Dictionary<Guid,
                                                                    object> fields)
{
    SPLListItem item = null;
    SPLListItemCollection collection = null;

    collection = web.Lists[listName].GetItems(this.BuildQuery(listItemId));

    if ( collection != null && collection.Count > 0 )
    {
        item = collection[0];

        foreach ( Guid key in fields.Keys )
        {
            item[key] = fields[key];
        }

        item.Update();
    }
}
```

In summary, updating a list item is a three-step process:

1. Get the **SPListItem** associated with the current list item ID. (The code for this is not shown.)
2. Set each field of the item. The code uses the **Dictionary** object that was provided as an argument to the method invocation. The dictionary contains key/value pairs that specify each of the list item's field values.
3. Use the **SPListItem.Update** method to commit the changes.

Registration Workflow Implementation

SharePoint has special features that work with Windows Workflow Foundation. These features allow long-running business processes, such as the routing process for approving a training registration, to retain state information, and to ensure that the business rules are properly followed. In the Training Management application, an instance of the approval workflow is initiated each time an employee registers for a course. More specifically, registering for a course translates into adding a new item to the Registrations list, which initiates an instance of the approval workflow.

There are two files that define the approval workflow. They are the Workflow.Designer.cs file and the Workflow.cs file. There is also a Registration Task Approval Content Type. The following topics are discussed in this section:

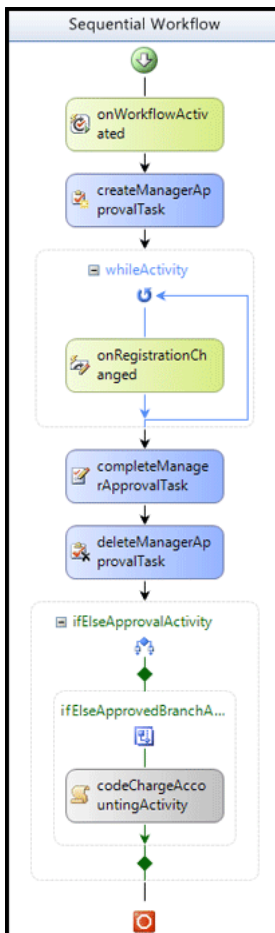
- [Designing the Registration Approval Workflow](#). This topic discusses the Workflow.Designer.cs file.
- [Executing the Registration Approval Workflow](#). This topic discusses what happens when an instance of the workflow executes.
- [The Registration Approval Task Content Type](#). This topic discusses the content type that defines the approval task that appears on the manager dashboard.

Designing the Registration Approval Workflow

The registration approval workflow that is located in Contoso.TrainingMangement.Workflows is a sequential workflow. Sequential workflows are similar to flow charts. Activities occur one after the other, from beginning to end. Like flow charts, sequential workflows can contain branches and loops. The registration approval workflow is created with the Visual Studio workflow designer. By using the workflow designer, you can specify activities that must occur in a particular sequence, in addition to identifying decision points and process loops.

The Workflow.Designer.cs file contains the code that corresponds to the property settings and sequence of activities that are depicted in the designer. Never directly edit this code; instead, use the designer to change the workflow. The SharePoint framework provides built-in functionality that connects SharePoint tasks to Windows Workflow Foundation workflows. The following illustration shows the registration approval workflow.

The registration approval workflow



The approval workflow includes the following activity types:

- **Event activities.** Event activities are initiated by the SharePoint runtime when specific external input is received by the workflow runtime. Examples include **OnWorkflowActivated** and **OnRegistrationChanged**. Event activities can respond to external user actions such as user input. SharePoint has a predefined set of events that it will raise as event activities.
- **Workflow-initiated activities.** Some activities are initiated by the workflow engine whenever they are enabled by the control state of the workflow. Types of workflow-initiated activities in the approval workflow include **CreateTask**, **CompleteTask**, **DeleteTask**, and **ExecuteCode** activities.
- **Control-flow activities.** The Training Management workflow includes some activities that control the flow of activities. Examples include the **While** activity for looping and the **IfElseApprovalActivity** for branching.

The workflow diagram in the Visual Studio designer uses colors and special icons to visually distinguish these types of activities. The following icons are used:

- **Green oblongs.** These are for events.
- **Blue oblongs.** These are for tasks that include SharePoint functionality.
- **Grey oblongs.** These are for tasks that include only custom code.

As illustrated in the workflow diagram, the approval workflow proceeds as follows:

1. **onWorkflowActivated** is the initial event. The workflow is instantiated by SharePoint when a user registers for a course. This is also when the initial event is raised. The Training Management application does not have a programmer-provided handler for this activity.
2. **createManagerApprovalTask** invokes the code that creates an approval task and configures the fields. The

approval task is added to the RegistrationApprovalTasks list and appears in the dashboard of the manager who authorizes the class registration. This code also performs housekeeping functions such as creating the unique identifier that identifies this task.

3. **whileActivity** is a looping activity. The **while** loop accepts **onRegistrationChanged** events until a programmer-provided termination condition is met. The **onRegistrationChanged** event is the event that is raised by SharePoint whenever a registration list item changes. If the manager approves or rejects a registration request, the handler for this event changes the data state of the workflow to terminate the loop and to enable the subsequent tasks to run.
4. **completeManagerApprovalTask** does not have application code associated with it. Instead, because of the type of activity (**CompleteTask**), SharePoint automatically changes the status of the list item task to **Completed**.
5. **deleteManagerApprovalTask** does not have application code associated with it. It uses built-in SharePoint functionality for housekeeping.
6. **ifElseApprovalActivity** is a conditional branch point in the workflow. If the manager approves the registration request, the left branch is taken to create accounting entries. If the manager rejects the request, no accounting activities occur. A programmer-provided method that is referenced as a property of this activity tests the condition.
7. **codeChargeAccountingActivity** causes application-provided code to execute to generate accounting entries for the class registration.

For step-by-step instructions about how to create a workflow, download the [Visual Studio 2008 extensions for Windows SharePoint Services 3.0, v. 1.3 – Mar 2009 CTP](#) from Microsoft.com. For general information about SharePoint workflows, see [SharePoint Workflow Solutions](#) on MSDN.

The Correlation Token Property

A correlation token is a unique identifier that enables mapping between the objects in a workflow and the environment that hosts the Windows Workflow Foundation workflow runtime. It is similar to an object ID. The correlation token property, named **CorrelationToken**, specifies the type of correlation token. The Training Management application's Registrations list has only one workflow associated with it. Whenever an item is added to the Registrations list, a new task is created with one associated workflow instance. This means that the application requires two correlation tokens. One token is for the workflow instance and is named **workflowToken**. The other token is for the SharePoint task and is named **taskToken**. Each activity in the workflow has a property that specifies its owner and correlation token. Task-related activities such as **CreateTask** and **CompleteTask** use the task token for correlation. For more information, see [Correlation Tokens in Workflows](#) on MSDN.

Code-Related Properties

Code-related properties of the workflow activities associate these activities with the code in the Training Management application. You can view these properties by selecting an activity in the workflow designer. These properties are the following:

- **Invoked**. Event activities use the **Invoked** property to specify the code that should run when the event occurs. For example, an event can occur when an employee registers for a course.
- **MethodInvoking**. SharePoint workflow-initiated activities use this property to specify the method that contains the application logic associated with this task.
- **ExecuteCode**. Similar to the **MethodInvoking** property, the **ExecuteCode** property specifies a method to be executed. This property is not used by activity types provided by SharePoint; it is found on the **Code** activity provided by Windows Workflow Foundation.
- **Condition**. This property is used with **While** and **IfThen** activities. It designates the method that controls the branching logic. For example, it designates when a **while** loop should end.

Executing the Registration Approval Workflow

The Workflow.cs file, in the Contoso.TrainingManagement.Workflows project, controls the execution of the registration approval workflow. It is the code-beside file for the approval workflow created in the designer. It contains the code that is invoked by the workflow activities. The following section discusses the fields and methods included in the **Workflow** class.

The Class Fields

The Training Management application defines four fields for the **Workflow** class. At run time, the workflow engine persists the values of these fields so that the state of the workflow can be restored when it is needed.

The fields are the following:

- **workflowProperties**. This is an instance of **SPWorkflowActivationProperties** class. It represents the properties of the workflow instance when it is instantiated. The run-time context provided by SharePoint gives this object access to the list item that is associated with this workflow instance.
- **taskId**. This field is the unique identifier that is required by each SharePoint task.
- **taskProperties**. This field is an instance of the **SPWorkflowTaskProperties** class. It contains properties that record the current state of the registration approval task, such as the person to whom the task is assigned, the task title, and description. These properties are visible from the SharePoint task properties page.
- **managerApprovalTaskComplete**. This field is a Boolean value that indicates whether the manager has acted on the request. The default setting is **false**.

For information about the **SPWorkflowActivationProperties** class, see [SPWorkflowActivationProperties Class \(Microsoft.SharePoint.Workflow\)](#) on MSDN. For information about the **SPWorkflowTaskProperties** class, see [SPWorkflowTaskProperties Class \(Microsoft.SharePoint.Workflow\)](#) on MSDN.

Event Handler and Condition Methods

The approval workflow has five entry points that are called by the workflow engine at various stages of the workflow's progression. These methods are referred to by properties of the workflow activities described in "Code-Related Properties" in [Designing the Registration Approval Workflow](#). The methods are the following:

- **CreateManagerApprovalTask_MethodInvoking**. This method populates the task properties object that map this workflow to the registration that is being processed.
- **OnRegistrationChanged_Invoked**. This event handler is executed when a registration list item changes. This method then determines whether the manager has completed the registration request and sets this value into the **managerApprovalTaskComplete** field.
- **ManagerApprovalTaskCompleteCondition**. This method indicates when the request has been acted on by the manager.
- **ManagerApprovedCondition**. This method indicates whether the completed task resulted in an approval or a rejection.
- **CodeChargeAccountingActivity_ExecuteCode**. This method creates a new transaction that charges the cost center associated with this user. It occurs if the registration has been approved.

Note:

The methods invoked by the workflow engine have administrative privileges.

The CreateManagerApprovalTask_MethodInvoking Method

This method is referenced as the **MethodInvoking** property of the **createManagerApprovalTask** activity in the workflow. The method is invoked when the workflow is activated. The method runs before SharePoint creates the approval task. This allows the method to initialize the fields that define the task.

This initialization is shown in the following code.

```
C#
private void CreateManagerApprovalTask_MethodInvoking(object sender, EventArgs e)
{
    // Assign a new Guid to the taskId.
    taskId = Guid.NewGuid();

    Controller controller = new Controller();
    controller.PopulateManagerApprovalTaskProperties(taskProperties,
        workflowProperties.Web, workflowProperties.Item);
}
```

This code initializes the **taskId** field with a new GUID. This is the correlation token. For more information, see [The Correlation Token Property](#).

The **workflowProperties** object is an instance of the **SPWorkflowActivationProperties** class. It provides a data access path to the SharePoint Web and to the current registration list item.

The **CreateManagerApprovalTask_MethodInvoking** method then invokes a controller class to initialize the task properties. This is shown in the following code from the **Controller** class located in Contoso.TrainingManagement.Workflows.RegistrationApproval.Controller.

C#

```
public void PopulateManagerApprovalTaskProperties(
    SPWorkflowTaskProperties taskProperties, SPWeb web, SPListItem workflowItem)
{
    IHRManager hrManager = serviceLocator.Get<IHRManager>();
    IRegistrationRepository registrationRepository =
        serviceLocator.Get<IRegistrationRepository>();

    ITrainingCourseRepository trainingCourseRepository =
        serviceLocator.Get<ITrainingCourseRepository>();

    Registration registration = registrationRepository.Get(workflowItem.ID, web);

    TrainingCourse course = trainingCourseRepository.Get(registration.CourseId, web);

    SPUser user = GetSPUser(web, registration.UserId);
    SPUser manager = GetSPUser(web, this.hr.GetManager(user.LoginName));

    taskProperties.AssignedTo = manager.LoginName;
    taskProperties.Title = String.Format("Approve {0} registration request from
                                         {1}.", course.Title, user.Name);
}
```

The **PopulateManagerApprovalTaskProperties** method calculates some temporary values, such as **user**, **course**, and **manager** that set the task properties. The **workflowItem** value is an **SPListItem** object, and **user** and **manager** are **SPUser** objects. For more information about the **SPListItem** class, see [SPListItem](#) on MSDN. For more information about the **SPUser** class, see [SPUser \(Microsoft.SharePoint\)](#) on MSDN.

This code updates the **taskProperties** object's fields. They are the following:

- **AssignedTo.** This is the manager's logon name.
- **Title.** This is a string that includes the course title and the user name.

The Training Management application implements the **IHRManager** service as a stub that uses methods that return hard-coded sample values. This code is located in the **HRManager.cs** file of the **Contoso.HRManagement** project.

The following is the interface.

C#

```
public interface IHRManager
{
    string GetManager(string username);
    string GetCostCenter(string username);
    List<string> GetCostCenters();
    List<string> GetDirectReports(string username);
}
```

The **GetManager** method is a stub that always returns a string in the form **@ComputerName\spgmanager** where **ComputerName** is the name of the server running the Training Management application.

The **GetCostCenter** method returns the string "DEP100".

The **GetCostCenters** method returns a list that contains three elements: "DEP100", "DEP200", and "DEP300".

The **GetDirectReports** method returns a list that contains either one element or no elements. If the user name provided as the argument is in the form **"@ComputerName\spgmanager"**, the method returns a list that contains **"@ComputerName\spgemployee"**. Otherwise, the list is empty.

The OnRegistrationChanged_Invoked Method

The following code shows the **onRegistrationChanged_Invoked** method.

C#

```
private void OnRegistrationChanged_Invoked(object sender,
                                           ExternalDataEventArgs e)
{
    Controller controller = new Controller();
    managerApprovalTaskComplete =
        controller.IsManagerApprovalTaskComplete(workflowProperties.Item);
}
```

This method sets the **managerApprovalTaskComplete** field to **true** if the registration status is set to either **Approved** or **Rejected**. The **IsManagerApprovalTaskComplete** method performs this test. This is shown in the following code.

C#

```
public bool IsManagerApprovalTaskComplete(SPListItem workflowItem)
{
    bool isComplete = false;
```

```

        string status = workflowItem[Fields.RegistrationStatus].ToString();

        switch ( status )
        {
            case approvedString:
                isComplete = true;
                break;
            case rejectedString:
                isComplete = true;
                break;
            default:
                isComplete = false;
                break;
        }
        return isComplete;
    }
}

```

The ManagerApprovalTaskCompleteCondition Method

The following code shows the **ManagerApprovalTaskCompleteCondition** method.

```

C#
private void ManagerApprovalTaskCompleteCondition(object sender,
                                                ConditionalEventArgs e)
{
    e.Result = !managerApprovalTaskComplete;
}

```

This method checks the status of the **managerApprovalTaskComplete** field. If it is **true**, the workflow exits the **while** loop and continues to the next step, **completeManagerApprovalTask**.

The ManagerApprovedCondition Method

The following code shows the **ManagerApprovedCondition** method.

```

C#
private void ManagerApprovedCondition(object sender, ConditionalEventArgs e)
{
    Controller controller = new Controller();
    e.Result = controller.IsManagerApprovalTaskApproved(workflowProperties.Item);
}

```

This method determines whether the status field is set to **Approved**. The result determines which branch of the **ifElseApprovalActivity** the workflow takes. If the result is **true**, the workflow continues to the **codeChargeAccountingActivity** coding activity. If the result is **false**, the workflow ends.

The CodeChargeAccountingActivity_ExecuteCode Method

The following code shows the **CodeChargeAccountingActivity_ExecuteCode** method.

```

C#
private void CodeChargeAccountingActivity_ExecuteCode(object sender, EventArgs e)
{
    Controller controller = new Controller();
    controller.ChargeAccounting(workflowProperties.Web, workflowProperties.Item);
}

```

The following **ChargeAccounting** method uses the **SPWorkflowActivationProperties** object in the **workflowProperties** field to retrieve the registration, the course, and the user. It uses this data to create an accounting transaction. This is an example of how a workflow activity can access the information in the SharePoint data store.

```

C#
public void ChargeAccounting(SPWeb web, SPListItem workflowItem)
{
    IHRManager hrManager = serviceLocator.Get<IHRManager>();
    IAccountingManager accountingManager = serviceLocator.Get<IAccountingManager>();
    IRegistrationRepository registrationRepository =
serviceLocator.Get<IRegistrationRepository>();
    ITrainingCourseRepository trainingCourseRepository =
serviceLocator.Get<ITrainingCourseRepository>();

    // Get the registration and training course related to this task.
    Registration registration = registrationRepository.Get(workflowItem.ID, web);
    TrainingCourse trainingCourse = trainingCourseRepository.Get(registration.CourseId, web);

    // Get the user related to this registration.
    SPUser user = GetSPUser(web, registration.UserId);

    // Construct the transaction related to this approved registration.
    Transaction tran = new Transaction();
}

```

```

        tran.Amount = trainingCourse.Cost;
        tran.CostCenter = hrManager.GetCostCenter(user.LoginName);
        tran.Bucket = trainingBucketString;
        tran.Description = String.Format("{0} training course registration by {1}.",
trainingCourse.Title, user.Name);

        accountingManager.SaveTransaction(tran);
    }

```

The **Transaction** class that is defined in the Contoso.AccountingManagement project has properties that describe an accounting transaction. After these are provided, the application invokes the **SaveTransaction** method of the **AccountingManager** class. The Contoso.AccountingManagement project implements a small database that uses the **System.Data.DataSet** class provided by the .NET Framework. The **DataSet** class is an in-memory database cache. For more information, see [DataSet Class](#) on MSDN.

The Contoso.AccountingManagement project has an interface and service architecture that is typical for Web applications such as Training Management. However, because the accounting service is only an example, the TrainingManagement application uses the same hardcoded initial data values for the data set whenever the server application is loaded. Updates to the data set, including transactions, do not persist between application restarts.

The following code shows the methods provided by the **IAccountingManager** interface.

C#

```

public interface IAccountingManager
{
    float GetBudget(string costCenter, string bucket);
    float GetRemainingBudget(string costCenter, string bucket);
    IList<Transaction> GetTransactions(string costCenter, string bucket);
    void SaveTransaction(Transaction transaction);
}

```

If the **bucket** argument equals "Training", the **GetBudget** method returns 5000 for cost center "DEP100", 8000 for cost center "DEP200", and 8000 for cost center "DEP300". For arguments other than these, the method returns the value **float.NaN**.

For a particular cost center, the **GetRemainingBudget** method returns the budget returned by **GetBudget** less the amount of any transactions for this cost center that have been saved using the **SaveTransaction** method.

The **GetTransactions** method returns a list of transactions that have been saved (using the **SaveTransaction** method) from the time that the Training Management application was started.

The **SaveTransaction** method adds the transaction to the in-memory database.

The code is located in the AccountingManager.cs file of the Contoso.AccountingManagement project. The **ValidateTransaction** method validates the data used in the transaction.

C#

```

public void SaveTransaction(Transaction transaction)
{
    ValidateTransaction(transaction);

    AccountingDataSet.TransactionRow row =
        AccountingDataSet.Instance.Transaction.NewTransactionRow();

    row.Amount = transaction.Amount;
    row.Bucket = transaction.Bucket;
    row.CostCenter = transaction.CostCenter;
    row.Description = transaction.Description;

    AccountingDataSet.Instance.Transaction.AddTransactionRow(row);
    AccountingDataSet.Instance.AcceptChanges();
}

```

This code adds the transaction record to the database.

The Registration Approval Task Content Type

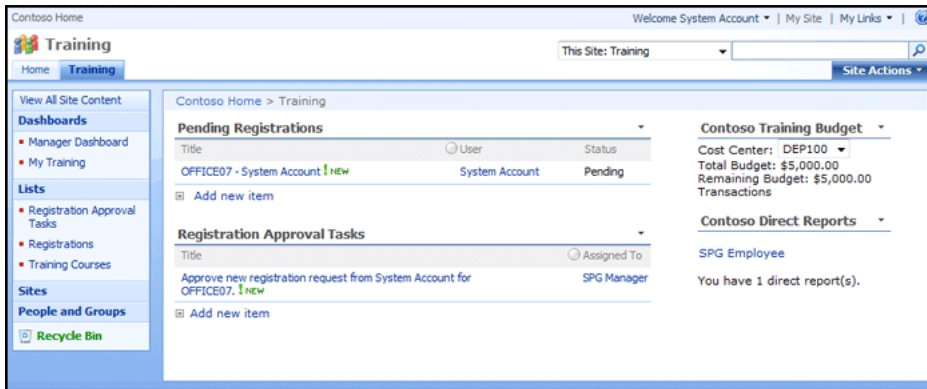
The `RegistrationApprovalTaskContentType.xml` file contains the definition of the **RegistrationApprovalTaskContentType**. It is created by the developer. The **ContentType** ID is formed from a prefix and a GUID. The prefix `0x010801` is followed by the special characters `00` and then followed by a user-generated GUID. This identifies the content type as being a subtype of the **Workflow Item** content type. For more information about content type inheritance, see [Base Content Type Hierarchy](#) on MSDN.

When you add a new item to the Registrations list, the first step of the workflow creates a new task of **RegistrationApprovalTaskContentType**. This task appears on the manager's dashboard and on the task management pages. For more information about content type IDs, see [Content Type IDs](#) on MSDN.

View the Manager Dashboard Use Case

The manager dashboard is an example of how standard SharePoint Web Parts, custom Web Parts, and user controls can be incorporated into a SharePoint application. To see the manager's view of the dashboard, log on as spgmanager. From the Quick Launch, select **Manager Dashboard**. The dashboard uses both custom Web Parts and Web Parts that are provided by SharePoint. The following illustration shows an example of the dashboard.

The manager dashboard



The four Web Parts on the page display the following:

- They display pending registrations, which is a standard SharePoint List View Web Part.
- They display registration approval tasks, which is a standard SharePoint List View Web Part.
- They display training budget information, which is a custom Web Part.
- They display direct reports information for the current user, which is a user control that is wrapped in a custom Web Part.

For guidance on deciding between standard and custom Web Parts, see [Using Standard and Custom Web Parts](#). For more information about Web Parts, see [ASP.NET Web Parts Controls](#) on MSDN.

Note:

There is an interaction between the Registration Approval Tasks Web Part and the Training Budget Web Part. When managers approve a registration, they are redirected to the manager dashboard, which renders the Training Budget Web Part. The modification of the registration triggers the workflow. The workflow runs in a separate thread. It communicates to the Accounting service and charges the cost of the training course to the manager's cost center. To see the change in the budget, refresh the manager dashboard.

Pending Registration and Registration Approval Tasks Web Parts

The Pending Registrations Web Part and the Registration Approval Tasks Web Part are SharePoint List View Web Parts. The Pending Registrations Web Part is based on a custom view of the registrations list, and the Registration Approval Tasks Web Part is based on the Registration Approval Tasks list view. Because the data for these two Web Parts is in SharePoint, you can use the standard List View Web Part combined with custom views to display the data. There is no need to develop a custom user interface.

The Pending Registrations Web Part is based on the following code that defines the custom registrations list view. The code is located in the List Definitions\RegistrationListDefinition\schema.xml file in the Contoso.TrainingManagement project.

XML

```
...
<ViewFields>
  <FieldRef Name="LinkTitle">
  </FieldRef>
  <FieldRef Name="User">
  </FieldRef>
  <FieldRef Name="RegistrationStatus">
  </FieldRef>
</ViewFields>
<Query>
  <Where>
    <Eq>
      <FieldRef Name="RegistrationStatus" />
      <Value Type="Text">Pending</Value>
    </Eq>
  </Where>
  <OrderBy>
    <FieldRef Name="Modified" Ascending="FALSE">
    </FieldRef>
  </OrderBy>
</Query>
...
```

The List View definition creates an instance of a List View Web Part on the Manage.aspx page that displays the custom registrations list view. The code for the definition is located in the Forms\ManagerDashboard\ManagerDashboardElement.xml file of the Contoso.TrainingManagement project.

XML

```
<View List="Lists/Registrations" BaseViewID="3" DisplayName="Pending Registrations"
      Name="Pending Registrations" RecurrenceRowset="TRUE" WebPartZoneID="Left"
      WebPartOrder="1">
  <![CDATA[
    <WebPart xmlns=http://schemas.microsoft.com/WebPart/v2>
      <Assembly>Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
        PublicKeyToken=71e9bce111e9429c</Assembly>

      <TypeName>Microsoft.SharePoint.WebPartPages.ListViewWebPart</TypeName>
      <Title>Pending Registrations</Title>
    </WebPart>
  ]]>
</View>
```

The code for the Registration Approval Tasks Web Part is similar to the code for the Pending Registrations Web Part.

Training Budget and Direct Reports Web Parts

The Training Budget Web Part and the Direct Reports Web Part are custom Web Parts. The Training Budget Web Part accesses the Accounting Management service for financial information for the selected cost center. This functionality requires a custom Web Part because the accounting data is not stored in SharePoint. Therefore, it requires a custom user interface to display it.

The Direct Reports Web Part is a combination of a custom user control and a custom Web Part. The user control accesses the HR Management service to display the direct reports for the current user. The custom Web Part wraps the user control and allows it to be placed inside of a Web Part zone on the Web Part page.

Including a custom Web Part on a Web Part page has the following four phases:

1. The code is compiled and deployed. This installs the Web Part in SharePoint.
2. Developers can place the Web Part on the manager dashboard in one of two ways. They can either use code or the **File** element definitions in the ManageElement.xml file.
3. At run time, the **OnInit** method executes and creates the Web Part layout.
4. When the page loads, the **OnLoad** method populates the Web Part with the data.

These steps are discussed in the next section.

Training Budget Web Part

C#

```
[Guid("1e98f121-e107-4301-854a-0db8ba88385b")]
public class TrainingBudgetWebPart :
    System.Web.UI.WebControls.WebParts.WebPart, ITrainingBudgetView
{
    /// ...
}
```

The Contoso.TrainingManagement project also contains an XML file named TrainingBudget.webpart. This is shown in the following code.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<webParts>
  <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <!--
        The following Guid is used as a reference to the web part class,
        and it will be automatically replaced with actual type name at deployment time.
      -->
      <type name="1e98f121-e107-4301-854a-0db8ba88385b" />
      <importErrorMessage>Cannot import Training Budget Web Part.</importErrorMessage>
    </metaData>
    <data>
      <properties>
        <property name="Title" type="string">Contoso Training Budget</property>
        <property name="Description" type="string">The Contoso Training Budget Web Part will
display total budget, remaining budget, and a list of transactions, grouped by cost
center.</property>
      </properties>
    </data>
  </webPart>
</webParts>
```

The **Guid** attribute of the **TrainingBudgetWebPart** class matches the GUID in the TrainingBudgetWebPart.xml file. When the project is deployed, this GUID enables Visual Studio extensions for Windows SharePoint Services to

add the correct type information in the Web Part definition. The Web Part is installed in the SharePoint Web Part gallery.

The Web Parts are placed on the manager dashboard page with **File** element definitions. The following is the XML code that locates the Training Budget Web Part and Direct Reports Web Part. It is located in Forms\ManagerDashboard\ManageElement.xml file.

XML

```
<AllUsersWebPart WebPartZoneID="Right" WebPartOrder="1">
  <![CDATA[
    <webParts>
      <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
        <metaData>
          <type name="Contoso.TrainingManagement.WebParts.TrainingBudgetWebPart,
            Contoso.TrainingManagement, Version=1.0.0.0, Culture=neutral,
            PublicKeyToken=9f4da00116c38ec5" />
          <importErrorMessage>Cannot import Training Budget Web
            Part.</importErrorMessage>
        </metaData>
        <data>
          <properties>
            <property name="Title" type="string">Training Budget</property>
            <property name="Description" type="string">The Training Budget Web Part
              will display total budget, remaining budget, and a list of transactions, grouped by cost
              center.</property>
          </properties>
        </data>
      </webPart>
    </webParts>
  ]]>
</AllUsersWebPart>
<AllUsersWebPart WebPartZoneID="Right" WebPartOrder="2">
  <![CDATA[
    <webParts>
      <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
        <metaData>
          <type name="Contoso.TrainingManagement.WebParts.DirectReportsWebPart,
            Contoso.TrainingManagement, Version=1.0.0.0, Culture=neutral,
            PublicKeyToken=9f4da00116c38ec5" />
          <importErrorMessage>Cannot import Direct Reports Web
            Part.</importErrorMessage>
        </metaData>
        <data>
          <properties>
            <property name="Title" type="string">Direct Reports</property>
            <property name="Description" type="string">The Direct Reports Web Part
              displays all direct reports for the current user.</property>
          </properties>
        </data>
      </webPart>
    </webParts>
  ]]>
</AllUsersWebPart>
```

The **type** information is included in this file because, unlike the Webpart.xml file, it is not maintained by Visual Studio extensions for Windows SharePoint Services.

The Web Parts are installed as part of the solution using modules in the TrainingBudgetWebPart.xml and DirectReportsWebPart.xml files. The modules install the files to the _catalogs/wp folder, which is the location of the Web Part Gallery page. After the Web Parts are in the gallery, they are declaratively added to the page with the **AllUsersWebPart** elements. These are in the folders Forms\Manage\ManageElement and Forms\Default\DefaultElement XML.

At run time, the **CreateChildControls** method creates the user interface (UI) elements that comprise the Web Part. This is shown in the following code. The code is located in the TrainingBudgetWebPart.cs file in Contoso.TrainingManagement\WebParts\TrainingBudgetWebPart project.

C#

```
protected override void CreateChildControls()
{
    base.CreateChildControls();

    layout = new Panel();
    totalBudget = new Label();
    totalBudget.Attributes.Add("name", "TotalBudget");
    remainingBudget = new Label();
    remainingBudget.Attributes.Add("name", "RemainingBudget");
    costCenters = new DropDownList();
    costCenters.Attributes.Add("name", "CostCenters");
    costCenters.AutoPostBack = true;
    costCenters.SelectedIndexChanged += new EventHandler(costCenters_SelectedIndexChanged);
```

```

        transactions = new GridView();
        transactions.Attributes.Add("name", "Transactions");

        Label label = new Label();
        label.Text = "Cost Center: ";
        layout.Controls.Add(label);
        layout.Controls.Add(costCenters);
        layout.Controls.Add(new LiteralControl("<br />"));

        label = new Label();
        label.Text = "Total Budget: ";
        layout.Controls.Add(label);
        layout.Controls.Add(totalBudget);
        layout.Controls.Add(new LiteralControl("<br />"));

        label = new Label();
        label.Text = "Remaining Budget: ";
        layout.Controls.Add(label);
        layout.Controls.Add(remainingBudget);
        layout.Controls.Add(new LiteralControl("<br />"));

        label = new Label();
        label.Text = "Transactions";
        layout.Controls.Add(label);
        layout.Controls.Add(new LiteralControl("<br />"));

        layout.Controls.Add(transactions);

        this.Controls.Add(layout);
    }

```

This code demonstrates how to programmatically lay out a Web Part. There is no Visual Studio designer support for Web Parts.

The **OnLoad** method executes when the page loads. The code is located in the TrainingBudgetWebPart.cs file in the Contoso.TrainingManagement\WebParts\TrainingBudgetWebPart folder.

```

C#
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);

    if ( !Page.IsPostBack )
    {
        presenter.SetCostCenterSource();
        costCenters.DataBind();
        presenter.SetTransactionSource();
        transactions.DataBind();
    }
}

```

The **OnLoad** method populates the Web Part's data controls. It uses a presenter class named **TrainingBudgetPresenter** to do this. The following code shows the **TrainingBudgetPresenter** class. This code is found in the WebParts\TrainingBudgetWebPart\TrainingBudgetPresenter.cs file of the Contoso.TrainingManagement project.

```

C#
public TrainingBudgetPresenter(ITrainingBudgetView view)
{
    ServiceLocator serviceLocator = ServiceLocator.GetInstance();

    this.view = view;
    this.acct = serviceLocator.Get<IAccountingManager>();
    this.hr = serviceLocator.Get<IHRManager>();
}

public void SetCostCenterSource()
{
    List<string> costCenters = hr.GetCostCenters();
    view.CostCenters = costCenters;
}

public void SetTransactionSource()
{
    string costCenter = view.CostCenter;

    view.TotalBudget = acct.GetBudget(costCenter, "Training");
    view.RemainingBudget = acct.GetRemainingBudget(costCenter, "Training");
    view.Transactions = acct.GetTransactions(costCenter, "Training");
}

```

This code shows how the Accounting Management service is invoked to get the data required by the Training Budget Web Part.

Direct Reports Web Part

The Direct Reports Web Part is similar to the Training Budget Web Part. The difference is that instead of programmatically creating the UI elements, it embeds a user control. The **CreateChildControls** method performs this function. It is shown in the following code. This code is found in the WebParts\DirectReportsWebPart\DirectReportsWebPart.cs file of the Contoso.TrainingManagement project.

```
C#
protected override void CreateChildControls()
{
    base.CreateChildControls();

    try
    {
        DirectReports directReports =
            (DirectReports) Page.LoadControl(
                "~/_controltemplates/TrainingManagement/DirectReports.ascx");
        directReports.Web = SPContext.Current.Web;
        directReports.LoginName = SPContext.Current.Web.CurrentUser.LoginName;
        directReports.ShowLogin = this.showLogin;
        this.Controls.Add(directReports);
    }
    catch ( HttpException ex )
    {
        this.Controls.Add(new LiteralControl("<br />An unexpected error occurred loading Web
Part. " + ex.Message));
    }
}
```

The advantage of a user control is that it allows you to use the Visual Studio design surface. This option is not available for Web Parts. To learn how to wrap a user control in a Web Part, see [How to: Wrap a User Control Inside of a Web Part for SharePoint](#).

The Direct Reports Web Part contains a custom (programmer-provided) property named **Show Login Name**. The following code example, from the Web Parts\DirectReportsWebPart\DirectReportsWebPart.cs file in the Contoso.TrainingManagement project, exposes a user control property through a Web Part property.

```
C#
private bool showLogin = false;

[Personalizable(PersonalizationScope.Shared)]
[WebBrowsable(true)]
[WebDisplayName("Show Login Name")]
[WebDescription("Show login name for direct reports.")]
[SPWebCategoryName("Development")]
public bool ShowLogin
{
    get
    {
        return showLogin;
    }
    set
    {
        this.showLogin = value;
    }
}
```

The property definition uses custom attributes provided by SharePoint to allow the property to be recognized by SharePoint at run time. The property value is used within the **CreateChildControls** method to modify the embedded user control.

Employee View of Manager Dashboard

When employees log on to the Training Management application, they can see the manager dashboard. However, employees have insufficient privileges to view the registration approval tasks. The following illustration shows an example of the employee's view of the manager dashboard.

Employee view of management dashboard

Contoso Home

Training

Home

Training

Welcome SPS Employee

My Site

My Links

This Site: Training

View All Site Content

Dashboards

Manager Dashboard

My Training

Lists

Registrations

Training Courses

Sites

People and Groups

Contoso Home > Training

Pending Registrations

Title	User	Status
OFFICE07 - System Account	System Account	Pending

Add new item

Registration Approval Tasks - (Series Items)

Access denied. You do not have permission to perform this action or access this resource.

Contoso Training Budget

Cost Center: DEP100

Total Budget: \$5,000.00

Remaining Budget: \$5,000.00

Transactions

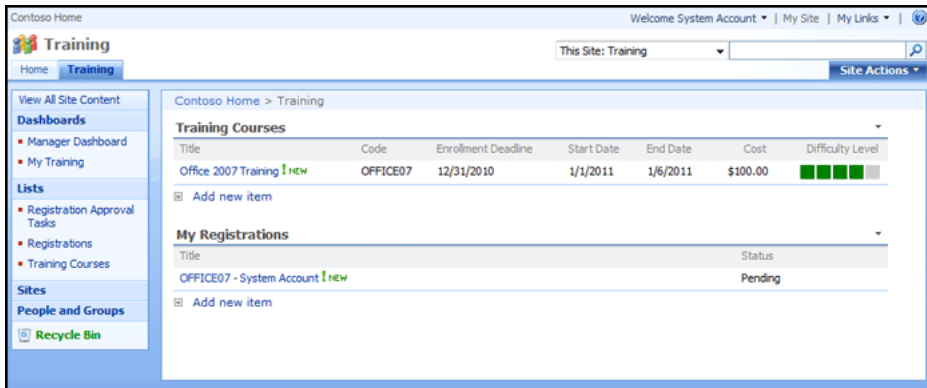
Contoso Direct Reports

No direct reports were found.

View the Training Dashboard Use Case

When employees log on to the Training Management application, they can see the training dashboard by clicking **My Training** in the Quick Launch. The dashboard displays the list of available training courses and the list of courses that the current employee has registered for. The following illustration shows an example of the training dashboard.

Training dashboard



Training Courses and My Registrations Web Parts

The Training Courses Web Part and the My Registrations Web Part are SharePoint List View Web Parts. The Training Courses Web Part is a default list view. The My Registrations Web Part is based on a custom view of the registrations list. Because the data for these two Web Parts is in SharePoint, the standard List View Web Part, combined with a custom view for the My Registrations Web Part, is an ideal way to display it. There is no need to develop a custom user interface.

The My Registrations Web Part is based on the following code that defines the custom registrations list view. The code is located in the ListDefinitions\RegistrationListDefinition\schema.xml file in the Contoso.TrainingManagement project.

XML

```
...
<ViewFields>
  <FieldRef Name="LinkTitle">
  </FieldRef>
  <FieldRef Name="RegistrationStatus">
  </FieldRef>
</ViewFields>
<Query>
  <Where>
    <Eq>
      <FieldRef Name="User" />
      <Value Type="Integer">
        <UserID />
      </Value>
    </Eq>
  </Where>
  <OrderBy>
    <FieldRef Name="RegistrationStatus" />
  </OrderBy>
</Query>
...
```

The List View definition creates an instance of a List View Web Part on the Default.aspx page that displays the custom registrations list view. The code for the definition is located in the Forms\TrainingDashboard\TrainingDashboardElement.xml file of the Contoso.TrainingManagement project.

XML

```
<View List="Lists/Registrations" BaseViewID="2" RecurrenceRowset="TRUE"
  WebPartZoneID="Left" WebPartOrder="4">
  <![CDATA[
    <WebPart xmlns=http://schemas.microsoft.com/WebPart/v2
      xmlns:iwp="http://schemas.microsoft.com/WebPart/v2/Image">
      <Assembly>Microsoft.SharePoint, Version=12.0.0.0,
        Culture=neutral, PublicKeyToken=71e9bce11e9429c</Assembly>
      <TypeName>Microsoft.SharePoint.WebPartPages.ListViewWebPart</TypeName>
      <Title>My Registrations</Title>
    </WebPart>
  ]]>
</View>
```


Training Management Application Upgrade

This topic discusses how to upgrade different SharePoint components in the Training Management application, such as content types and event receivers. It also explains how to apply the Contoso theme to the SharePoint site. The Contoso theme is an example of site branding. The topic includes the following:

- [Installing the Upgrades](#). This is a procedure for installing the upgrades.
- [Applying the Contoso Theme](#). This is a procedure for applying the Contoso theme.
- [Upgrading the Training Management Application](#). This topic discusses how the SharePoint components in the Training Management application are upgraded.

Installing the Upgrades

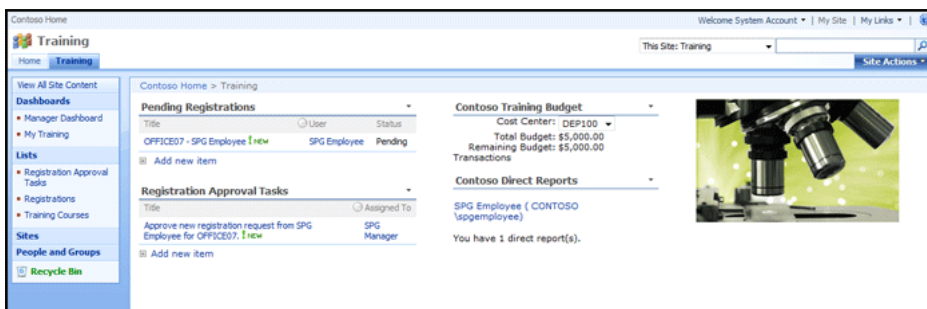
The following procedure demonstrates how to install the Training Management application upgrades. You must install the application before you can install the upgrades. For installation instructions, see [Installing the Training Management Application](#). For more information about installing or uninstalling features, see [Installing or Uninstalling Features](#) on MSDN.

To install the upgrades

1. In SharePoint, deactivate the Training Management Web feature on the Training Management site.
2. From a command prompt, navigate to the <RI install dir>\Contoso.TrainingManagement.RI\Contoso.TrainingManagement\bin\debug directory, and then run the following command to retract and remove Contoso.TrainingManagement:
setup.bat /u
3. Run IISreset.
4. In Visual Studio 2008, open Contoso.TrainingManagement.RI.Upgrade.sln.
5. In Visual Studio, right-click the **Contoso.TrainingManagement** project, and then click **Deploy**. This deploys the SharePoint solution to the localhost URL.
6. Run IISreset.
7. In SharePoint, activate the Contoso Training Management Web feature on the training site.
8. In Visual Studio 2008, right-click the **Contoso.TrainingManagement.Workflows.RegistrationApproval** project, and then click **Properties**. Click the **Debug** tab. Enter the URL of the SharePoint site where the Training Management application is located, for example, <http://localhost/training>.
9. In Visual Studio 2008, right-click the **Contoso.TrainingManagement.Workflows.RegistrationApproval** project, and then click **Deploy**. This builds the project and deploys the SharePoint solution to the URL of the SharePoint site where the Training Management application is located.
10. Manually set Contoso.TrainingManagement.Workflows.RegistrationApproval v1 to **No New Instances**. To disable any new workflow instances, go to the list settings for the Registrations list and browse to the Workflow settings. Click **Remove a workflow**. Find Registration Approval v1, and set it to **No New Instances**.

The following illustration shows an example of one of the upgrade changes the application. There is now an image on several of the Web pages.

View of manager dashboard with image



For more information about updating the Training Management application, see [Upgrading the Training Management Application](#). For information about adding SharePoint features to an application, see [Stapling Features to Site Definitions](#). For general guidance about deploying and upgrading SharePoint applications, see [Deployment Scenarios](#).

Applying the Contoso Theme

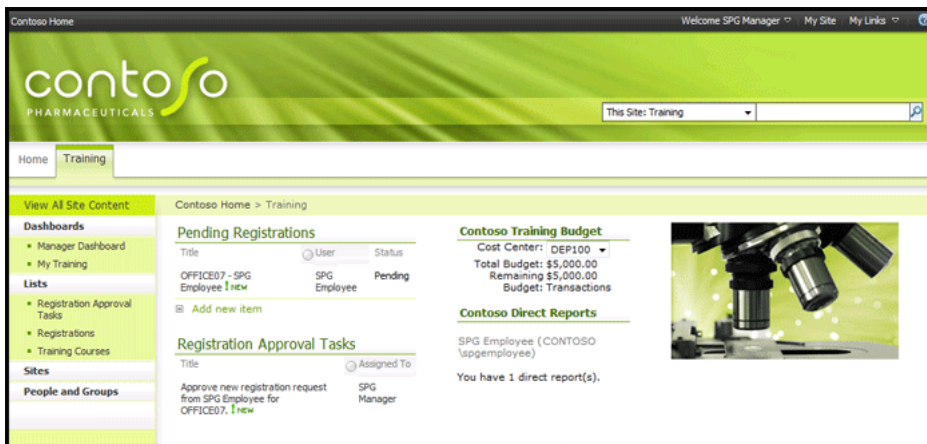
The following procedure demonstrates how to apply the Contoso theme to the Training Management SharePoint site. You must install the application before you can apply the theme. You do not have to apply the upgrades. For installation instructions, see [Installing the Training Management Application](#). For more information about installing or uninstalling SharePoint Features, see [Installing or Uninstalling Features](#) on MSDN.

To apply the Contoso theme

1. In Visual Studio, open the Contoso.RI.UpdateTheme.sln file.
2. Right-click the **Contoso.RI.UpdateTheme** project, and then click **Properties**. In the **Debug** section, specify the URL of the SharePoint site where the Training Management application is located.
3. Right-click the **Contoso.Ri.UpdateTheme** project, and then click **Deploy**.

The following illustration shows an example of how the site appears after you apply the Contoso theme.

View of manager dashboard with theme and feature



For more information about updating the Training Management application, see [Upgrading the Training Management Application](#). For information about adding SharePoint features to an application, see [Stapling Features to Site Definitions](#). For general guidance about deploying and upgrading SharePoint applications, see [Deployment Scenarios](#).

Upgrading the Training Management Application

The Contoso.RI.Upgrade solution demonstrates how various site elements can be upgraded with new functionality and bug fixes.

The following are the elements that the upgrade changes and the elements that remain unchanged:

- **Site columns.** The upgrade adds a new site column named **Instructor**.
- **Content types.** The upgrade adds a new field named **Instructor** to the **TrainingCourse** content type.
- **Web Parts.** The upgrade adds a Web Part property named **ShowTransactionGridLines** to the **TrainingBudgetWebPart**. It removes a **ShowLogin** Web Part from the **DirectReportsWebPart**.
- **Files and modules.** The upgrade adds an image to the `trainingdashboard.aspx` and `managerdashboard.aspx` pages.
- **Application pages.** The upgrade adds an image to the `registrationapproval.aspx` page.
- **User controls.** The upgrade modifies the `DirectReports.ascx` user control to add an **AlternatingItemStyle** property to the **DataList** control.
- **List item event handlers.** Although the upgrade does not change any list item event handlers, it does include some special handling that is required to account for the new version number.
- **Workflows.** The upgrade modifies the workflow to contain a new custom code activity in the **ifElseApprovalActivity** to log to the workflow history list.
- **Custom actions.** The upgrade makes no changes to custom actions.
- **List templates.** The upgrade makes no changes to list templates.
- **List definitions.** The upgrade makes no changes to list definitions.
- **List instances.** The upgrade makes no changes to list instances.

For guidelines and considerations for upgrading the different elements in a SharePoint applications, see [Upgrading an Application](#).

Note:

When upgrading an application, it is a common practice to increase the version number of the assemblies that make up the application. In the case of the Training Management application, the only assembly that was versioned was `Contoso.TrainingManagement`. Even though the **Contoso.TrainingManagement.Common**, **Contoso.TrainingManagement.Repository**, and **Contoso.TrainingManagement.ServiceLocator** assemblies are part of the SharePoint solution, the upgrade does not affect these assemblies enough to require a new assembly version. Cases where upgrades cause significant changes and the assemblies require a new version require careful planning. For example, in the Training Management application, it is important to ensure that the registration approval workflow continues to function. This is because of the shared nature of those assemblies. SharePoint removes all assemblies in a SharePoint solution when the solution is retracted. This becomes an issue because the registration approval workflow also depends on the **Contoso.TrainingManagement.Common**, **Contoso.TrainingManagement.Repository**, and **Contoso.TrainingManagement.ServiceLocator** assemblies.

Installing the Upgrade Solution

To install the upgrade SharePoint solution, follow the procedure in [Installing the Upgrades](#). The upgrade installation replaces the originally deployed files with new versions. An alternative approach to the procedure is to use the STSADM command line tool. If you use this approach, you must first run the **stsadm -o retractsolution** command to retract the original solution before you install the upgrade solution.

Note:

Another installation approach is to use the **stsadm -o upgradesolution** operation. This operation removes the files that are associated with the earlier version of the SharePoint solution and adds the files that are contained in the new version of the solution. This option also backs up the earlier version of the SharePoint solution and stores it in the configuration database. This operation does not automatically install any new SharePoint features that are included in the solution or run any feature deactivation or activation code. You must manually install any new SharePoint features before they can be activated. Use the **stsadm -o installfeature** command to perform this operation. If you must deactivate or activate any SharePoint features, you must either use the **stsadm -o deactivatefeature** and **stsadm -o activatefeature** commands or use the SharePoint user interface.

QuickStarts

The QuickStarts are Visual Studio solutions that demonstrate general techniques for developing SharePoint applications. This guidance includes the following QuickStarts:

- [Debugging an SPItemEventReceiver](#). This QuickStart demonstrates how to debug an **SPItemEventReceiver** that was created with Visual Studio extensions for Windows SharePoint Services.
- [Accessing SharePoint List Items](#). This QuickStart demonstrates three patterns for programmatically accessing SharePoint list items.

Debugging an SPItemEventReceiver

This QuickStart demonstrates how to debug an **SPItemEventReceiver** that was created with Visual Studio extensions for Windows SharePoint Services version 1.3. The SharePoint solution contains a site definition that defines a sample content type, a list definition, and a list instance with an item event receiver. The **SPGDebuggingQuickStart** list contains two custom fields named **MyDateTimeField1** and **MyDateTimeField2**. The **MyDateTimeField2** field is hidden. It is populated by adding one day to the **MyDateTimeField1** field.

Building and Running the QuickStart

The QuickStart ships as source code. This means you must first compile it before you can run it.

To build and run the Debugging an SPItemEventReceiver QuickStart

1. In Visual Studio, open the SPGDebuggingQuickStart.sln solution file. For information about how to manually debug a SharePoint application and how to attach to the w3wp process, see [How to: Debug SharePoint Applications](#).
2. On the **Build** menu, click **Rebuild Solution**.
3. Right-click **SPGDebuggingQuickStart**, and then click **Deploy**

Note:

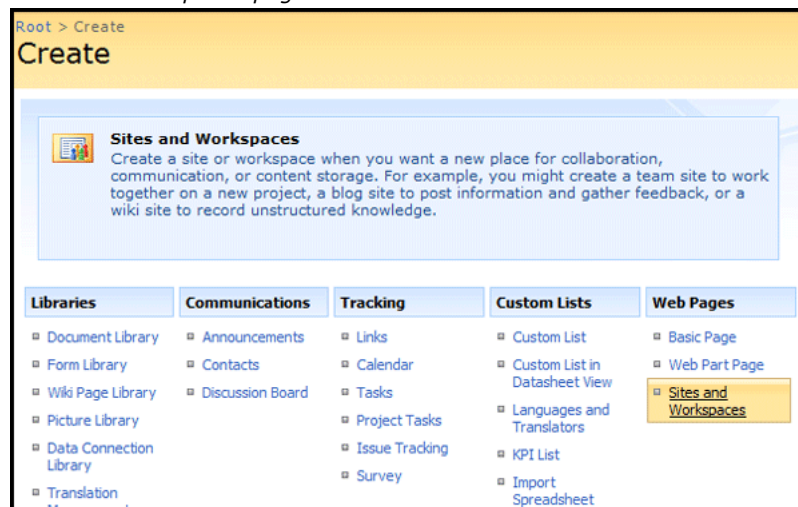
The deployment process may fail if the Visual Studio extensions for Windows SharePoint Services application pool identity does not have permissions to deploy to http://localhost. To correct this, add the Visual Studio extensions for Windows SharePoint Services application pool identity as a full control user on the http://localhost site collection. After you do this, repeat step 3.

The next procedure demonstrates how to use the QuickStart.

To debug an SPItemEventReceiver

1. When you start the QuickStart, you should see the SharePoint site. If the browser does not automatically open, browse to http://localhost.
2. Click the **Site Actions** tab, and then click **Create** (or click **Create Site** if the localhost is a Team Site). The Create page appears. In the Web Pages column, click the **Sites and Workspaces** link. The following illustration shows the Create page.

Sites and workspaces page



3. In the **Title** box, type **Debugging QuickStart**. In the **URL name** box, type **spgdebuggingquickstart**. Under **Select a template**, click the **Development** tab, click **SPGDebuggingQuickStart**, and then click **Create**. The following illustration shows the site page.

SPGDebugging configuration

- After the site is created, click the **Debugging QuickStart** tab. In the Quick Launch, click **SPGDebuggingQuickStartList**, and then click **New** on the toolbar. Provide values for the **Title** and **MyDateTimeField 1** boxes. Click **OK**. Notice that the **MyDateTimeField 2** field is not shown because it is unpopulated. The following illustration shows the list form.

New list item form

Debugging Techniques

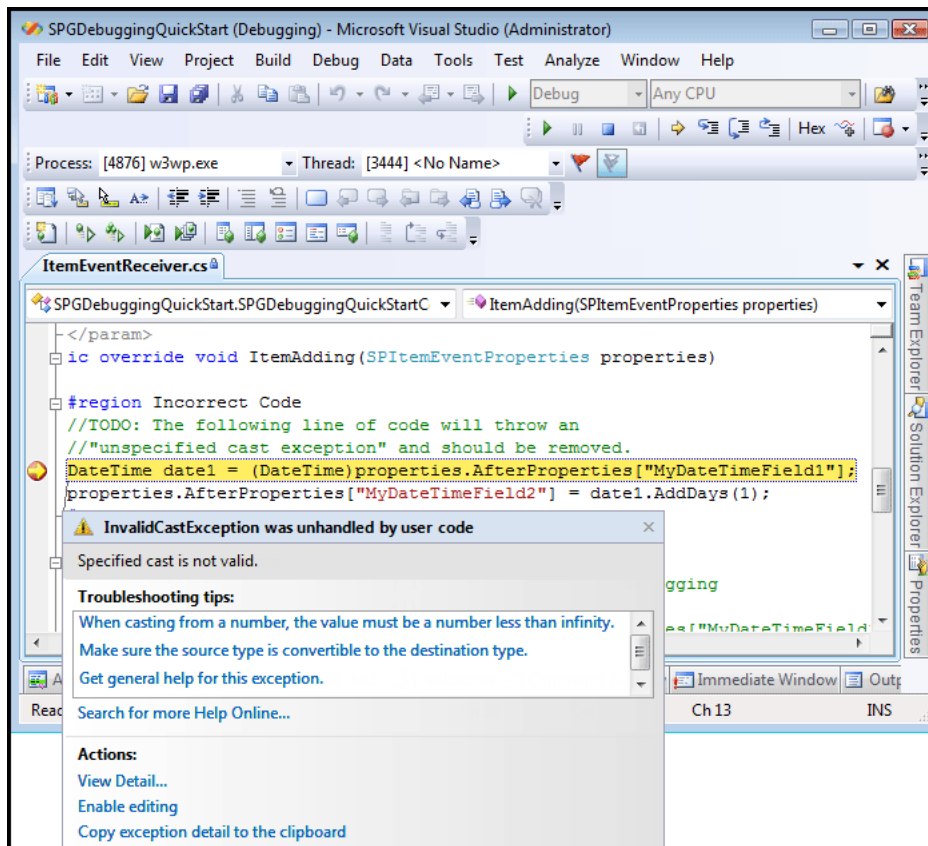
The following code demonstrates the incorrect way to access and set a **DateTime** field from the **AfterProperties** collection.

```
C#
public override void ItemAdding(SPItemEventProperties properties)
{
    DateTime date1 = (DateTime)properties.AfterProperties["MyDateTimeField1"];
    properties.AfterProperties["MyDateTimeField2"] = date1.AddDays(1);
}
```

To debug the list

- In Visual Studio, place a breakpoint in the **ItemAdding** method, click **Attach to process** on the **Tools** menu.
- Attach to all the W3wp.exe processes.
- Navigate to the SPGDebuggingQuickStartList through the link on the QuickStart, and then add a new item to the list. The following illustration shows the exception.

Debugging information



The following code demonstrates one way to correct the exception.

```

C#
public override void ItemAdding(SPItemEventProperties properties)
{
    DateTime date1 = new DateTime();
    if (DateTime.TryParse((string)properties.AfterProperties["MyDateTimeField1"], out date1))
    {
        properties.AfterProperties["MyDateTimeField2"] =
date1.ToUniversalTime().AddDays(1).ToString("yyyy-MM-dd");
    }
    else
    {
        properties.Cancel = true;
        properties.ErrorMessage = "An error occurred.";
    }
}

```

For more information about debugging, see [How to: Debug SharePoint Applications](#).

Accessing SharePoint List Items

This QuickStart demonstrates the following three patterns for programmatically accessing SharePoint list items:

- **The Direct SharePoint List Access pattern.** This pattern uses the SharePoint object model directly, with no intervening layers of abstraction.
- **The List Item Repository Access pattern.** This pattern abstracts calls to SharePoint lists. It uses a programmer-provided repository class.
- **The Entity Repository Access pattern.** This pattern provides two abstraction layers. It uses business entities with entity repositories and a **ListItemRepository** class.

For more information about the repository pattern, see [The Repository Pattern](#).

The SPGSharePointDataAccessQuickStart.sln file defines the QuickStart. The Visual Studio solution contains five projects. The main project is SPGSharePointDataAccessQuickStart.SharePoint. This project deploys a custom SharePoint application page that demonstrates how to access SharePoint list items. This page is deployed to `_layouts/SPG/SPGSharePointDataAccessQuickStart.aspx` on the SharePoint Web application. (The QuickStart interacts with a SharePoint Announcements list, which is located in the root site of the site collection that the QuickStart is deployed to.)

QuickStart Setup

This section describes how to set up the QuickStart. The site collection that the QuickStart is deployed to requires an Announcements list at the root site.

To set up the QuickStart

1. In Visual Studio, open SPGSharePointDataAccessQuickStart.sln.
2. Right-click the **SPGSharePointDataAccessQuickStart.SharePoint** project, and then click **Properties**.
3. On the **Debug** tab, enter the URL of your local SharePoint test instance. The default is `http://localhost`.
4. Right-click the solution, and then click **Build Solution**.
5. Right-click the **SPGSharePointDataAccessQuickStart.SharePoint** project, and then click **Deploy**.

Note:

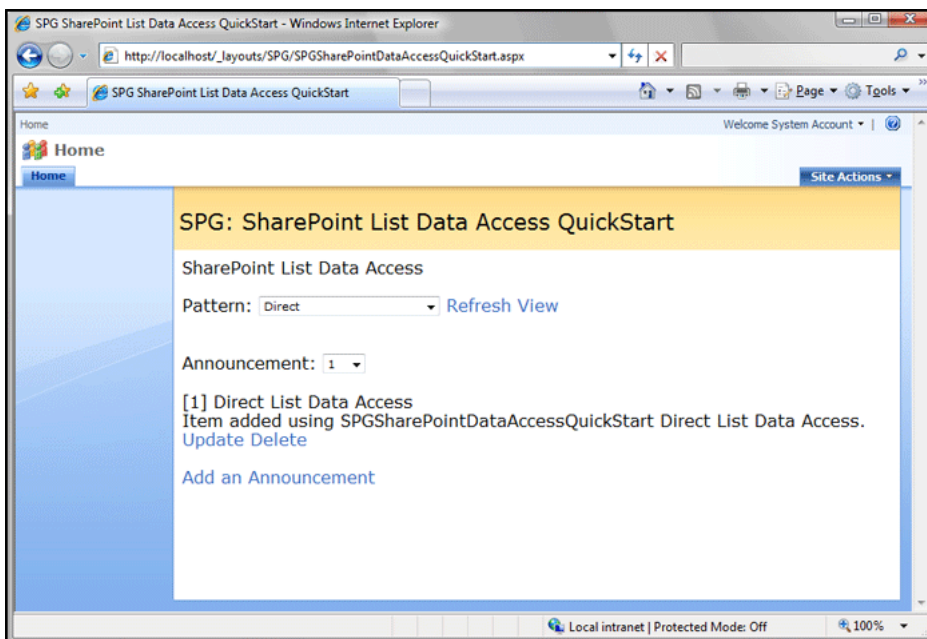
The deployment process may fail if the Visual Studio extensions for Windows SharePoint Services application pool identity does not have proper permissions to deploy to `http://localhost`. Add the Visual Studio extensions for Windows SharePoint Services application pool identity as a full control user on the `http://localhost` site collection. To do this, click **Site Actions**, point to **Site Settings**, and then click **People And Groups**. Click **New**. In the **Users/Groups** field, enter Visual Studio extensions for Windows SharePoint Services application pool identity. In the **Permissions** area, select **Full Control**. After you do this, repeat step 5.

6. In the Web browser, navigate to `http://{servername}/_layouts/SPG/SPGSharePointDataAccessQuickStart.aspx`.

Using the List Data Access Interface

The following illustration shows the List Data Access user interface.

List data access interface



The page contains the following elements:

- A drop-down box to select the list data access pattern

- A link to refresh the page
- A drop-down box that contains a list of the **SPListItem** IDs in the Announcements list
- An **Add an Announcement** link to add a new item to the Announcements list using the selected pattern
- The title and description of the currently selected list item
- A link to update the description of the currently selected item
- A link to delete the currently selected item

You can experiment with this page to add, update, and delete items in the Announcements list.

To use the application

1. In the **Pattern** drop-down box, click **Direct**, **ListItemRepository**, or **AnnouncementRepository**.
2. Click **Add an Announcement** to add a new announcement to the list.
3. You see a new ID listed in the drop-down box.
4. Select that new ID to show the new announcement.
5. Click **Update** to update the description of the announcement.
6. Click **Delete** to delete the announcement.

The application behaves similarly regardless of the pattern you select. However, when you examine the code paths with the Visual Studio debugger, you see that a different technique is used for each pattern.

Using the Visual Studio Debugger for a Code Walkthrough

The following section demonstrates how to use the Visual Studio debugger to understand the code.

To use the Visual Studio debugger

1. In Visual Studio, in the SPGSharePointDataAccessQuickStart.SharePoint project, open the `\Templates\Layouts\SPG\SPGSharePointDataAccessQuickStart.aspx.cs` file and set a breakpoint at the start of the **Page_Load** method.
2. In Visual Studio, click **Attach to process** on the **Tools** menu.
3. Attach to all the W3wp.exe processes.
4. In the user interface, add, update, or delete an announcement (for information, see the previous procedure). The debugger breaks at the first line of the **Page_Load** method.
5. Use the F10 key (Step Over) and the F11 key (Step Into) to step through the code to see how the pattern you chose is implemented. You can also set additional breakpoints.

As you debug the Direct pattern, you will notice that it directly invokes the methods of the **SPList** class and the **SPListItem** class. Lists are identified with GUIDs and string constants.

In the List Item Repository pattern, calls to the **SPList** class and the **SPListItem** class are abstracted by the **ListItemRepository** class. This class insulates you from the details of directly invoking the SharePoint object model.

The Entity Repository pattern uses two layers to abstract the interaction with the **SPListItem** class. The first layer is a business entity abstraction that is provided by the **Announcement** class and by the **AnnouncementRepository** class. The second layer is provided by the **ListItemRepository** class. The business entity abstraction allows you to use an application-level **Announcement** data type as an argument to the **Add**, **Update**, and **Delete** operations.

Topics at a Glance

This section is an overview of items this guidance contains. It includes the following topics:

- [Development How-to Topics](#). These topics give step-by-step instructions for performing common development tasks.
- [Developer QuickStarts](#). These topics describe stand-alone Visual Studio solutions for common development tasks.
- [Guidelines](#). These topics give guidance on how to create effective SharePoint applications.
- [Project and Feature Map](#). This section gives links to topics that describe the contents of the software assets that are included in this guidance.

Development How-to Topics

This guidance contains the following How-to topics for specific development tasks:

- [How to: Create a Custom Content Type with Event Receivers](#). This topic demonstrates how to create a custom content type in Visual Studio using Visual Studio extensions for Windows SharePoint Services version 1.3.
- [How to: Wrap a User Control Inside of a Web Part for SharePoint](#). This topic demonstrates how to create a Web Part to serve as a wrapper for an ASP.NET user control that is hosted inside of a SharePoint application. It also demonstrates how to create Web Part properties that are propagated to the user control.
- [How to: Debug SharePoint Applications](#). This topic includes procedures that show how to debug your application with Visual Studio extensions for Windows SharePoint Services. It also shows how to perform manual and remote debugging.
- [How to: Implement a SharePoint Workflow with ASP.NET Forms](#). This topic shows how to use Visual Studio to create ASP.NET custom workflow task forms.
- [How to: Perform ASP.NET-Related Development with Visual Studio](#). This topic demonstrates how custom ASPX pages and custom ASCX controls can be developed in a Web application project and then deployed using an extensions project.
- [How to: Programmatically Configure a WCF Client Endpoint](#). This topic includes procedures that describe how developers can create a SharePoint Web Part that uses data from a Windows Communication Foundation (WCF) Web service. The Web Part programmatically configures the WCF clients.
- [How to: Use the Hierarchical Configuration to Store an Endpoint Address](#). This topic demonstrates how to programmatically store and retrieve the endpoint addresses of WCF services with the Web Part that is implemented in [How to: Programmatically Configure a WCF Endpoint](#).
- [How to: Create an Automated Build and Deployment Solution with Team Foundation Server Team Build](#). This topic walks you through setting up a Build definition and setting up a computer to run the defined build. The Build function sets up and runs MSBuild on a targeted computer that is running the Visual Studio Team Foundation Server build agent. The build agent provides continuous building and testing using a Visual Studio Team System project.
- [Service Locator Development How-to Topics](#). The SharePoint service locator is a reusable component that allows you to decouple consumers of an interface from the implementations of that interface. Instead of creating an object by invoking the constructor of a class, you request an object with a specified interface from the service locator. The specific tasks are the following:
 - [How to: Register Services with the SharePoint Service Locator](#)
 - [How to: Get Services from the SharePoint Service Locator](#)
- [Configuration Manager Development How-to Topics](#). The SharePoint Guidance Library includes a hierarchical configuration manager that can safely store and retrieve configuration settings using SharePoint property bags. The topic includes the following procedures:
 - [How to: Add and Update Configuration Settings Using Configuration Manager](#). This topic describes how to save configuration data to any of the four levels that make up the SharePoint context.
 - [How to: Remove a Configuration Setting Using Configuration Manager](#). This topic describes how to remove configuration data from any of the four levels that make up the SharePoint context.
 - [How to: Retrieve Configuration Settings Using Configuration Manager](#). This topic describes how to retrieve configuration data from any of the four levels that make up the SharePoint context.
- [List Repository Development How-to Topics](#). A repository is a design pattern that separates a data source from its associated business logic by encapsulating the data access details. Repositories translate the underlying data representation into an entity model that fits the problem domain. The SharePoint Guidance Library includes classes that help you to implement repositories for SharePoint lists. The topic includes the following procedures:
 - [How to: Create a List Repository](#). This topic describes how to create a repository class for a SharePoint list.
 - [How to: Submit a Query to a List](#). This topic describes how to use the **CAMLQueryBuilder** class to retrieve a list item from a SharePoint list.
 - [How to: Populate a Business Entity from a List Item](#). This topic demonstrates how to convert a list item into a strongly-typed business entity.
 - [How to: Save a Business Entity to a List](#). This topic demonstrates how to convert a business entity into a list item and then add it to a SharePoint list.
- [How to: Create a Custom Subsite Creation Workflow](#). The SharePoint Guidance Library provides a subsite creation component. The subsite creation component creates subsites that are based on site templates and site definitions. The component includes a SharePoint sequential workflow and configuration data that is stored in a SharePoint list. This procedure demonstrates how to create a custom workflow by reusing activities that are included in the SharePoint Guidance Library.
- [How to: Use the Safe Script Manager to Provide AJAX Support to Web Parts](#). The **SafeScriptManager** class is an ASP.NET custom control that loads the .NET Framework's **ScriptManager** control if it is not already present on the current Web page. The procedure demonstrates how to add the safe script manager, either in code or by including it on a custom master page.
- [Logging and Tracing Development How-to Topics](#). The SharePoint Guidance Library includes a basic logging and tracing component that is tailored to the SharePoint environment. This topic includes the following procedures:
 - [How to: Log an Event for Operations](#). This topic describes how to log messages that are meant for system administrators.

- [How to: Log an Exception to Operations](#). This topic describes how to log unhandled exceptions.
- [How to: Create a Trace Message During Development](#). This topic describes how to write debugging information to the Unified Logging Service (ULS) trace log.
- [How to: Customize the Logger for Unit Testing](#). This topic describes how to create a mock logger for unit testing.
- [How to: Customize the Logger in an Application](#). This topic describes how to use a custom logger.
- [How to: Select a Custom Event Source Name](#). This topic describes how to use an event source other than the default event source.
- [How to: Expose BDC Profile Pages](#). By default, the profile page is hosted on the Shared Services Administration site collection where the Business Data Catalog (BDC) administration site is located. This procedure describes how you can create your own BDC profile pages and host them in any site that is not part of the Shared Services Provider site collection.
- [How to: Create Automated UI Test Methods for SharePoint](#). This topic explains how to use the Microsoft Visual Studio Team System Web test record and playback engine.
- [How to: Lock Down Identity Viewing for a Site](#). This topic includes a procedure that shows how to prevent unauthorized users from seeing the People.aspx page.
- [How to: Expose a SharePoint Application to the Extranet and Use Forms-Based Authentication](#). This topic includes a procedure that shows how to extend a SharePoint application to the extranet and enable the application to use forms-based authentication.

Developer QuickStarts

QuickStarts are Visual Studio solutions that demonstrate general techniques for developing SharePoint applications. This guidance includes the following QuickStarts:

- [Debugging an SPItemEventReceiver](#). This QuickStart demonstrates how to debug an **SPItemEventReceiver** that was created with Visual Studio extensions for Windows SharePoint Services.
- [Accessing SharePoint List Items](#). This QuickStart demonstrates three patterns for programmatically accessing SharePoint list items.

Guidelines

For more information about specific guidance for SharePoint applications, you can explore the following guidance sections:

- [Developing SharePoint Applications](#). This topic contains an overview of the SharePoint platform for ASP.NET developers and a quick tour of the Partner Portal application. It is intended for developers who are familiar with ASP.NET concepts but who are new to SharePoint.
- [Patterns](#). Patterns provide general solutions to commonly occurring problems. This guidance describes patterns that can be adapted to the SharePoint context. These include the following application and design patterns:
 - [Event-Driven Site Creation](#). This pattern describes a model for creating a site in response to an event.
 - [Application Instance Resolution](#). This pattern describes a model that automatically directs users to the appropriate area for information partitioned across site collections.
 - [Business Event Coordination](#). This pattern describes a model for coordinating events between external systems and SharePoint.
 - [The Model-View-Presenter \(MVP\) Pattern](#). The MVP pattern maximizes the amount of code that can be tested with automation and separates business logic from user interface (UI) logic to make the code easier to understand and maintain.
 - [The Repository Pattern](#). This pattern separates the logic that retrieves application data. The repository separates the low-level entity model from the business logic that acts on the model.
 - [The Service Locator Pattern](#). The service locator decouples classes from their dependencies so that these dependencies can be replaced or updated with little or no change to the classes. This is useful for a number of testing, versioning, and deployment scenarios.
 - [The Trusted Façade Pattern](#). The trusted façade helps you authenticate external users of a SharePoint application that is located in a perimeter network.
- [Integrating Line-of-Business Systems](#). This section discusses how to integrate line-of-business (LOB) systems with SharePoint. It describes how to consume LOB services in SharePoint and expose services from SharePoint. It also gives security considerations for LOB integration.
- [Considerations for Content-Driven Applications](#). Creating a content-driven application can involve many people with skills in many different areas. Each area has its own challenges. This section includes the following topics:
 - [Integrating Publishing Sites and Collaboration Sites](#). Microsoft Office SharePoint Server 2007 includes publishing capabilities. This topic describes how publishing sites can be integrated with SharePoint collaboration sites.
 - [Techniques for Aggregating List and Site Information](#). Although SharePoint offers several approaches that minimize the impact of aggregating data, developers and architects should perform this task judiciously. This topic gives guidance for aggregating list and site information.
 - [Cross-Site Collection Navigation](#). Cross-site collection navigation refers to a Web-based user interface that allows you to access sites that are located in different site collections. This topic explains how to use XML to customize the **SPXmlContentMapProvider** class and how to implement global navigation by creating a custom navigation solution.
 - [Understanding Custom Site Navigation](#). This topic gives the general steps that are required to implement a breadcrumb user interface for custom site navigation.
 - [Developing Custom Publishing Page Layouts](#). This topic describes how to use rich HTML field controls, how to develop custom field controls, and how to use Web Parts on a publishing page.
 - [Options for Site Branding](#). This topic describes the site branding options that are available in SharePoint. Site branding is the process of customizing a Web site to reflect organizational identity.
 - [Understanding Publishing and Content Deployment](#). This topic explains how to deploy content from an authoring farm to a production farm.
 - [Using the Central Template Gallery](#). This topic gives procedures for customizing the central template gallery using the STSADM command.
 - [Extracting Artifacts Created with SharePoint Designer and the Browser](#). This topic describes how to extract authored artifacts that are stored in the content database and convert them into source files that you can manage as Visual Studio solution artifacts under version control.
- [Considerations for Enterprise-Scale Applications](#). This section of the guidance addresses some common scenarios in enterprise-scale application development. Considerations for enterprise-scale applications include the following topics:
 - [Managing Application Configuration](#). This topic covers the various application configuration options available to SharePoint developers.
 - [Providing Application Diagnostics](#). This topic addresses some techniques and tools for creating code that is easy to instrument and troubleshoot.
 - [Improving Application Quality Through Testing](#). This topic provides a comprehensive overview of the types of testing needed for enterprise applications.
 - [Building for Scale](#). For an enterprise application to scale appropriately, the underlying code must be written in a scalable way and sufficient stress testing needs to be performed to evaluate its behavior under normal and excessive user load. The Building for Scale section covers some of the decisions and techniques needed for creating scalable SharePoint code. It also describes scale and stress testing for SharePoint applications.
- [Considerations for Extranet Development](#). This topic briefly describes some of the security issues you should

consider when you develop an extranet for SharePoint. It also links to the relevant SharePoint product documentation and gives several procedures.

Project and Feature Map

This guidance comes with a downloadable library of reusable components and two reference implementations.

For a description of the reusable components, see [The SharePoint Guidance Library](#).

For information about the Partner Portal application, see [Partner Portal Reference Implementation](#).

For a description of the Training Management application, see [Training Management Reference Implementation](#).

Glossary

This glossary includes definitions of terms that appear in the documentation.

Terms

adding a solution. The process of uploading a SharePoint solution to a SharePoint solution store.

application. The set of functionality under development to solve a particular business problem. See *SharePoint application*.

application page. An ASP.NET page that is located in the `_layouts` folder in SharePoint and cannot be customized.

artifact. An asset of an application or a project deliverable, such as binaries, pages, styles, XML definitions, and layouts. Artifacts may reside in the file system or in the database.

authentication provider. Software service responsible for validating user credentials. Also known as *membership provider*.

authored artifacts. Customized solution artifacts or new artifacts that are created using SharePoint Designer or the Web browser after a solution is deployed. They are stored in a SharePoint content database. See *artifact*.

Central Administration. A Web application provided by SharePoint for administering SharePoint Web applications. Also known as *Centralized Administration Site*.

column. See *list column* and *site column*.

command-line tool. See *SharePoint command-line tool*.

content database. The backend storage that holds SharePoint customizations, user documents, and so on.

content page. A customizable SharePoint Web page that contains user-provided content or supports customization, such as a Web Part page. Compare with *application page*.

content type. A reusable collection of settings to define the metadata and behavior for a certain category of content.

custom action. A link, toolbar button, menu item, or any control that can be added to a toolbar or menu that a user sees in SharePoint. For example, the Contoso Training Management application provides a custom action for registering for a course.

custom field type. See *field type*.

deploying a solution. The process of extracting and moving SharePoint solution artifacts to SharePoint Web applications.

Designer. See *SharePoint Designer*.

document library. A SharePoint list that contains documents.

event handler. A server-side code routine that is invoked when a predefined condition occurs, such as when a new item is added to a list. May be associated with a Web application, site collection, site, list, or document library.

feature. See *SharePoint feature* and *Web feature*.

feature receiver. A server-side code routine that is called as part of installation, activation, deactivation, or removal of a feature. Also known as *feature event receiver*.

feature stapling. The process of associating a SharePoint feature with a specific site definition, without modifying the site definition or creating code routines to activate the feature on each site. Also known as *feature site template association*.

field. An atomic unit of data storage. List items contain one or more fields. Fields may also exist at the site level. See *list column* and *site column*.

field control. A programmer-provided user interface for editing and displaying a field.

field reference. A definition placed in a content type for the purpose of including a site column or list column. The SharePoint object model uses the class **SPFieldLink** to represent field references. Note that field references can also occur in lists and list views. Field references in lists and content types can only reference site columns. Field references in list views can reference both list columns and site columns.

field type. Defines a type that can be used as a field or column in SharePoint. SharePoint has a set of standard field types, and developers may also define custom field types. For more information, see [Custom Field Types](#) on MSDN.

group. See *SharePoint group*.

IIS Web site. An application boundary defined by Microsoft Internet Information Services (IIS); associated with a single SharePoint Web application.

image library. A SharePoint list that contains images.

library. See *document library* and *image library*.

list. See *SharePoint list*.

list column. The user interface that displays a field. Part of a SharePoint list. See *field*.

list definition. The schema for a SharePoint list. May contain user customizations stored in the content database.

list item. A record in a SharePoint list.

list item event handler. A method that is invoked by SharePoint when list items are added or modified.

list template. An element of the site definition that defines the initial structure of a list.

master page. A template used by ASP.NET that allows the user to apply a similar appearance to multiple Web pages.

mock object. An instance of a class that impersonates the behavior of another class for the purposes of testing. Mock objects provide an isolated environment for unit testing.

Model-View-Presenter (MVP) pattern. A software design pattern that formalizes the data flow between components by means of an interface.

object model. See *SharePoint object model*.

personalization. A feature provided by SharePoint that allows users to change site elements. Changes are persisted in the content database. Also known as *customization*.

permission level. A named capability used in access control. For example, Read and Write are permission levels.

port. A TCP port. Port numbers may appear in URLs after the server name. For example, in <http://myserver:80/default.aspx>, the port number is 80.

Quick Launch bar. A user interface element with links to commonly used pages of the current SharePoint site.

removing a solution. The process of removing a SharePoint solution from the SharePoint solution store.

Repository pattern. A programming idiom that isolates the details of data-store access into a software component.

retracting a solution. The process of removing a deployed solution from a server.

root Web. See *top-level site*.

server farm. A collection of servers for the purpose of load balancing or a collection of servers that perform different purposes, such as a Web front-end server, an application server, a database server, and an indexing server.

server. A host running IIS.

server control. An ASP.NET server control, which is a server-side software component that determines how a particular region of Web page will be rendered as HTML.

Service Locator pattern. A programming idiom that replaces static library dependencies with a run-time component responsible for producing an object that supports a specific interface.

SharePoint application. An application designed to run on SharePoint. SharePoint applications are packaged in SharePoint solutions, and the functionality is typically encapsulated in SharePoint features for modularity and management.

SharePoint command-line tool. The STSADM.exe executable.

SharePoint Designer. A client application for customizing the visual appearance and page elements of a SharePoint site.

SharePoint feature. A modular unit of functionality that can be individually activated or deactivated by the administrator for each site. Each feature is defined by a Feature.xml file.

SharePoint group. A collection of users for the purpose of access control. Permission levels can be set on a per-group basis.

SharePoint list. A structure of items or rows and columns or fields that contain data provided by SharePoint similar to a database table. The SharePoint object model uses the class **SPList** to represent a SharePoint list.

SharePoint object model. A .NET-based application programming interface (API) to SharePoint components.

SharePoint theme. A collection of graphics and cascading style sheets that control the appearance of a site.

SharePoint solution. An installation package in CAB format that is a deployable unit for SharePoint. A SharePoint application is packaged in one or more SharePoint solutions. Also known as *Web solution package (WSP)*. The file name extension is .wsp. The packaged files may include executable libraries (DLLs), site definitions, features, and other files. Contrast with *Visual Studio solution*.

SharePoint solution store. A centralized collection of all SharePoint solutions for a server farm.

SharePoint Web application. A load-balanced Web application for SharePoint that is based in Internet Information Services (IIS). For information, see [Server and Site Architecture: Object Model Overview](#) on MSDN.

site. A SharePoint Web site. The SharePoint object model uses the class **SPWeb** to represent a site. For historical reasons, sometimes referred to as a *Web*, a *site instance*, or a *Web site instance*. For more information, see [Server and Site Architecture: Object Model Overview](#) on MSDN.

Site Actions menu. A drop-down menu that provides the ability to manage a SharePoint site and create new subsites.

site collection. Represents a collection of sites in a Web application, including a top-level Web site and all its subsites. A site collection acts as an administrative unit that allows for common security permissions and list templates, content types, and workflows to be shared by multiple sites. The SharePoint object model uses the class **SPSite** to represent a site collection. Sometimes (for historical reasons) referred to as a *site*. Properly, however, this is a site collection. For more information, see [Server and Site Architecture: Object Model Overview](#) on MSDN.

site column. A site column, also known as a field, is a reusable column definition, or template, that can be assigned to multiple lists or content type definitions across multiple SharePoint sites. Site columns are reusable and help ensure the consistency of metadata across sites and lists. The SharePoint object model uses the class **SPField** to represent a site column. Also known as *column template* or *field*.

site definition. A file-based representation of the structure of the site. Contains a server-side collection of files that defines the structure of one or more configurations. For more information, see [Site Definition](#) on MSDN.

site feature. A modular unit of functionality that may be individually activated or deactivated by the administrator for each site or site collection. Each feature is defined by a Feature.xml file in a solution.

site template. A customized site design based on an existing site definition stored in an .stp file. This is the delta or difference between a site and its underlying site definition. Applies mostly to layout and appearance. For more information, see [Site Template](#) on MSDN.

solution. See *SharePoint solution*.

solution artifacts. Files created by developers or designers that are packaged into a SharePoint solution (WSP). After solution artifacts are deployed, they are available for activation across all sites within the deployed scope (SharePoint Web application or SharePoint farm).

subsites. Refers to any site in a site collection other than the first (or top-level) site. Also known as *subwebs*.

theme. See *SharePoint theme*.

top-level site. The first site in a site collection. Allows administration of attributes shared by all sites in the site collection. Also known as *root Web*.

user control. An ASP.NET user control. This is a programmer-provided visual element that appears on a Web page running in the ASP.NET environment.

unit testing. A software testing technique that focuses on exercising the features of individual functions or modules in isolation. Can be contrasted with system-level testing.

Visual Studio extensions for Windows SharePoint Services. A Visual Studio project type that supports the creation of SharePoint applications.

Visual Studio project. A configuration and packaging division within a Visual Studio solution. It is typical that each project builds a single library (DLL) or executable (.exe file).

Visual Studio solution. A file that contains all of the settings needed to build and debug a specific application being developed in Visual Studio.

Web application. See *SharePoint Web application*.

Web content. Content such as text and images that is created by content authors.

Web feature. A unit of functionality that may be individually activated or deactivated by the administrator for each site. Each feature is defined by a Feature.xml file.

Web Part. An ASP.NET server control that allows users to customize the appearance or behavior of the Web page.

workflow. A business process that proceeds in discrete operational steps, implemented using the Windows Workflow Foundation library.

workflow activity. A .NET Framework class that implements the behavior of a particular operational step of a workflow.

workflow instance. A workflow plus run-time data that indicates the current state of a running business process.

WSP file. See *SharePoint solution*.

Appendix

This appendix contains a list of publishing field types and content types. It also contains a list of build verification tests used by the Partner Portal application. The topics are the following:

- [Publishing Field Types and Content Types](#). This topic lists the standard field types and content types that are specifically for content types.
- [Build Verification Tests in the Partner Portal Application](#). This topic lists and describes the build verification tests that were used with the Partner Portal application. The Partner Portal application includes BVTs that use the Visual Studio Team System continuous testing system. The test builds and packages the solution, installs the solution on a SharePoint server, and executes a series of tests.

Publishing Field Types and Content Types

Microsoft Office SharePoint Server 2007 includes [content types](#) and [field types](#) that are specific to publishing sites. If possible, use these standard types on your publishing sites instead of custom types. The standard publishing field types for Microsoft Office SharePoint Server 2007 are defined in PublishingColumns.xml. The standard publishing content types are defined in PublishingContentTypes.xml. These files are typically located at %Program Files%\Common Files\Microsoft Shared\Web Server Extensions\12\TEMPLATE\FEATURES\PublishingResources. They are included in this appendix for convenience.

Publishing Field Types

The following are commonly used field types:

- [Page fields](#)
- [Chrome and page layout fields](#)
- [Cache fields](#)
- [Welcome page fields](#)
- [Article page fields](#)
- [Reusable text fields](#)
- [Content query fields](#)
- [Redirect page layout fields](#)

Page Fields

XML

```
<Field ID="{51d39414-03dc-4bd0-b777-d3e20cb350f7}" Name="PublishingStartDate"
StaticName="PublishingStartDate" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_publishingcolumns;"
DisplayName="$Resources:cmscore,column_startdate_displayname;"
Type="PublishingScheduleStartDateFieldType" Required="FALSE" CanToggleHidden="TRUE"
Sealed="TRUE" StorageTZ="UTC" />

<Field ID="{a990e64f-faa3-49c1-aafa-885fda79de62}" Name="PublishingExpirationDate"
StaticName="PublishingExpirationDate" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_publishingcolumns;"
DisplayName="$Resources:cmscore,column_expirationdate_displayname;"
Type="PublishingScheduleEndDateFieldType" Required="FALSE" CanToggleHidden="TRUE"
Sealed="TRUE" StorageTZ="UTC" />

<Field ID="{ae1a4dd-0f19-417d-8721-95a1d28762ab}" Name="PublishingContact"
StaticName="PublishingContact" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_publishingcolumns;"
DisplayName="$Resources:cmscore,column_contact_displayname;" Type="User" Required="FALSE"
Sealed="TRUE" List="UserInfo" />

<Field ID="{c79dba91-e60b-400e-973d-c6d06f192720}" Name="PublishingContactEmail"
StaticName="PublishingContactEmail" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_publishingcolumns;"
DisplayName="$Resources:cmscore,column_contactemail_displayname;" Type="Text"
Required="FALSE" Sealed="TRUE" MaxLength="255" />

<Field ID="{7546ad0d-6c33-4501-b470-fb3003ca14ba}" Name="PublishingContactName"
StaticName="PublishingContactName" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_publishingcolumns;"
DisplayName="$Resources:cmscore,column_contactname_displayname;" Type="Text" Required="FALSE"
Sealed="TRUE" MaxLength="255" />

<Field ID="{dc47d55f-9bf9-494a-8d5b-e619214dd19a}" Name="PublishingContactPicture"
StaticName="PublishingContactPicture" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_publishingcolumns;"
DisplayName="$Resources:cmscore,column_contactpicture_displayname;" Type="URL" Format="Image"
Required="FALSE" Sealed="TRUE" />

<Field ID="{0f800910-b30d-4c8f-b011-8189b2297094}" Name="PublishingPageLayout"
StaticName="PublishingPageLayout" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_publishingcolumns;"
DisplayName="$Resources:cmscore,column_pagelayout_displayname;" Type="URL" Required="FALSE"
Sealed="TRUE" ReadOnly="TRUE" />

<Field ID="{914fdb80-7d4f-4500-bf4c-ce46ad7484a4}" Name="PublishingVariationGroupID"
StaticName="PublishingVariationGroupID" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="Hidden" DisplayName="$Resources:cmscore,column_variationgroupid_displayname;"
Type="Text" Required="FALSE" Sealed="TRUE" Hidden="TRUE" MaxLength="255" />
```

```
<Field ID="{766da693-38e5-4b1b-997f-e830b6dfcc7b}"
Name="PublishingVariationRelationshipLinkFieldID"
StaticName="PublishingVariationRelationshipLinkFieldID"
SourceID="http://schemas.microsoft.com/sharepoint/v3" Group="_Hidden"
DisplayName="$Resources:cmscore,column_variationrelationshiplinkfieldid_displayname;"
Type="URL" Required="FALSE" Sealed="TRUE" Hidden="TRUE" />
<Field ID="{61cbb965-1e04-4273-b658-eedaa662f48d}" Name="Audience" StaticName="Audience"
SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_publishingcolumns;"
DisplayName="$Resources:cmscore,PageContentType_TargetToField_DisplayName;" Type="TargetTo"
Required="FALSE" Sealed="TRUE" />
```

Chrome and Page Layout Fields

XML

```
<Field ID="{188ce56c-61e0-4d2a-9d3e-7561390668f7}" Name="PublishingPreviewImage"
StaticName="PublishingPreviewImage" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_previewimage_displayname;" Type="URL"
Required="FALSE" Sealed="TRUE" />

<Field ID="{7581e709-5d87-42e7-9fe6-698ef5e86dd3}" Name="PublishingHidden"
StaticName="PublishingHidden" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_hidden_displayname;" Type="Boolean"
Required="FALSE" Sealed="TRUE" />

<Field ID="{75bed596-0661-4edd-9724-1d607ab8d3b5}" Name="_PublishingMigratedGuid"
StaticName="_PublishingMigratedGuid" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_migratedguid_displayname;" Type="Guid"
Required="FALSE" Sealed="TRUE" Hidden="TRUE" ReadOnly="TRUE" />

<Field ID="{b510aac1-bba3-4652-ab70-2d756c29540f}" Name="PublishingAssociatedContentType"
StaticName="PublishingAssociatedContentType"
SourceID="http://schemas.microsoft.com/sharepoint/v3" Group="_Hidden"
DisplayName="$Resources:cmscore,column_associatedcontenttype_displayname;"
Type="ContentTypeIdFieldType" Required="FALSE" Sealed="TRUE" />

<Field ID="{d211d750-4fe6-4d92-90e8-eb16dff196c8}" Name="PublishingAssociatedVariations"
StaticName="PublishingAssociatedVariations"
SourceID="http://schemas.microsoft.com/sharepoint/v3" Group="_Hidden"
DisplayName="$Resources:cmscore,column_associatedvariations_displayname;"
Type="LayoutVariationsField" Required="FALSE" Sealed="TRUE" />
```

Cache Fields

XML

```
<Field ID="{983f490b-fc53-4820-9354-e8de646b4b82}" Name="PublishingCacheDisplayName"
StaticName="PublishingCacheDisplayName" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_cachedisplayname_displayname;"
Type="Text" Description="$Resources:cmscore,column_cachedisplayname_description;"
Required="FALSE" Sealed="TRUE" MaxLength="255" />

<Field ID="{9550e77a-4d10-464f-bc0c-102d5b1aec42}" Name="PublishingCacheDisplayDescription"
StaticName="PublishingCacheDisplayDescription"
SourceID="http://schemas.microsoft.com/sharepoint/v3" Group="_Hidden"
DisplayName="$Resources:cmscore,column_cachedisplaydescription_displayname;" Type="Text"
Description="$Resources:cmscore,column_cachedisplaydescription_description;" Required="FALSE"
Sealed="TRUE" MaxLength="255" />

<Field ID="{db03cb99-cf1e-40b8-adc7-913f7181dac3}" Name="PublishingCachePerformACLCheck"
StaticName="PublishingCachePerformACLCheck"
SourceID="http://schemas.microsoft.com/sharepoint/v3" Group="_Hidden"
DisplayName="$Resources:cmscore,column_cacheperformaclcheck_displayname;" Type="Boolean"
Description="$Resources:cmscore,column_cacheperformaclcheck_description;" Required="FALSE"
Sealed="TRUE" />

<Field ID="{d8f18167-7cff-4c4e-bdbe-e7b0f01678f3}" Name="PublishingCacheEnabled"
StaticName="PublishingCacheEnabled" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_cacheenabled_displayname;"
Type="Boolean" Description="$Resources:cmscore,column_cacheenabled_description;"
Required="FALSE" Sealed="TRUE" />

<Field ID="{bdd1b3c3-18db-4acf-a963-e70ef4227fbc}" Name="PublishingCacheDuration"
StaticName="PublishingCacheDuration" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_cacheduration_displayname;"
Type="Number" Description="$Resources:cmscore,column_cacheduration_description;" Min="0"
Required="FALSE" Sealed="TRUE" />

<Field ID="{5b4d927c-d383-496b-bc79-1e61bd383019}" Name="PublishingCacheCheckForChanges"
StaticName="PublishingCacheCheckForChanges"
SourceID="http://schemas.microsoft.com/sharepoint/v3" Group="_Hidden"
```



```

DisplayName="$Resources:cmscore,column_cachecheckforchanges_displayname;" Type="Boolean"
Description="$Resources:cmscore,column_cachecheckforchanges_description;" Required="FALSE"
Sealed="TRUE" />

<Field ID="{4689a812-320e-4623-aab9-10ad68941126}" Name="PublishingVaryByCustom"
StaticName="PublishingVaryByCustom" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_varybycustom_displayname;" Type="Text"
Description="$Resources:cmscore,column_varybycustom_description;" Required="FALSE"
Sealed="TRUE" />

<Field ID="{89587dfd-b9ca-4fae-8eb9-ba779e917d48}" Name="PublishingVaryByHeader"
StaticName="PublishingVaryByHeader" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_varybyheader_displayname;" Type="Text"
Description="$Resources:cmscore,column_varybyheader_description;" Required="FALSE"
Sealed="TRUE" MaxLength="255" />

<Field ID="{b8abfc64-c2bd-4c88-8cef-b040c1b9d8c0}" Name="PublishingVaryByParam"
StaticName="PublishingVaryByParam" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_varybyparam_displayname;" Type="Text"
Description="$Resources:cmscore,column_varybyparam_description;" Required="FALSE"
Sealed="TRUE" MaxLength="255" />

<Field ID="{d4a6af1d-c6d7-4045-8def-cefa25b9ec30}" Name="PublishingVaryByRights"
StaticName="PublishingVaryByRights" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_varybyrights_displayname;"
Type="Boolean" Description="$Resources:cmscore,column_varybyrights_description;"
Required="FALSE" Sealed="TRUE" />

<Field ID="{0A90B5E8-185A-4dec-BF3C-E60AAE08373F}" Name="PublishingAuthenticatedUse"
StaticName="PublishingAuthenticatedUse" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_authenticateduse_displayname;"
Type="Boolean" Description="$Resources:cmscore,column_authenticateduse_description;"
Required="TRUE" Sealed="TRUE" />

<Field ID="{773ED051-58DB-4ff2-879B-08B21AB001E0}" Name="PublishingCacheAllowWriters"
StaticName="PublishingCacheAllowWriters"
SourceID="http://schemas.microsoft.com/sharepoint/v3" Group="_Hidden"
DisplayName="$Resources:cmscore,column_allowwriters_displayname;" Type="Boolean"
Description="$Resources:cmscore,column_allowwriters_description;" Required="FALSE"
Sealed="TRUE" />

<Field ID="{18f165be-6285-4a57-b3ab-4e9f913d299f}" Name="PublishingCacheability"
StaticName="PublishingCacheability" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_cacheability_displayname;"
Type="Choice" Description="$Resources:cmscore,column_cacheability_description;"
Required="TRUE" Sealed="TRUE">
  <CHOICES>
    <CHOICE>NoCache</CHOICE>
    <CHOICE>Private</CHOICE>
    <CHOICE>Server</CHOICE>
    <CHOICE>ServerAndNoCache</CHOICE>
    <CHOICE>Public</CHOICE>
    <CHOICE>ServerAndPrivate</CHOICE>
  </CHOICES>
</Field>

```

Welcome Page Fields

XML

```

<Field ID="{3894ec3f-4674-4924-a440-8872bec40cf9}" Name="PublishingPageIcon"
StaticName="PublishingPageIcon" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_pagelayoutcolumns;"
DisplayName="$Resources:cmscore,column_pageicon_displayname;" Type="Image" Required="FALSE"
Sealed="TRUE" RichText="TRUE" RichTextMode="FullHtml" />

<Field ID="{3de94b06-4120-41a5-b907-88773e493458}" Name="PublishingPageImage"
StaticName="PublishingPageImage" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_pagelayoutcolumns;"
DisplayName="$Resources:cmscore,column_pageimage_displayname;" Type="Image" Required="FALSE"
Sealed="TRUE" RichText="TRUE" RichTextMode="FullHtml" />

<Field ID="{F55C4D88-1F2E-4ad9-AAA8-819AF4EE7EE8}" Name="PublishingPageContent"
StaticName="PublishingPageContent" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_pagelayoutcolumns;"
DisplayName="$Resources:cmscore,column_pagecontent_displayname;" Type="HTML" Required="FALSE"
Sealed="TRUE" RichText="TRUE" RichTextMode="FullHtml" />

<Field ID="{B3525EFE-59B5-4f0f-B1E4-6E26CB6EF6AA}" Name="SummaryLinks"
StaticName="SummaryLinks" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_pagelayoutcolumns;"

```

```
DisplayName="$Resources:cmscore,column_summarylinks_displayname;" Type="SummaryLinks"
Required="FALSE" Sealed="TRUE" RichText="TRUE" RichTextMode="FullHtml" />
```

```
<Field ID="{27761311-936A-40ba-80CD-CA5E7A540A36}" Name="SummaryLinks2"
StaticName="SummaryLinks2" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_pagelayoutcolumns;"
DisplayName="$Resources:cmscore,column_summarylinks2_displayname;" Type="SummaryLinks"
Required="FALSE" Sealed="TRUE" RichText="TRUE" RichTextMode="FullHtml" />
```

Article Page Fields

XML

```
<Field ID="{D3429CC9-ADC4-439b-84A8-5679070F84CB}" Name="ArticleByLine"
StaticName="ArticleByLine" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_pagelayoutcolumns;"
DisplayName="$Resources:cmscore,column_articlebyline_displayname;" Type="Text"
Required="FALSE" Sealed="TRUE" MaxLength="255" />

<Field ID="{71316CEA-40A0-49f3-8659-F0CEFDDBD4F}" Name="ArticleStartDate"
StaticName="ArticleStartDate" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_publishingcolumns;"
DisplayName="$Resources:cmscore,column_articlestartdate_displayname;" Type="DateTime"
Format="DateOnly" Required="FALSE" Sealed="TRUE" StorageTZ="UTC" />

<Field ID="{66F500E9-7955-49ab-ABB1-663621727D10}" Name="PublishingImageCaption"
StaticName="PublishingImageCaption" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_pagelayoutcolumns;"
DisplayName="$Resources:cmscore,column_imagecaption_displayname;" Type="HTML"
Required="FALSE" Sealed="TRUE" RichText="TRUE" RichTextMode="FullHtml" />

<Field ID="{a932ec3f-94c1-48b1-b6dc-41aaa6eb7e54}" Name="HeaderStyleDefinitions"
StaticName="HeaderStyleDefinitions" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_pagelayoutcolumns;"
DisplayName="$Resources:cmscore,column_headerstyledefinitions_displayname;" Type="HTML" />
```

Reusable Text Fields

XML

```
<Field ID="{3a4b7f98-8d14-4800-8bf5-9ad1dd6a82ee}" Name="ContentCategory"
StaticName="ContentCategory" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_contentcategory_displayname;"
Type="Choice" Format="Dropdown" FillInChoice="FALSE" Required="FALSE" Sealed="FALSE"
AllowDeletion="FALSE">
  <Default>None</Default>
  <CHOICES>
    <CHOICE>$Resources:cmscore,column_contentcategory_defaultvalue;</CHOICE>
  </CHOICES>
</Field>

<Field ID="{e977ed93-da24-4fcc-b77d-ac34eea7288f}" Name="AutomaticUpdate"
StaticName="AutomaticUpdate" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_automaticupdate_displayname;"
Description="$Resources:cmscore,column_automaticupdate_description;" Type="Boolean"
Required="FALSE" Sealed="TRUE">
  <Default>1</Default>
</Field>

<Field ID="{890e9d41-5a0e-4988-87bf-0fb9d80f60df}" Name="ReusableText"
StaticName="ReusableText" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_reusabletext_displayname;" Type="Note"
Required="FALSE" Sealed="TRUE" RichText="FALSE" />

<Field ID="{82dd22bf-433e-4260-b26e-5b8360dd9105}" Name="ReusableHtml"
StaticName="ReusableHtml" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="_Hidden" DisplayName="$Resources:cmscore,column_reusablehtml_displayname;" Type="HTML"
Required="FALSE" Sealed="TRUE" RichText="TRUE" RichTextMode="FullHtml" />
```

Content Query Fields

XML

```
<Field ID="{543BC2CF-1F30-488e-8F25-6FE3B689D9AC}" Name="PublishingRollupImage"
StaticName="PublishingRollupImage" SourceID="http://schemas.microsoft.com/sharepoint/v3"
Group="$Resources:cmscore,group_pagelayoutcolumns;"
DisplayName="$Resources:cmscore,column_rollupimage_displayname;" Type="Image"
Required="FALSE" Sealed="TRUE" RichText="TRUE" RichTextMode="FullHtml" />
```

Redirect Page Layout Fields

XML

```
<Field ID="{AC57186E-E90B-4711-A038-B6C6A62A57DC}" Name="RedirectURL"
StaticName="RedirectURL" SourceID="http://schemas.microsoft.com/sharepoint/v3"
```

```

Group="$Resources:cmscore,group_pagelayoutcolumns;"
DisplayName="$Resources:cmscore,column_redirecturl_displayname;" Type="URL" Required="FALSE"
/>

```

Publishing Content Types

This section lists the publishing content types. For information about how to interpret a content type ID and how to create your own custom content types, see [Content Type IDs](#) on MSDN.

XML

```

<ContentType ID="0x010100C568DB52D9D0A14D9B2FDCC96666E9F2"
Name="$Resources:cmscore,contenttype_systempage_name;"
Description="$Resources:cmscore,contenttype_systempage_description;" Group="_Hidden"
Sealed="TRUE" Version="0">
  <FieldRefs>
    <FieldRef ID="{9da97a8a-1da5-4a77-98d3-4bc10456e700}" Name="Comments"
  </FieldRefs>
  <FieldRef ID="{51d39414-03dc-4bd0-b777-d3e20cb350f7}" Name="PublishingStartDate"/>
  <FieldRef ID="{a990e64f-faa3-49c1-aa85-fda79de62}" Name="PublishingExpirationDate"/>
  <FieldRef ID="{aeala4dd-0f19-417d-8721-95a1d28762ab}" Name="PublishingContact"/>
  <FieldRef ID="{c79dba91-e60b-400e-973d-c6d06f192720}" Name="PublishingContactEmail"/>
  <FieldRef ID="{7546ad0d-6c33-4501-b470-fb3003ca14ba}" Name="PublishingContactName"/>
  <FieldRef ID="{dc47d55f-9bf9-494a-8d5b-e619214dd19a}" Name="PublishingContactPicture"/>
  <FieldRef ID="{0f800910-b30d-4c8f-b011-8189b2297094}" Name="PublishingPageLayout"/>
  <FieldRef ID="{914fdb80-7d4f-4500-bf4c-ce46ad7484a4}" Name="PublishingVariationGroupID"/>
  <FieldRef ID="{766da693-38e5-4b1b-997f-e830b6dfcc7b}" Name="PublishingVariationRelationshipLinkFieldID"/>
  <FieldRef ID="{543BC2CF-1F30-488e-8F25-6FE3B689D9AC}" Name="PublishingRollupImage"/>
  <FieldRef ID="{61cbb965-1e04-4273-b658-eedaa662f48d}" Name="Audience"/>
</FieldRefs>
</ContentType>

<ContentType ID="0x010100C568DB52D9D0A14D9B2FDCC96666E9F2007948130EC3DB064584E219954237AF39"
Name="$Resources:cmscore,contenttype_page_name;"
Description="$Resources:cmscore,contenttype_page_description;"
Group="$Resources:cmscore,group_publishingcontenttypes;" Sealed="FALSE" Version="0">
  <FieldRefs>
  </FieldRefs>
  <DocumentTemplate TargetName="/_layouts/CreatePage.aspx" />
</ContentType>

<ContentType ID="0x0101000F1C8B9E0EB4BE489F09807B2C53288F"
Name="$Resources:cmscore,contenttype_systemmasterpage_name;"
Description="$Resources:cmscore,contenttype_systemmasterpage_description;" Group="_Hidden"
Sealed="TRUE" Version="0">
  <FieldRefs>
    <FieldRef ID="{9da97a8a-1da5-4a77-98d3-4bc10456e700}" Name="Comments"
  </FieldRefs>
  <FieldRef ID="{aeala4dd-0f19-417d-8721-95a1d28762ab}" Name="PublishingContact"/>
  <FieldRef ID="{188ce56c-61e0-4d2a-9d3e-7561390668f7}" Name="PublishingPreviewImage"/>
  <FieldRef ID="{7581e709-5d87-42e7-9fe6-698ef5e86dd3}" Name="PublishingHidden"/>
</FieldRefs>
</ContentType>

<ContentType ID="0x0101000F1C8B9E0EB4BE489F09807B2C53288F0054AD6EF48B9F7B45A142F8173F171BD1"
Name="$Resources:cmscore,contenttype_masterpage_name;"
Description="$Resources:cmscore,contenttype_masterpage_description;"
Group="$Resources:cmscore,group_publishingcontenttypes;" Sealed="FALSE" Version="0">
  <FieldRefs>
  </FieldRefs>
  <DocumentTemplate TargetName="PublishingMasterTemplate.master" />
</ContentType>

<ContentType ID="0x01010007FF3E057FA8AB4AA42FCB67B453FFC1"
Name="$Resources:cmscore,contenttype_systempagelayout_name;"
Description="$Resources:cmscore,contenttype_systempagelayout_description;" Group="_Hidden"
Sealed="TRUE" Version="0">
  <FieldRefs>
    <FieldRef ID="{9da97a8a-1da5-4a77-98d3-4bc10456e700}" Name="Comments"
  </FieldRefs>
  <FieldRef ID="{aeala4dd-0f19-417d-8721-95a1d28762ab}" Name="PublishingContact"/>
  <FieldRef ID="{188ce56c-61e0-4d2a-9d3e-7561390668f7}"

```

```

Name="PublishingPreviewImage"/>
    <FieldRef ID="{7581e709-5d87-42e7-9fe6-698ef5e86dd3}" Name="PublishingHidden"/>
    <FieldRef ID="{75bed596-0661-4edd-9724-1d607ab8d3b5}" Name="_PublishingMigratedGuid"/>
    <FieldRef ID="{b510aac1-bba3-4652-ab70-2d756c29540f}" Name="PublishingAssociatedContentType" />
    <FieldRef ID="{d211d750-4fe6-4d92-90e8-eb16dff196c8}" Name="PublishingAssociatedVariations"/>
  </FieldRefs>
</ContentType>

<ContentType ID="0x01010007FF3E057FA8AB4AA42FCB67B453FFC100E214EEE741181F4E9F7ACC43278EE811"
Name="$Resources:cmscore,contenttype_pagelayout_name;"
Description="$Resources:cmscore,contenttype_pagelayout_description;"
Group="$Resources:cmscore,group_publishingcontenttypes;" Sealed="FALSE" Version="0">
  <FieldRefs>
  </FieldRefs>
  <DocumentTemplate TargetName="/_layouts/NewPageLayout.aspx" />
</ContentType>

<ContentType ID="0x010087D89D279834C94E98E5E1B4A913C67E"
Name="$Resources:cmscore,contenttype_pageoutputcache_name;"
Description="$Resources:cmscore,contenttype_pageoutputcache_description;" Group="_Hidden"
Sealed="TRUE" Version="0">
  <FieldRefs>
    <FieldRef ID="{983f490b-fc53-4820-9354-e8de646b4b82}" Name="PublishingCacheDisplayName"/>
    <FieldRef ID="{9550e77a-4d10-464f-bc0c-102d5b1aec42}" Name="PublishingCacheDisplayDescription"/>
    <FieldRef ID="{db03cb99-cf1e-40b8-adc7-913f7181dac3}" Name="PublishingCachePerformACLCheck"/>
    <FieldRef ID="{d8f18167-7cff-4c4e-bdbe-e7b0f01678f3}" Name="PublishingCacheEnabled"/>
    <FieldRef ID="{bdd1b3c3-18db-4acf-a963-e70ef4227fbc}" Name="PublishingCacheDuration"/>
    <FieldRef ID="{5b4d927c-d383-496b-bc79-1e61bd383019}" Name="PublishingCacheCheckForChanges"/>
    <FieldRef ID="{4689a812-320e-4623-aab9-10ad68941126}" Name="PublishingVaryByCustom"/>
    <FieldRef ID="{89587dfd-b9ca-4fae-8eb9-ba779e917d48}" Name="PublishingVaryByHeader"/>
    <FieldRef ID="{b8abfc64-c2bd-4c88-8cef-b040c1b9d8c0}" Name="PublishingVaryByParam"/>
    <FieldRef ID="{d4a6af1d-c6d7-4045-8def-cefa25b9ec30}" Name="PublishingVaryByRights"/>
    <FieldRef ID="{18f165be-6285-4a57-b3ab-4e9f913d299f}" Name="PublishingCacheability"/>
    <FieldRef ID="{0A90B5E8-185A-4dec-BF3C-E60AAE08373F}" Name="PublishingAuthenticatedUse"/>
    <FieldRef ID="{773ED051-58DB-4ff2-879B-08B21AB001E0}" Name="PublishingCacheAllowWriters"/>
  </FieldRefs>
</ContentType>

<ContentType
ID="0x010100C568DB52D9D0A14D9B2FDCC96666E9F2007948130EC3DB064584E219954237AF390064DEA0F50FC8C
147B0B6EA0636C4A7D4" Name="$Resources:cmscore,contenttype_welcomepage_name;"
Description="$Resources:cmscore,contenttype_welcomepage_description;"
Group="$Resources:cmscore,group_pagelayoutcontenttypes;" Sealed="FALSE" Version="0">
  <FieldRefs>
    <FieldRef ID="{3de94b06-4120-41a5-b907-88773e493458}" Name="PublishingPageImage"/>
    <FieldRef ID="{F55C4D88-1F2E-4ad9-AAA8-819AF4EE7EE8}" Name="PublishingPageContent"/>
    <FieldRef ID="{B3525EFE-59B5-4f0f-B1E4-6E26CB6EF6AA}" Name="SummaryLinks"/>
    <FieldRef ID="{27761311-936A-40ba-80CD-CA5E7A540A36}" Name="SummaryLinks2"/>
  </FieldRefs>
  <DocumentTemplate TargetName="/_layouts/CreatePage.aspx" />
</ContentType>

<ContentType
ID="0x010100C568DB52D9D0A14D9B2FDCC96666E9F2007948130EC3DB064584E219954237AF3900242457EFB8B24
247815D688C526CD44D" Group="$Resources:cmscore,group_pagelayoutcontenttypes;"
Name="$Resources:cmscore,contenttype_articlepage_name;"
Description="$Resources:cmscore,contenttype_articlepage_description;"
  <FieldRefs>
    <FieldRef ID="{3de94b06-4120-41a5-b907-88773e493458}" Name="PublishingPageImage"/>
    <FieldRef ID="{F55C4D88-1F2E-4ad9-AAA8-819AF4EE7EE8}" Name="PublishingPageContent"/>
    <FieldRef ID="{B3525EFE-59B5-4f0f-B1E4-6E26CB6EF6AA}" Name="SummaryLinks"/>
    <FieldRef ID="{D3429CC9-ADC4-439b-84A8-5679070F84CB}" Name="ArticleByLine"/>
  </FieldRefs>

```

```

        <FieldRef ID="{71316CEA-40A0-49f3-8659-F0CEFDDBD4F}" Name="ArticleStartDate"/>
        <FieldRef ID="{66F500E9-7955-49ab-ABB1-663621727D10}"
Name="PublishingImageCaption"/>
        <FieldRef ID="{a932ec3f-94c1-48b1-b6dc-41aaa6eb7e54}"
Name="HeaderStyleDefinitions"/>
    </FieldRefs>
    <DocumentTemplate TargetName="/_layouts/CreatePage.aspx" />
</ContentType>
<!-- Reusable Text Content Type -->
<ContentType ID="0x01004D5A79BAFA4A4576B79C56FF3D0D662D"
Name="$Resources:cmscore,contenttype_reusablelehtml_name;"
Description="$Resources:cmscore,contenttype_reusablelehtml_name;" Group="_Hidden"
Sealed="TRUE" Version="0">
    <FieldRefs>
        <FieldRef ID="{82642ec8-ef9b-478f-acf9-31f7d45fbc31}" Name="LinkTitle"/>
        <FieldRef ID="{9da97a8a-1da5-4a77-98d3-4bc10456e700}" Name="Comments"/>
        <FieldRef ID="{3a4b7f98-8d14-4800-8bf5-9ad1dd6a82ee}" Name="ContentCategory"/>
        <FieldRef ID="{e977ed93-da24-4fcc-b77d-ac34eea7288f}" Name="AutomaticUpdate"/>
        <FieldRef ID="{890e9d41-5a0e-4988-87bf-0fb9d80f60df}" Name="ReusableText"/>
    </FieldRefs>
</ContentType>

<ContentType ID="0x01002CF74A4DAE39480396EEA7A4BA2BE5FB"
Name="$Resources:cmscore,contenttype_reusablelehtml_name;"
Description="$Resources:cmscore,contenttype_reusablelehtml_name;" Group="_Hidden" Sealed="TRUE"
Version="0">
    <FieldRefs>
        <FieldRef ID="{82642ec8-ef9b-478f-acf9-31f7d45fbc31}" Name="LinkTitle"
Required="TRUE"/>
        <FieldRef ID="{9da97a8a-1da5-4a77-98d3-4bc10456e700}" Name="Comments"/>
        <FieldRef ID="{3a4b7f98-8d14-4800-8bf5-9ad1dd6a82ee}" Name="ContentCategory"/>
        <FieldRef ID="{e977ed93-da24-4fcc-b77d-ac34eea7288f}" Name="AutomaticUpdate"/>
        <FieldRef ID="{82dd22bf-433e-4260-b26e-5b8360dd9105}" Name="ReusableHTML"/>
    </FieldRefs>
</ContentType>

    <!-- END Reusable Text Content Type -->
    <!-- Redirect Content Type -->
<ContentType ID=
"0x010100C568DB52D9D0A14D9B2FDCC96666E9F2007948130EC3DB064584E219954237AF3900FD0E870BA0694887
9DBD5F9813CD8799" Name="$Resources:cmscore,contenttype_redirectpage_name;"
Description="$Resources:cmscore,contenttype_redirectpage_description;"
Group="$Resources:cmscore,group_pagelayoutcontenttypes;" Sealed="FALSE" Version="0">
    <FieldRefs>
        <FieldRef ID="{AC57186E-E90B-4711-A038-B6C6A62A57DC}" Name="RedirectURL"/>
        <FieldRef ID="{543BC2CF-1F30-488e-8F25-6FE3B689D9AC}"
Name="PublishingRollupImage"/>
    </FieldRefs>
    <DocumentTemplate TargetName="/_layouts/CreatePage.aspx" />
</ContentType>

<ContentTypeBinding
ContentTypeId="0x01010007FF3E057FA8AB4AA42FCB67B453FFC100E214EEE741181F4E9F7ACC43278EE811"
ListUrl=" catalogs/masterpage" />

```

Build Verification Tests in the Partner Portal Application

Build verification tests (BVTs) provide a regular quality check from the perspective of end user functionality. In most teams BVTs run on a regular schedule, such as once per day. Build verification tests are a combination of integration tests and user interface tests. The Partner Portal application includes BVTs that use the Visual Studio Team System continuous testing system. The test builds and packages the solution, installs the solution on a SharePoint server, and executes a series of tests. These actions occur through a set of agents installed on several clients.

The following table lists the BVTs the Partner Portal application uses. They can be found in the Test/PartnerPortal directory.

S.No	Test cases	Description
1	_VerifyCreateIncident_1	This test creates Incident 1 and navigates to the Partner 1 Incident page.
2	_VerifyCreateIncident_2	This test creates Incident 2 and navigates to the Partner 1 Incident page.
3	_VerifyCreateIncident_3	This test creates Incident 3 and navigates to the Partner 2 Incident page.
4	_VerifyCreateIncident_4	This test creates Incident 4 and navigates to the Partner 2 Incident page.
5	VerifyIncidentsPagePartner1	This test verifies the Incidents page of Partner 1.
6	VerifyIncidentsPagePartner2	This test verifies the Incidents page of Partner 2.
7	_VerifyCreateOrderExceptionPartner1	This test verifies the creation of an order exception for Partner 1.
8	_VerifyCreateOrderExceptionPartner2	This test verifies the creation of an order exception for Partner 2.
9	VerifyCollapsibleControlBusinessEventTypeConfiguration	This test verifies the text in the collapsible control in the Business Event Type Configuration page.
10	VerifyCollapsibleControlExtranetProductCatalog	This test verifies the text in the collapsible control in the Product Catalog page in the extranet port.
11	VerifyCollapsibleControlExtranetProductPage	This test verifies the text in the collapsible control in the Product page in the extranet port.
12	VerifyCollapsibleControlPartner1HomePage	This test verifies the text in the collapsible control in the Partner 1 Home page.
13	VerifyCollapsibleControlPartner1IncidentPage	This test verifies the text in the collapsible control in the Partner 1 Incident page
14	VerifyCollapsibleControlPartner1OrderExceptionPage	This test verifies the text in the collapsible control in the Partner 1 Order Exception page.
15	VerifyCollapsibleControlPartner2HomePage	This test verifies the text in the collapsible control in the Partner 2 Home page.
16	VerifyCollapsibleControlPartner2IncidentPage	This test verifies the text in the collapsible control in the Partner 2 Incident page.
17	VerifyCollapsibleControlPartner2OrderExceptionPage	This test verifies the text in the collapsible control in the Partner 2 Order Exception page.
18	VerifyCollapsibleControlPartnerCentral	This test verifies the text in the collapsible control in the Partner Central page.
19	VerifyCollapsibleControlPartnerDirectory	This test verifies the text in the collapsible control in the Partner Directory page.
20	VerifyCollapsibleControlPartnerSites	This test verifies the text in the collapsible control in the Partner Sites page.
21	VerifyCollapsibleControlProductCatalog	This test verifies the text in the collapsible control in the Product Catalog page.

22	VerifyCollapsibleControlProductPage	This test verifies the text in the collapsible control in the Product page.
23	VerifyCollapsibleControlPromoPage	This test verifies the text in the collapsible control in the Promotion page.
24	VerifyCollapsibleControlSubSiteCreationRequests	This test verifies the text in the collapsible control in the Sub Site Creation Requests page.
25	VerifyCollapsibleSPGSubSite	This test verifies the text in the collapsible control in the SPG subsite page.
26	verifycustomeventsources	This test verifies that the custom event source is logging errors from the Partner Portal application.
27	VerifyExtranetContosoPartner2User1	This test verifies the logon of ContosoPartner2User1 in the extranet port.
28	VerifyExtranetContosoPartner1User1	This test verifies the logon of ContosoPartner1User1 in the extranet port.
29	VerifyBusinessEventTypeConfiguration	This test verifies the Business Event Type Configuration page in the Partner Central page.
30	VerifyPartnerCentral	This test verifies the Partner Central page.
31	verifyPartnerSiteDirectoryPage	This test verifies the Partner Site Directory page in Partner Central.
32	verifySPGSubsitePage	This test verifies the SPGSUBSITE page in Partner Central.
33	VerifySubSiteCreationRequests	This test verifies the Sub Site Creation Requests page in Partner Central.
34	VerifyProductCatalog_0	This test verifies the Web Parts in the Product Catalog page with the category ID 0 .
35	VerifyProductCatalog_1	This test verifies the Web Parts in the Product Catalog page with the category ID 1 .
36	VerifyProductCatalog_2	This test verifies the Web Parts in the Product Catalog page with the category ID 2 and verifies the details in the productList Web Part in the Product Catalog page with the category ID 8 .
37	VerifyProductCatalog_3	This test verifies the Web Parts in the Product Catalog page with the category ID 2 and verifies the details in the Product List Web Part in the Product Catalog page with the category ID 12 .
38	verifyProductDetail	This test verifies the Product Details page of the product with the SKU 6000000000 .
39	VerifyExtranetProductCatalog_0	This test verifies the Web Parts in the Product Catalog page with the category ID 0 for the extranet user.
40	VerifyExtranetProductCatalog_1	This test verifies the Web Parts in the Product Catalog page with the category ID 1 for the extranet user.
41	VerifyExtranetProductCatalog_2	This test verifies the Web Parts in the Product Catalog page with the category ID 2 for the extranet user.
42	VerifyExtranetProductCatalog_4	This test verifies the Web Parts in the Product Catalog page with the category ID 4 and verifies the details in the Product List Web Part in the Product Catalog page with the category ID 15 for the extranet user.
43	verifyProductDetail_FBA	This test verifies the Product Details page of the product with the SKU 6000000000 for the extranet user.
44	VerifyPartner1HomePage	This test verifies the Partner 1 Home page.
45	VerifyPartner2HomePage	This test verifies the Partner 2 Home page.

46	VerifyPartner1HomePage_FBA	This test verifies the Partner1 Home page for the extranet user.
47	VerifyPartner2HomePage_FBA	This test verifies the Partner 2 Home page for the extranet user.
48	VerifyIncidentManagementServicePage	This test verifies whether the incident management service was running correctly.
49	VerifyIncidentSiteServicePage	This test verifies whether the incident site service was running correctly.
50	VerifynewincidentPage	This test verifies the New Incident Creation page.
51	VerifyPricingServicePage	This test verifies whether the pricing service service was running correctly.
52	VerifyProductCatalogServicePage	This test verifies whether the Product catalog service was running correctly.
53	Z_VerifyExtranetOrderExceptionPartner1	This test verifies the order exception site created by Partner 1 in the extranet port using forms-based authentication (FBA) credentials.
54	Z_VerifyExtranetOrderExceptionPartner2	This test verifies the order exception site created by Partner 2 in the extranet port using FBA credentials.
55	Z_VerifyOrderExceptionSitePartner1	This test verifies the order exception site created by Partner 1.
56	Z_VerifyOrderExceptionSitePartner2	This test verifies the order exception site created by Partner 2.
57	Z_VerifyPartner1Incidents	This test verifies the Incident 1 and Incident 2 sites created by Partner 1.
58	Z_VerifyPartner2Incidents	This test verifies the Incident 3 and Incident 4 sites created by Partner 2.
59	VerifyPartnerHomeContosoPartner1User6	This test verifies the Partner 1 Home page for the Partner 1 Windows user.
60	VerifyPartnerHomeContosoPartner2User6	This test verifies the Partner 2 Home page for the Partner 2 Windows user.
61	VerifyProductCatalogContosoPartner1User6	This test case verifies the Web Parts in the Product Catalog page with the category ID 0 for the Partner 1 Windows user.
62	VerifyProductCatalogContosoPartner2User6	This test verifies the Web Parts in the Product Catalog page with the category ID 0 for the Partner 2 Windows user.
63	VerifyProductPageContosoPartner1User6	This test verifies the Product Details page of the product with the SKU 6000000000 for the Partner 1 Windows user.
64	VerifyProductPageContosoPartner2User6	This test verifies the Product Details page of the product with the SKU 6000000000 for the Partner 2 Windows user.
65	VerifySubSiteCreationServicePage	This test verifies the subsite creation page.