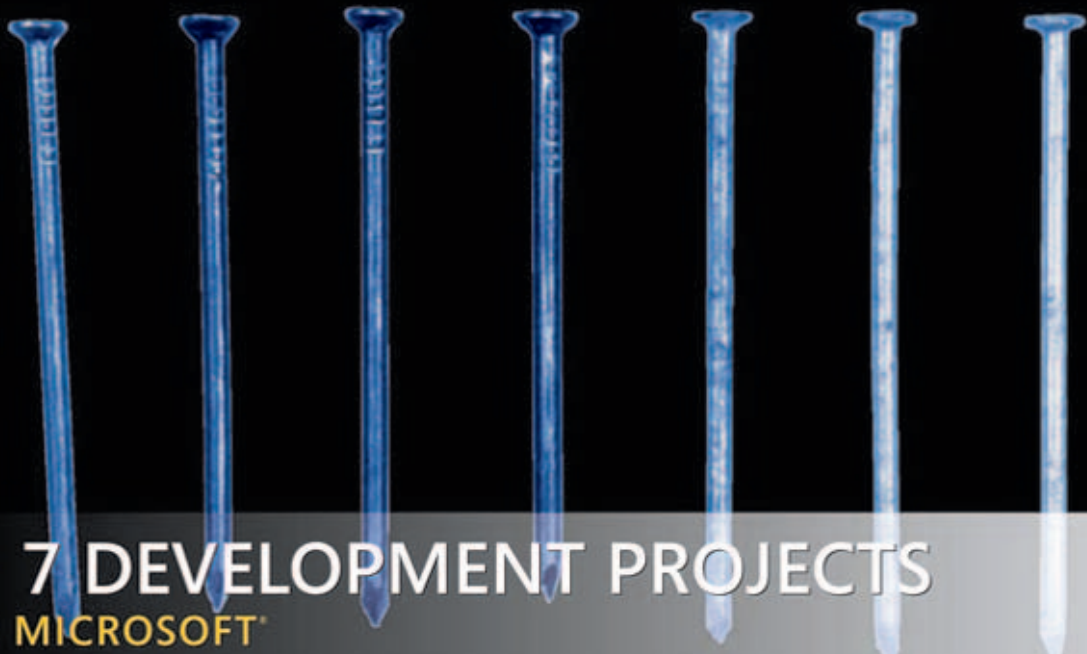


Microsoft

Special TechEd 2006 Edition
Not for Resale

Based on
May 2006
Pre-release
Code



7 DEVELOPMENT PROJECTS
FOR MICROSOFT®
OFFICE SHAREPOINT® SERVER 2007
AND WINDOWS® SHAREPOINT
SERVICES VERSION 3.0

Microsoft Corporation

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright © 2006 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QWE 1 0 9 8 7 6

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Microsoft, Active Directory, ActiveX, Excel, FrontPage, InfoPath, Internet Explorer, JScript, Microsoft Dynamics, Microsoft Press, MSDN, Outlook, PivotChart, PivotTable, PowerPoint, SharePoint, Visual Basic, Visual C#, Visual Studio, Windows, Windows Server, and WinFX are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

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

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Acquisitions Editor: Ben Ryan

Project Editor: Melissa von Tschudi-Sutton

Production: Publishing.com

Body Part No. X12-23639

Contents at a Glance

1	Microsoft Windows SharePoint Services 3.0	1
2	Building Solutions with Office SharePoint Server 2007	21
3	Building a Basic SharePoint Site.	41
4	Organizing Lists and Documents with Site Columns and Content Types.	57
5	Working with Features in Windows SharePoint Services.	77
6	Windows SharePoint Services Core Development	101
7	Creating Workflows: The Missing Piece of Office Productivity	121
8	Introducing Excel Services	143
9	Microsoft Office InfoPath 2007 and Microsoft Office Forms Server 2007	183



Table of Contents

Acknowledgments	xi
Introduction	xiii
Solution Showcase for the 2007 Microsoft Office System	xiv
Accruent	xv
AVIVA Consulting Group	xvi
CorasWorks	xviii
KnowledgeLake	xix
Metalogix Software	xxi
OSISoft	xxiv
Quilogy	xxvi
Resolute	xxvii
3Sharp	xxx
Microsoft Office Developer Center	xxx
1 Microsoft Windows SharePoint Services 3.0	1
Integration with ASP.NET 2.0	2
Working with Master Pages	5
Web Parts in Windows SharePoint Services 3.0	8
Developing Custom Web Parts	11
Enhancements in Content Storage	13
Event Handlers	15
Workflows in Windows SharePoint Services 3.0	16
Site Definitions, Features, and Solutions	17
Internet-Style Security	19
Summary	19
2 Building Solutions with Office SharePoint Server 2007	21
Building Office SharePoint Server 2007 Portal Sites	23
Shared Service Providers	24
User Profiles	25
Office SharePoint Server 2007 Search	26
The Business Data Catalog	28
Web Content Management	30
Business Intelligence Features	35

Managing Documents and Business Processes	37
Office Forms Server 2007	38
Enterprise Content Management	39
The Single Sign-On Service	39
Summary	40
3 Building a Basic SharePoint Site	41
Creating a Site Collection and a Top-Level Site	41
Creating a List for Tracking Project Profiles	46
Creating a Document Library	48
Customizing the Home Page	49
Creating Child Sites	51
Creating a "Hello World" Web Part	52
4 Organizing Lists and Documents with Site Columns and Content Types	57
Content Types	57
Content Type Settings	58
File Formats	59
Site and List Content Types	59
Creating Content Types Based on Other Content Types	60
Controlling Changes to Content Types	61
Controlling Access to Content Types	62
Updating Content Types	62
Extending Content Types	65
Site Columns	66
Site Column Properties	67
Working with Site Columns and Content Types	67
Creating a Site Column for Project Lookups	68
Creating Custom Content Types	71
5 Working with Features in Windows SharePoint Services	77
Implementing Features	78
Feature Elements	78
Element Scope	79
Activation Dependencies and Scope	79
The Structure of Feature.xml	80

Features and the Windows SharePoint Services Object Model	82
Feature Classes	82
Accessing Feature Collections	83
Features and Events	84
Designing Windows SharePoint Services Applications Using Features	84
Activating and Deactivating Features	84
Working with a Custom Feature	86
Creating a Custom Feature	90
Creating a Callback Receiver Class for a Feature	93
Adding a Document Library on Activation	95
Adding an Event Handler to the Timesheets List	96
6 Windows SharePoint Services Core Development	101
Top-Level Classes	101
Updating Object Properties	103
Working with Collections	104
Using Indexers	105
Using the Windows SharePoint Services Object Model and the Data in a List	107
Generating Weekly Time Sheet Aggregate Views	107
Using the Packaging API to Stuff Weekly Time Sheet Aggregates into Word Documents	111
Adding Word Documents to a Document Library	114
Using an Event Handler to Generate Weekly Aggregate Documents	115
Adding the Assembly to the GAC	117
Registering the Event Handler Assembly	117
7 Creating Workflows: The Missing Piece of Office Productivity	121
Workflows and Activities	121
Windows Workflow Foundation Run-Time Engine	123
Building Custom Workflows	123
Installation and Deployment	125
Workflow Stages	125
Workflow Association	125
Workflow Initiation	126
Workflow Status	126
Workflow Task Completion	126
The OnWorkflowActivated Activity	127

Workflows in Action	128
Associating and Activating SharePoint Workflows	129
Creating a Human Workflow in Visual Studio 2005	134
8 Introducing Excel Services	143
Key Scenarios for Excel Services	144
Sharing Workbooks Through a Browser	144
Building Business Intelligence Dashboards	145
Reusing the Logic Encapsulated in Excel Workbooks	147
Excel Services Architecture Overview	148
Security	148
Performance and Scalability	148
Controlling Visible Information and Interacting with Workbooks	149
Defining Parameters	149
Interacting with Workbooks in the Browser	150
Building Applications with Excel Web Services	151
Error Handling	153
Sessions	153
Controlling and Protecting Workbooks	154
The View Item Right	154
Controlling Who Can Publish Workbooks to Excel Services	155
Controlling the Publishing Process for Workbooks in Excel Services	155
Data Connection Libraries	156
What Is a Data Connection Library?	156
Connecting to Databases Made Easy	157
Solving Connection Management Problems	158
Making Data Connectivity More Secure	159
Unsupported Features in Excel Services	159
Excel Services and Reporting in a Portal	162
Adding a Trusted Location and a Trusted Data Connection Library to an Excel Services Configuration	162
Building a Report for Excel Web Access	165
Coding with Excel Web Services	172
Monte Carlo Simulation	172
A Simple Mortgage Calculator	176
Excel Services User-Defined Functions	178

9	Microsoft Office InfoPath 2007 and Microsoft Office Forms Server 2007	183
	Key Scenarios	184
	Components and Architecture of Office Forms Server 2007	185
	Client Architecture and Postback Optimization	186
	The Form Template Converter	186
	Data Connections and Office Forms Server 2007	186
	Designing Form Templates	187
	Controls Supported on Office Forms Server 2007	187
	Controls Not Supported on Office Forms Server 2007	188
	Browser Support	188
	Controlling Postback Behavior When Control Values Change	189
	Deployment	189
	Security	190
	Custom ASP.NET Pages with the InfoPath Form Services Control	190
	Automating Office Forms Server 2007 Administration Tasks	191
	Working with InfoPath Forms and Forms Server	193
	Creating an InfoPath Form to Capture Time Sheet Data	193
	Publishing a Form to a Windows SharePoint Services Forms Library	200
	Embedding an InfoPath Form in an Application	201



Acknowledgments

Many people had a hand in putting this book together. Mike Fitzmaurice, of the Office Servers team, initiated the project. Ted Pattison wrote Chapters 1 and 2, and John Pierce, a technical writer with Microsoft's Information Worker Adoption Group, wrote and compiled the information presented in Chapters 3 through 9.

Ben Ryan and Melissa von Tschudi-Sutton of Microsoft Press guided the book through publication. Curt Philips managed the team that edited and produced the book, and Andrea Fox was the copy editor.

Members of the Excel Services team, including Jeff Wierer, Eran Megiddo, and David Gainer, provided material and reviews for the chapter about Excel Services. Dave Webster contributed to the chapter about Windows SharePoint Services workflows. Thanks also to the teams that put together the material for the beta versions of the Microsoft Windows SharePoint Services 3.0 SDK and the Microsoft Office SharePoint Server 2007 SDK.

Finally, thanks go to Rob Barker, Joel Frauenheim, Garry Gross, and other members of the Microsoft Office Solution Showcase team for their support of this project.

Introduction

Ahead of the release of the 2007 Microsoft Office system, Microsoft Corporation has frequently referred to what it calls the “new world of work”—a business environment and economy characterized by mobility; a worldwide network of customers, partners, and suppliers; new compliance and regulatory requirements; and a need for broad visibility into business processes and the information that supports and governs them. Together, factors such as these drive the need for organizations to integrate business applications, documents, and workflows and transform the content of documents into business information they can act on. These factors make plain the essential link between knowledge workers, business information, business processes, and software.

The 2007 Office system has much that’s new, including the Office XML file formats, an ability to more easily link to back-end data systems, and a new user interface. Each of these features provides opportunities for developers to extend Office applications. For example, the support of XML throughout the 2007 Office system provides a means to define information workflows and document management capabilities. An organization’s IT staff can integrate systems such as SAP, Siebel, or Microsoft Dynamics with Office applications so that workers can perform operations such as invoicing or ordering when working in Office rather than the back-end system itself.

Seven Development Projects for Microsoft Office SharePoint Server 2007 and Windows SharePoint Services Version 3.0 focuses on another, just as critical, element of Office development—Microsoft Windows SharePoint Services and Microsoft Office SharePoint Server 2007, platforms that developers can use to create collaboration applications as well as applications and features that support business intelligence, workflow, data calculation, team workspaces, document life cycle management, content management, knowledge discovery, and project management.

Information workers use an array of tools and devices to perform their jobs and collaborate with others: e-mail, desktop applications, mobile devices, Web browsers, Web conferencing, portals, and specialized line-of-business applications. Individual workers, project teams, and departments all need the capabilities provided by collaborative workspaces and the ability to archive and access information as projects are completed and new ones begin. Windows SharePoint Services, an integrated part of the Microsoft Windows Server 2003 operating system, provides services that enable teams and developers to build such workspaces. Windows SharePoint Services 3.0 can be tightly integrated with the 2007 Office system, further enabling information workers to make use of workspaces for organizing meetings, managing projects, authoring documents, and other activities that they frequently manage from traditional Office applications. Windows SharePoint Services 3.0 can also be integrated with line-of-business applications, providing access to data that workers need to update and analyze.

Office SharePoint Server 2007 builds on the Windows SharePoint Services framework and provides services such as search in portal sites, team sites, and content management sites; user profiles and audience targeting; and single sign-on to facilitate integration with enterprise data systems. Office SharePoint Server 2007 also introduces the Business Data Catalog, which enables integration between enterprise portal and line-of-business applications; new document management capabilities; Web content management, which provides tools for site branding, creating multilingual sites, and building content deployment solutions; Microsoft Office SharePoint Server 2007 Excel Services, a technology for viewing, calculating, and extracting values from a workbook through a Web browser or through a Web service; and Microsoft Office Forms Server 2007, which lets information workers use a Web browser to interact with form templates designed with Microsoft Office InfoPath 2007.

The last seven chapters of this book describe some basic and straightforward examples of the functionality and capabilities that can be included in a solution developed for Windows SharePoint Services, Office SharePoint Server 2007, and the 2007 Microsoft Office system. These examples often show sample code, but be aware that the examples were developed and tested only on the beta 1 build of the 2007 Office system. You'll see a Web Part that displays data from a SharePoint list, for example, as well as a Web application that uses formulas in an Excel worksheet to perform calculations, a document approval workflow, and more. The examples are meant to show off core features and concepts and to help generate ideas for solutions that can be developed or deployed in your own organization.

Solution Showcase for the Office System

Developers, development managers, and organizational decision makers can consult the Solution Showcase for the Office System at <http://www.microsoft.com/office/showcase>, to find examples, evidence, and information about ways that organizations are transforming their businesses and solving business problems by implementing Microsoft Office solutions. The Solution Showcase includes information about business scenarios and solutions for a number of industries, including manufacturing, financial services, public sector and government agencies, retail operations, health care, and professional services.

Working with partners and customers, the Solution Showcase team identifies high-impact business scenarios that can be addressed with the Office System. The Solution Showcase provides information and demonstrations that document the technical and business value of Office solutions related to these scenarios and that help facilitate discussions with business leaders. Solutions developed by Microsoft technology partners are shown alongside the business scenarios.

In preparing for the release of the 2007 Office system, Microsoft worked with partners and customers to train and prepare them to develop solutions based on the 2007 Office system. To illustrate what you will soon see in the marketplace and the types of solutions developers will have a hand in building, the following sections describe examples of solutions that some of Microsoft's technology partners currently have under way.

Accruent

Accruent's Store Lifecycle Management (SLM) product is a suite of applications that is built on the .NET 2.0 Framework and uses Web Parts in its portal interface. The applications address the full spectrum of real estate activities: market planning, site selection, design and construction, lease administration, and facilities management. SLM consolidates department-based, manual processes into a single automated process for real estate operations. For example, SLM uses a location-centric data model to merge multiple versions of real estate information into a single, unified repository. SLM's financial engine can then analyze real estate performance to identify expense leakage and resource inefficiencies. Companies can execute store construction and remodel projects more quickly using program management templates, workflows, alerts, and scheduling capabilities to facilitate collaboration. Finally, controls are automatically instituted by a compliance system to monitor Sarbanes-Oxley compliance and generate accurate Financial Accounting Standards Boards (FASB) reports.

The functionality in Accruent's Desktop Connector offers connectivity to 2007 Office system applications such as Microsoft Office Outlook 2007, Microsoft Office Excel 2007, and Microsoft Office Word 2007. This connectivity integrates information from SLM applications with these commonly used programs, providing a natural flow of information and interaction to users where they need it. These applications help optimize the effect that real estate locations—stores, restaurants, and bank branches, for example—have on company performance by increasing revenue days and same-store sales, reducing occupancy and construction costs, achieving Sarbanes-Oxley compliance, and producing accurate FASB reports.

Figures 1 and 2 show examples of data integration between SLM and Office 2007 applications.

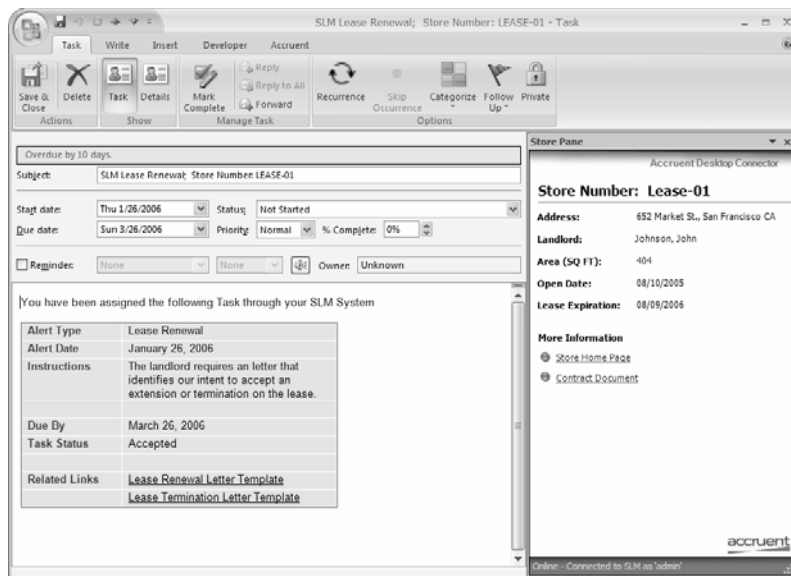


Figure 1 Integration with an Office Outlook 2007 task item. (© 2006 Accruent, Inc. All rights reserved.)

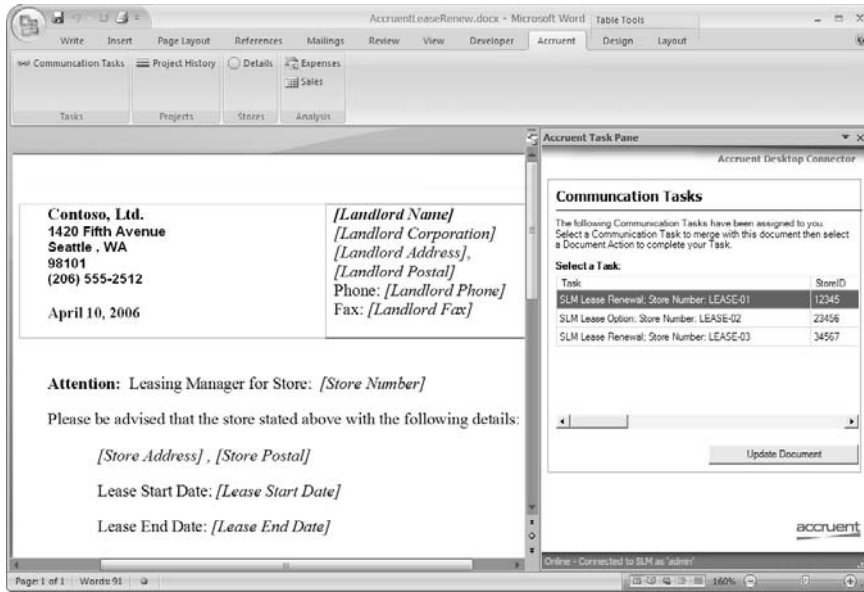


Figure 2 A customized task pane in Office Word 2007. (© 2006 Accruent, Inc. All rights reserved.)

AVIVA Consulting Group

The AVIVA Compliance Environment (ACE • S) is a solution for document management, process control, and executive visibility for Sarbanes-Oxley Section 302 and 404 compliance. The solution is based on a Web portal that uses Microsoft technologies to organize and manage internal control documentation in addition to evaluation and testing documents and results. ACE • S can be utilized by all designated professionals, including internal and external participants, so that all phases of a compliance program can be managed comprehensively. The solution is built on Windows SharePoint Services and its capabilities provided by products and technologies such as Microsoft Office InfoPath, ASP.NET 2.0, Microsoft SQL Server 2005 Reporting Services, Microsoft Information Bridge Framework 1.5, Office Word 2007, Office Excel 2007, and AVIVA Narrative Composer 2.1.

The solution's components provide a structured control framework that is based on COSO and the latest Public Company Accounting Oversight Board (PCAOB) recommendations and nomenclature. An organization can configure its own relationships between processes, control objectives, risks, controls, and so on, and use its own nomenclature if needed. In addition, through AVIVA's Narrative Composer, a Microsoft Word plug-in that provides a direct link from a document to a portal environment, the solution supports the creation of walkthrough documentation that provides context for each of the controls.

ACE•S manages the review process by ensuring that process owners periodically review internal control documentation for accuracy and completeness. The solution also provides an audit trail of all reviews and the date the review was performed. AVIVA uses SourceCode K2.net for advanced workflow capabilities that include the ability to ensure that policies and procedures conform to an organization's specifications.

Each user's main Web portal view is based on his or her role and contains lists of tasks, monitoring of relevant processes, and reports. ACE•S provides a comprehensive list of configured reports that provide visibility into the status of the compliance effort, including the Internal Control Matrix, the Deficiency Matrix, Assertion Coverage, and the Executive Summary. Each report is intended for a specific type of user, and users can create a subscription that automatically delivers critical reports at a specified interval.

Figure 3 shows a sample of the Executive Summary report. Figure 4 on the next page shows a Control Activity page.

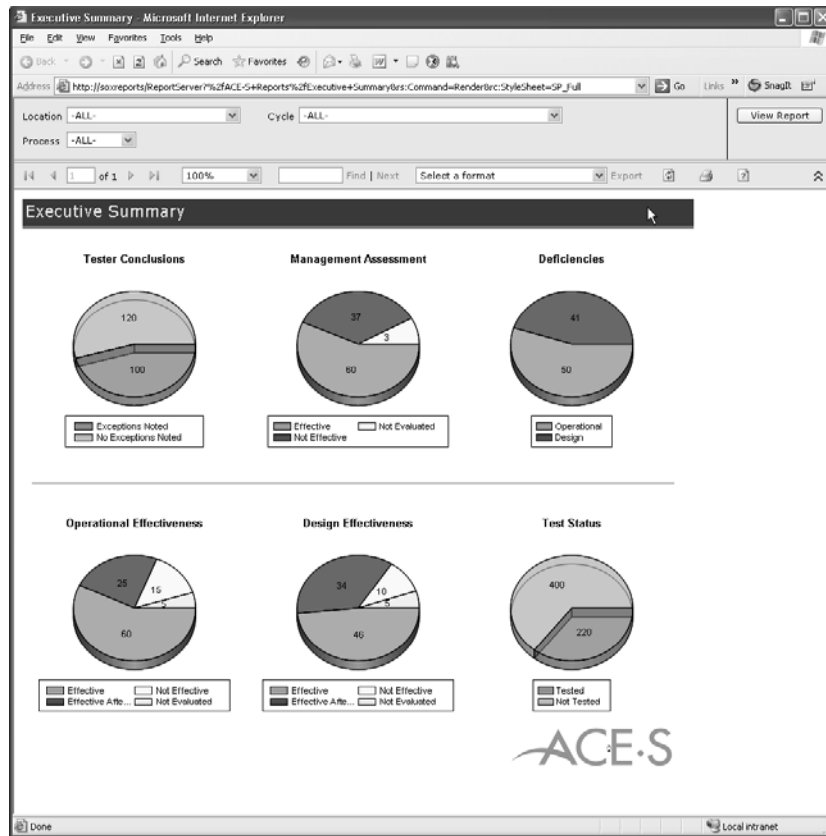


Figure 3 The Executive Summary report

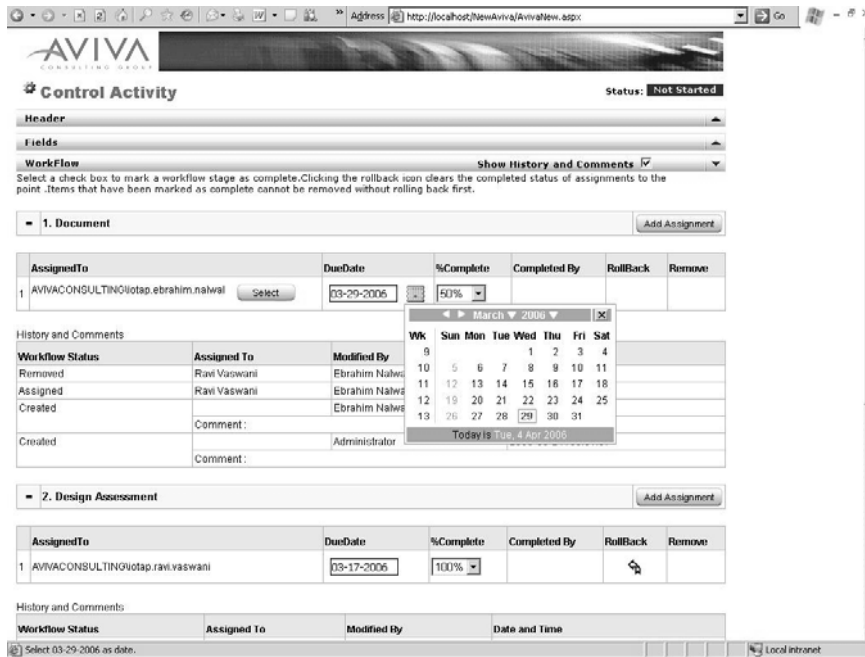


Figure 4 A Control Activity page used for managing compliance requirements

CorasWorks

CorasWorks is developing a solution named Workplace Suite for Office SharePoint Server 2007. The solution enables organizations to design, create, and manage a collaborative workplace of interconnected business applications without the time and expense of custom programming. The solution includes 40 business solutions that can be used as is or that can be easily extended to meet specific needs. For example, the Project Management solution can be configured for multiple small projects or a large project or to support an integrated, world-wide program management office.

Workplace Suite for Office SharePoint Server 2007 includes a set of “intelligent” modular components that can be snapped together to create flexible business applications and that integrate applications to create a complete workplace. The suite also includes the Workplace Configuration Manager, which is used to manage and control the behavior of the components, applications, interconnected solutions, and the workplace, including the integration of external data sources. In addition, the suite provides pre-built integration of Office SharePoint Server 2007 capabilities, including workflows, Office Forms Server 2007, enterprise content management, and the Business Data Catalog. Figure 5 shows an example of a workspace and the types of operations users can perform.

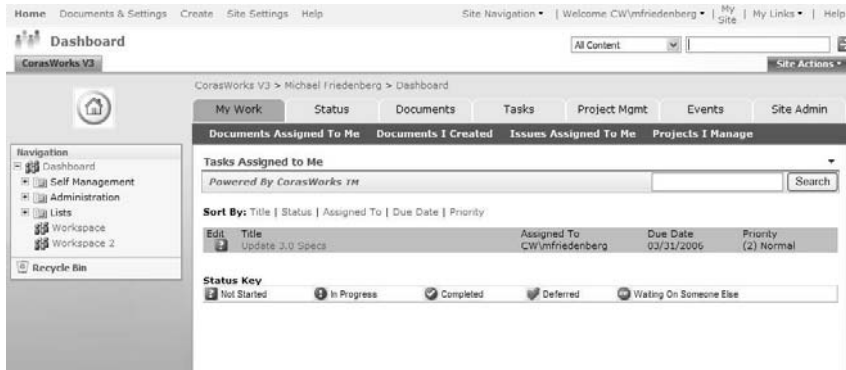


Figure 5 A status dashboard that's part of the Workplace Suite for Office SharePoint Server 2007

KnowledgeLake

KnowledgeLake extends the capabilities of Windows SharePoint Services as the core repository for an enterprise content management (ECM) solution. KnowledgeLake's ECM solution provides capture, imaging, and content-centric workflow that allow companies to manage millions of paper-based documents and unstructured information and streamline business workflow processes.

KnowledgeLake Capture is a production-level scanning solution, usually found in mailrooms, that allows users to scan, index, and export documents within a single user interface. KnowledgeLake Capture automatically exports images and indexing data to the SharePoint repository, where it queries SharePoint for the available libraries and content types. KnowledgeLake Capture is integrated with KnowledgeLake Imaging and features bar code and patch code recognition, image enhancement, and index validation. For the 2007 Office system release, KnowledgeLake Capture enables users to select content types directly from the Indexing screen and use optical character recognition (OCR) to apply metadata properties before saving scanned documents to a SharePoint site.

KnowledgeLake Imaging enhances the capabilities of SharePoint to enable easy organization, storage, and access of all unstructured business documents (paper, faxes, e-mail, computer reports, and so on). An integrated Web Part within the SharePoint interface allows users to search for documents by index values that are created and maintained as a SharePoint site feature. Users can even configure the search results by document type, date, or any other column.

The thin-client viewer allows users to view scanned documents from SharePoint using thumbnails, zooming, and rotating, with a user interface usually seen only in Windows Forms applications. This feature is made possible through AJAX technologies by allowing partial page postbacks. Users can also add annotations to the documents and save them back to SharePoint with the markup. Annotations, drawn completely on the client side using

JavaScript, include highlights, rectangles, text, stamps, straight lines, freehand lines, and sticky notes. Figures 6 and 7 show examples from the KnowledgeLake ECM solution.

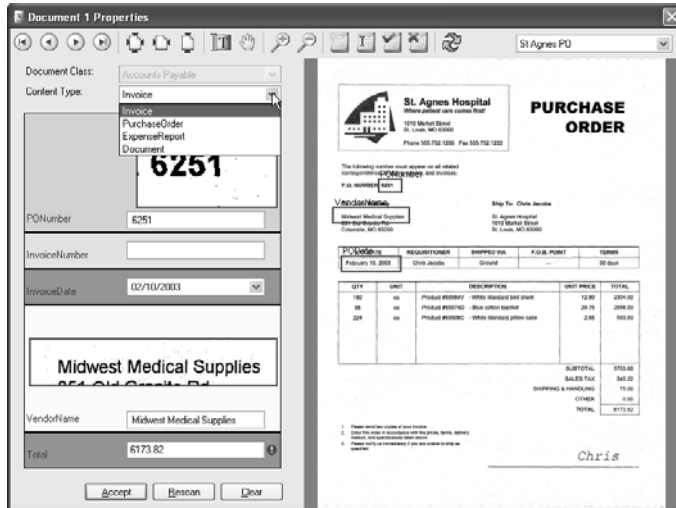


Figure 6 After a user clicks Accept, the image is uploaded to a SharePoint site. C#, SharePoint Web services, and the FrontPage Remote Procedure Call protocol allow this operation to occur.

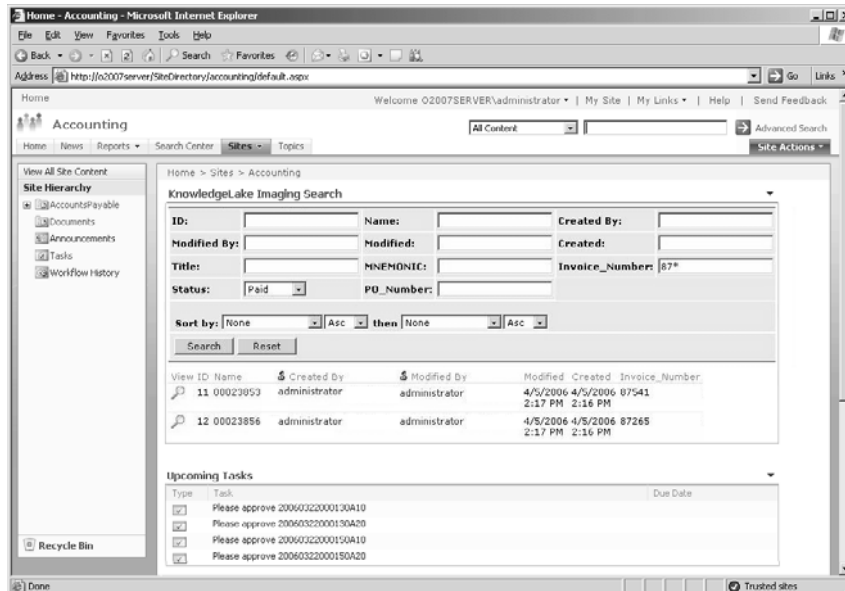


Figure 7 Once the search results are returned, the site feature adds an icon to view any image via KnowledgeLake Imaging's thin-client viewer (contains no ActiveX or Java).

SharePoint integration built into the viewer makes KnowledgeLake Imaging look and feel like an Office 2007 client application. Users can view and modify SharePoint properties (the

metadata added with KnowledgeLake Capture), as well as check the status and act on associated Office workflow items. Figure 8 shows the KnowledgeLake AJAX viewer.

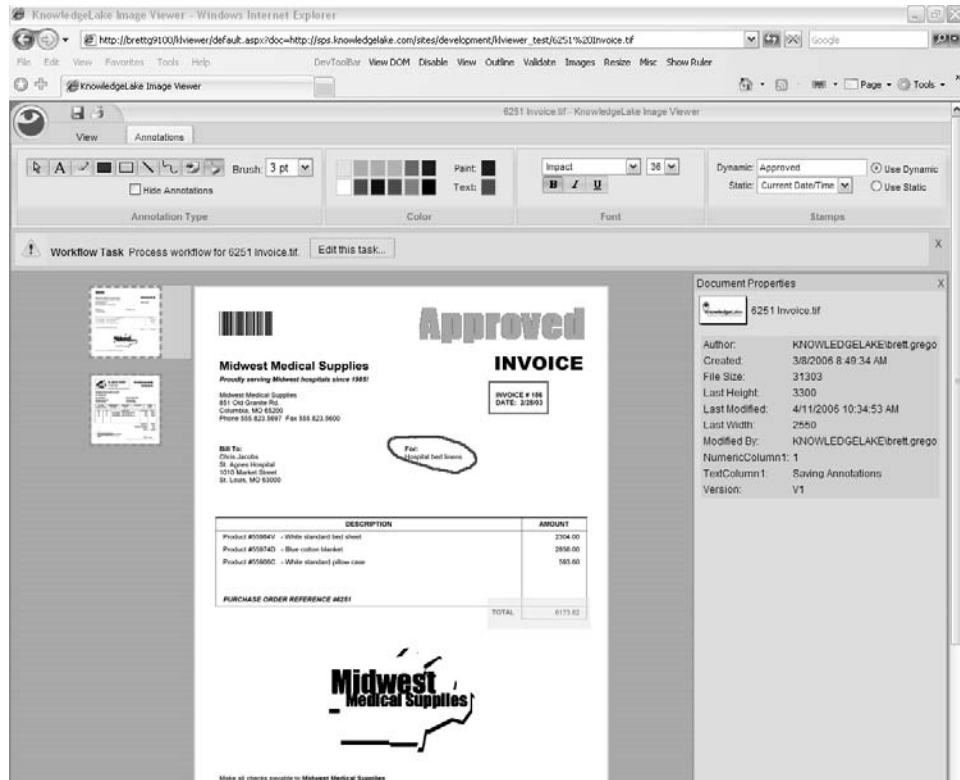


Figure 8 KnowledgeLake AJAX Image Viewer

Metalogix Software

Metalogix's Migration Manager is a content migration solution that enables users to discover, analyze, extract, and transform content from Web sites, file shares, existing SharePoint repositories, Microsoft Content Management Server 2002, and third-party enterprise content management systems. Information from these sources can be exported to SharePoint libraries and lists.

Migration Manager includes crawlers for discovering content on the file system or any HTTP-accessible Web site. These crawlers are compatible with enterprise content management systems such as Vignette, Interwoven, IBM, Tridion, Stellent, and Microsoft Content Management Server 2002. Static sites are also supported in addition to many other third-party systems. Migration Manager also helps organizations analyze content assets by providing a metadata explorer and filtering tools. Users can determine what type of content exists (.html, .doc, .pdf, or other file formats), when it was created, how frequently it is accessed and

modified, and who authored it. Migration Manager lets users add metadata of any data type to multiple content items at the same time. After the metadata is defined, it can be easily mapped to SharePoint library columns. Web content, documents, and metadata can be exported to any Office SharePoint Server 2007 site that is accessible through HTTP. Content in HTML or XML format can be mapped to Office SharePoint Server page fields or to Web Part pages, and metadata can be mapped to existing columns or new columns can be created automatically.

Figures 9, 10, and 11 illustrate operations of Migration Manager.

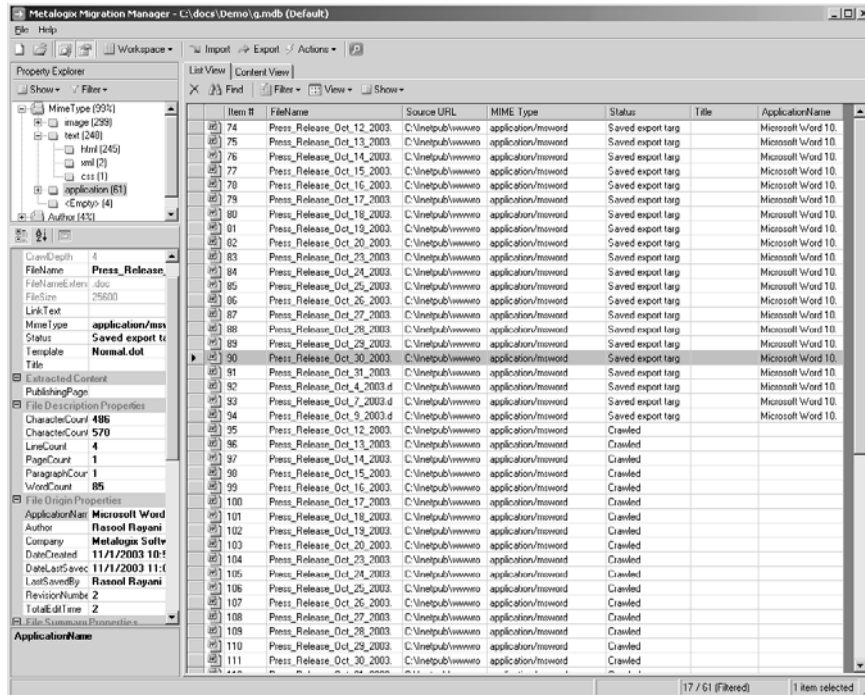


Figure 9 The main application window for Migration Manager. A row is added to the List View for each content item added. The Property sheet lets users view and edit metadata associated with a particular content item. The Property Explorer filters and organizes the list.

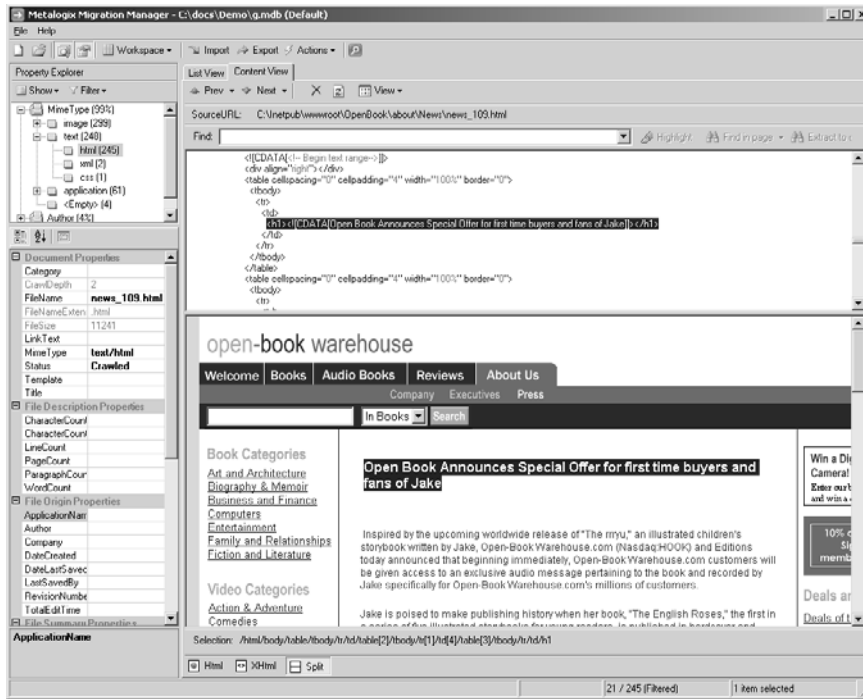


Figure 10 This split view enables a user to analyze an HTML or XML document selected in the List View. The upper pane shows HTML source code and the lower pane shows the rendered HTML. This view enables users to visually tag HTML and XML content and run XPath queries to extract content from within an HTML or XML document.

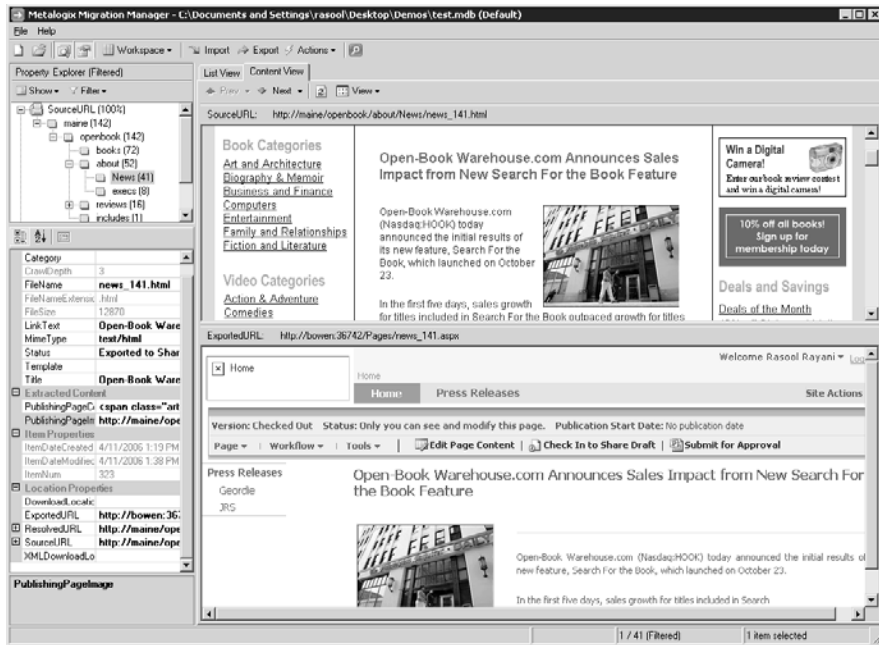


Figure 11 This split view enables users to view source content in the upper pane and exported content in the lower pane. This example shows an HTML page above a newly created page in an Office SharePoint Server 2007 pages library.

OSISoft

OSISoft has defined an Excel Services solution that includes a set of user-defined functions that retrieve time-series data from a proprietary database. The functions are defined for an Excel workbook, which can be rendered in a browser using Excel Web Access. By defining cells that act as parameters, a user can change the value of a cell when the workbook is rendered in the browser, and the solution can retrieve the data for any time period. For example, if one of the user-defined functions displayed an irregular trend in the data being analyzed, users could focus on that time period to find out the cause of the irregularity.

Figure 12 shows the workbook rendered in Excel Web Access with a new value in the Parameters list. Figure 13 shows the workbook in the browser with the updated data.

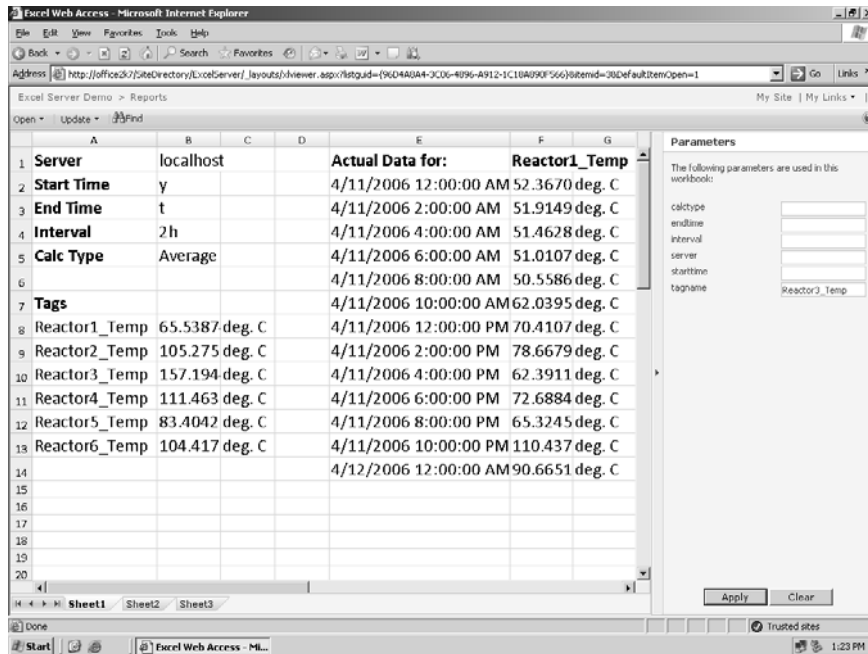


Figure 12 An Excel workbook rendered in Excel Web Access. Data can be updated based on values entered in the Parameters list.

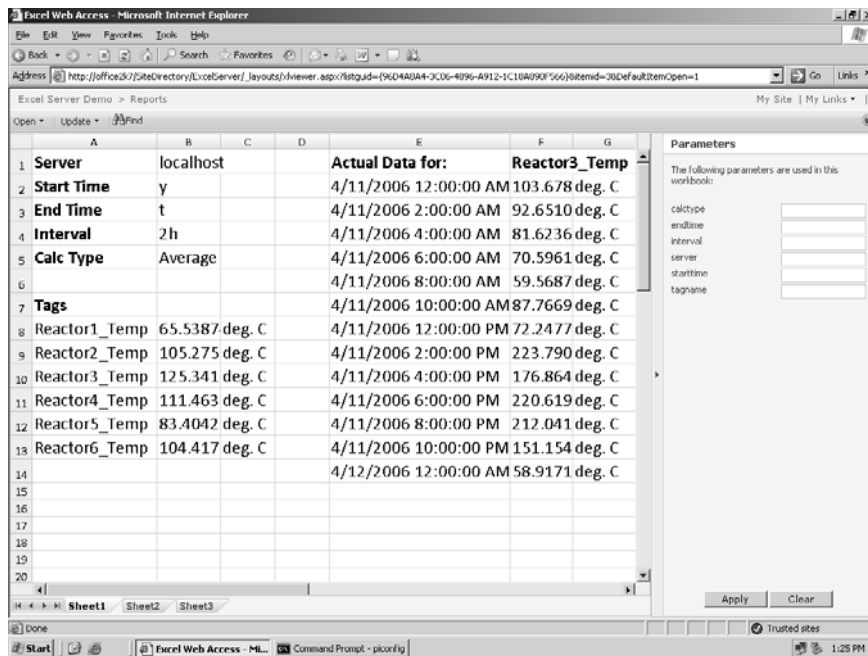


Figure 13 The updated data

Quilogy

Quilogy performed a proof of concept using the 2007 Office system for a large energy client. The customer had an inefficient paper and document-based process for bringing on contract staff. Quilogy and Microsoft helped them envision a new process that leverages the 2007 Office system. Office InfoPath 2007 forms, routed through an Office SharePoint Server 2007 workflow, automate the contractor on-boarding process—providing new speed, control, and accountability to this process. Office Word 2007 documents, such as the Contractor Background Investigation form, can be securely incorporated into the workflow process using Rights Management Services to provide strict control over document information. Workflow tasks can arrive in an Office Outlook 2007 e-mail message, show up on the Office Outlook 2007 To-Do Bar, and be completed offline. Managers can use Office Excel 2007 to produce visualizations of the workflow process and its metrics to easily identify bottlenecks and throughput.

Figure 14 shows the workflow process on a SharePoint site. Figure 15 shows an Office InfoPath 2007 form template rendered in Internet Explorer.

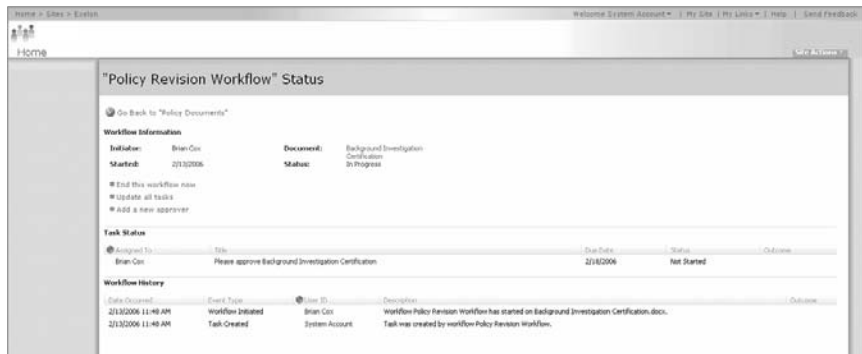


Figure 14 A workflow process on a SharePoint site

Hiring Manager View

Contractor Add Request

Job Code: 6003 - Outsourced Labor | Geographical Location: West

Supply Contract Number: 123 | Release Number: 123 | Contract Start Date: 2/11/06 | Contract End Date: 2/18/06

Contractor Status: Prior Contractor Prior Employee Retiree

Contractor Information

Contractor Name: Terry Earls | Employer/Vendor Name: Acme Professional Services

Gender: Male | Social Security Number: 123-45-6789 | Background Check: 2/6/06 | Back Ground Exemption:

Work Phone: 123-456-7890 | Home Phone: 555-555-5555 | Home State: IDAHO

Employee Class: R - Contractor | Pay Group: PGC | Full Time/Part Time: Full Time

Approval

Requestor Name: Luis Bonifaz | Requestor Work Phone: 555-123-4567

Requestor Email: LuisB@litwareinc.com | Requestor Submit Date: 2/10/2006

Manager Name: Brian Cox | Manager Work Phone: 555-123-4569

Manager Email: BrianC@litwareinc.com | Manager Employee ID: 152-110

Submit

Background Investigation Certification.docx
Microsoft Office Word document
10.7 KB

Infopath

- Intuitive, easy to use electronic forms
- Structure and business rules for collecting, using and integrating data to other systems or applications
- Eliminate Job Aids & Instructions by building logic and intelligence into form
- Required Fields(*) ensure form completion

<- Date Picker - enforces timestamp and/or common date format by end user.

<- Select from using drop-down list of values to enforce consistency, data integrity, structure.

<- Auto-population of field data based on Requestor Name reduces time to complete form, reduces errors and ensures accuracy

<- Insert Attachments - keep relevant & complementary documents accessible together using InfoPath Client version

Form template's location: http://os.litwareinc.com

Terry Earls.xml - Micr...

Figure 15 An Office InfoPath 2007 form template rendered in Internet Explorer

Resolute

Resolute has been working with Microsoft on solutions that integrate Microsoft Office Groove and Windows SharePoint Services. Some of this integration can be implemented out of the box. Groove and Windows SharePoint Services work together in a variety of common scenarios; for example, creating a workspace that can be used by a mobile sales force to build and review sales proposals, sharing internal content on a SharePoint site with users outside an organization, or archiving content stored in Groove workspaces to a SharePoint document library.

Another aspect of these solutions uses the Groove forms tool to perform integration with SharePoint lists on an existing site. The integration makes use of SharePoint Web services provided by the Groove client and the Groove Enterprise Data Bridge (EDB). Synchronization occurs by copying items from SharePoint lists into the forms tool list, and from the forms tool list into a SharePoint list. Because the Web services required for this component are available in both the Groove client and EDB, the solution works in both environments.

In the client environment, users must install a synchronization tool to perform the synchronization; however, only workspace members who will perform the synchronization are required to have the synchronization tool. In a server environment, where the server is running the EDB, a simple API exposed by the application permits server application developers to configure and control this synchronization process. The synchronization application can run as a

Windows service. To configure the server version of the application, a small Web application is provided that permits users to select workspaces and tools to synchronize. This application runs on the server that also hosts the EDB and will list each workspace that the EDB instance is a member of, along with all of the forms tools contained within these workspaces.

Another aspect of integrating Groove workspaces and SharePoint sites is a solution for provisioning Groove workspaces from SharePoint sites from both EDB and a Groove client add-in. In the solution, users will see a button on Groove-enabled SharePoint sites. This button is installed as part of a SharePoint site feature. When a user clicks this button, the user will be sent a workspace invitation. On accepting the invitation, the user will find a workspace with SharePoint Files Tools for the various document libraries on the site already set up.

Depending on which version is installed (client or server), the button will run code to launch the provisioning process on the user's computer or code to launch the provisioning process on the server. SharePoint site administrators can selectively enable the provisioning feature on individual sites or even on an entire farm. In the provisioning process, a workspace is created on the Groove client or on the EDB server. Next, SharePoint Files Tools are created for each document library on the SharePoint site, and synchronization options for these tools are set up. Finally, the list of site members from the SharePoint site is read, and each user with Contributor or a higher-level access is invited into the workspace. In the client version of this solution, the user who created the workspace will already be a member and will not need to be invited into the space. In the server version, the EDB server will remain a member of the workspace. After the Groove workspace has been provisioned, users can use the workspace without any additional software. Groove's SharePoint Files Tools will automatically handle synchronization without any additional intervention. Beyond simple usage, however, the server version of this scenario presents a compelling starting point for integration with other enterprise applications, or even other data held in SharePoint itself.

Figures 16 and 17 show a Groove form before synchronization and a SharePoint list after the form and the site have been synchronized.

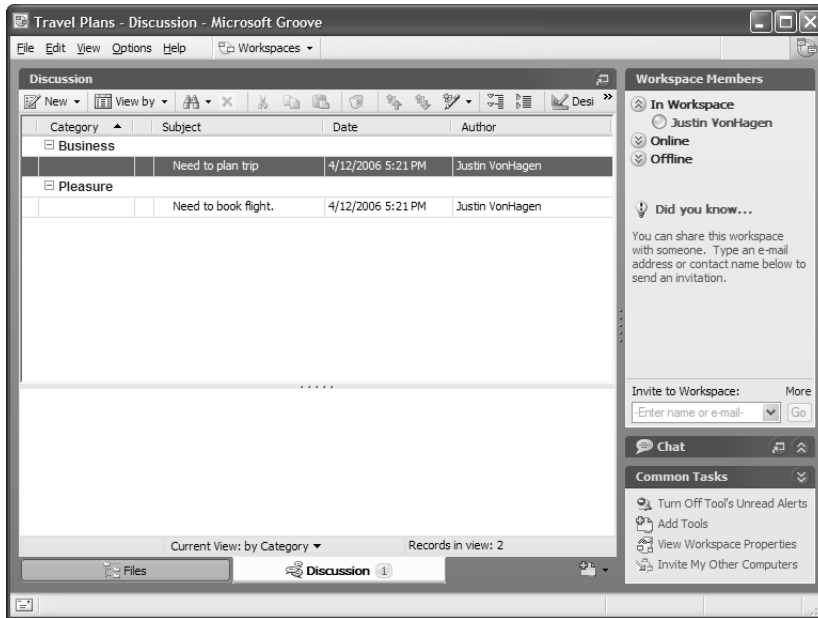


Figure 16 A Groove form before synchronization

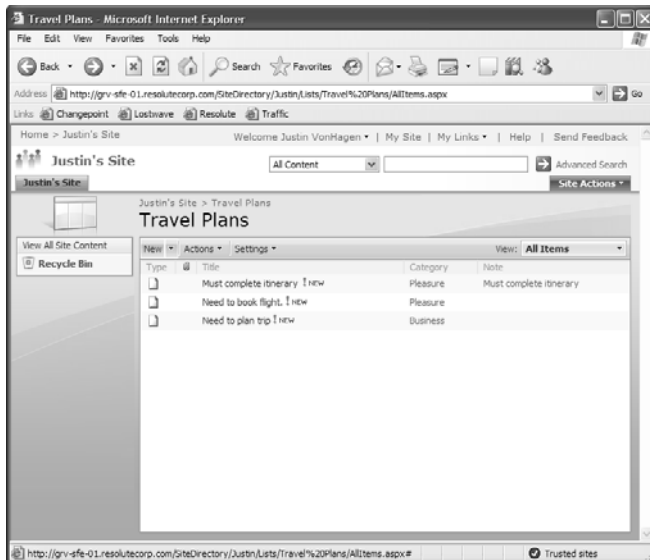


Figure 17 A SharePoint list after the Groove form and the list have been synchronized

3Sharp

3Sharp is developing a building permit application solution that uses Office InfoPath 2007, Forms Services, Excel Services, Windows SharePoint Services, and a workflow designed using Microsoft Office SharePoint Designer 2007. The 3Sharp solution is designed for municipalities that need an electronic process for building permit applications submitted by residents and business owners. This process requires that users have access to a public site to complete a building permit application form, which captures specific data about a building project. The process also requires a moderately complex workflow that executes based on the data submitted. The solution is designed for Office 2007 clients and servers, all with no code.

The core of the solution is an Office InfoPath 2007 form, which was designed once and deployed for access both through the Office InfoPath 2007 client and through a Web browser. To apply for a building permit, customers simply click a link on the city's public Web site and the browser-based form opens. Customers enter data in the form, including project information. The form is able to retrieve property data from a back-end system and is also able to retrieve a Permit Amount value using Excel Services. (The form sends customer-supplied information to a published Excel workbook and retrieves a resulting calculated value.) When the customer submits the form, it is routed to an internal SharePoint forms library. A custom SharePoint Designer-based workflow automatically notifies a clerk, who ensures that a design drawing has already been submitted, and then assigns a compliance reviewer for the project. The compliance reviewer receives a workflow e-mail message with a generated hyperlink to the actual form and reviews the form to determine whether the project meets certain requirements (for example, fire safety regulations) before submitting the form back to the forms library. If all requirements have been met, a workflow e-mail message is sent to the applicant indicating that the application has been approved and that the permit amount is due. Otherwise, a different e-mail message with a hyperlink to the form is sent to the applicant, who must then make changes and resubmit the data to start the review process again.

Microsoft Office Developer Center

Any developer working on an Office-based solution or application that is based on Windows SharePoint Services should take full advantage of the information available on the developer centers on MSDN, at <http://msdn.microsoft.com>. Developers familiar with the Microsoft Office Developer Center (<http://msdn.microsoft.com/office>) know this site as a resource where they can obtain information through technical articles and documentation, sample code, labs, and community activities.

Chapter 1

Microsoft Windows SharePoint Services 3.0

By Ted Pattison

With permissions to reprint from *MSDN Magazine*

In this chapter:

Integration with ASP.NET 2.0	2
Working with Master Pages	5
Web Parts in Windows SharePoint Services 3.0	8
Enhancements in Content Storage	13
Event Handlers	15
Workflows in Windows SharePoint Services 3.0	16
Site Definitions, Features, and Solutions	17
Internet-Style Security	19
Summary	19

The next major release of Microsoft SharePoint Products and Technologies is scheduled for fall 2006. This release includes Microsoft Windows SharePoint Services 3.0 and Microsoft Office SharePoint Server 2007. This chapter covers what's new for developers in Windows SharePoint Services 3.0. The developer-centric features of Office SharePoint Server 2007 will be covered in the next chapter, which builds on this one.

Windows SharePoint Services 3.0 is a royalty-free add-on for the Windows Server 2003 operating system. At its core, Windows SharePoint Services 3.0 plays the role of a scalable site-provisioning engine. It solves the problem of creating and managing hundreds or thousands of Web sites and making them accessible to tens of thousands of users. Windows SharePoint Services 3.0 scalability is achieved using an architecture designed with a Web farm environment in mind. This architecture is based on stateless front-end Web servers that rely on Microsoft SQL Server in the back end for storing content and other site-related data.

As in the previous versions, Windows SharePoint Services 3.0 supplies out-of-the-box collaboration features that make it simple for users to create and design Web sites with things like shared calendars, contact lists, and document libraries. However, developers should see Windows SharePoint Services 3.0 as something far more powerful than a collaboration tool aimed

at users. Windows SharePoint Services 3.0 is a full-fledged development platform that adds a tremendous amount of value on top of ASP.NET.

Windows SharePoint Services 3.0 provides value beyond the ASP.NET 2.0 development platform by means of a richer provider model. It facilitates provisioning and storage for pages, lists, and document libraries. This provisioning can be driven through custom code or through user actions in the browser-based user interface. Behind the scenes, Windows SharePoint Services 3.0 automatically works out how and where to store this content. Windows SharePoint Services 3.0 also eliminates many tedious tasks required in typical ASP.NET development because it additionally supplies the UI elements for users to add, view, and modify content.

In this chapter, I assume you already know something about the previous version of Windows SharePoint Services (2.0), so I can concentrate on the most significant developer enhancements to this new version. For more background information you can read “Use Windows SharePoint Services as a Platform for Building Collaborative Applications” (<http://msdn.microsoft.com/msdnmag/issues/04/07/WindowsSharePointServices/>).

Also keep in mind that research and code samples for this chapter are based on the beta 1 release of Windows SharePoint Services 3.0. It’s possible that some of the terms and code used in this chapter might differ in the released version.

Integration with ASP.NET 2.0

Windows SharePoint Services 3.0 provisioning starts at the level of the IIS Web site. Before you can create your first Windows SharePoint Services site, someone must run an administrative procedure to extend 3.0 functionality onto one or more IIS Web sites. In Windows SharePoint Services 2.0, the term *virtual server* was used to describe an IIS Web site that had been extended with Windows SharePoint Services functionality. To avoid confusion with another Microsoft product of the same name, the Windows SharePoint Services 3.0 documentation now refers to an IIS Web site extended with Windows SharePoint Services functionality as a *Web application*.

Windows SharePoint Services 2.0 was integrated with IIS 6.0 and ASP.NET 1.1 using an ISAPI filter DLL. This integration technique results in IIS routing requests to Windows SharePoint Services before ASP.NET. This routing has proven to be problematic in certain situations because Windows SharePoint Services takes control of an incoming HTTP request before it has a chance to be properly initialized with ASP.NET context.

The way in which Windows SharePoint Services 3.0 integrates with ASP.NET has been completely redesigned. First of all, Windows SharePoint Services 3.0 is built upon ASP.NET 2.0, which provides significant enhancements over ASP.NET 1.1. Furthermore, the integration between Windows SharePoint Services 3.0 and ASP.NET 2.0 was changed to route incoming requests through the ASP.NET runtime before Windows SharePoint Services. The Windows

SharePoint Services team achieved these improvements to the routing infrastructure by removing the ISAPI filter and adding an *HttpModule* and an *HttpHandler* that are registered with ASP.NET using standard Web.config entries. This means incoming HTTP requests always enter the ASP.NET run-time environment and are fully initialized with ASP.NET context before they are forwarded to the code written by the Windows SharePoint Services team to carry out Windows SharePoint Services–specific processing.

Also note that when you extend an IIS Web site to become a Windows SharePoint Services Web application, Windows SharePoint Services 3.0 adds a wildcard application map to the IIS metabase. This wildcard application map serves to route all incoming HTTP requests to the ASP.NET runtime regardless of their extension. This wildcard application map is necessary to forward a request for any type of file (for example, .pdf, .doc, .docx) to ASP.NET, which then forwards the request to Windows SharePoint Services for processing.

Another relevant issue of the new architecture has to do with how .aspx pages are parsed and compiled. The .aspx page parser used by ASP.NET 1.1 works only with .aspx pages that reside on the local file system. However, Windows SharePoint Services architecture relies on storing .aspx pages inside a SQL Server database. Since Windows SharePoint Services 2.0 relies on ASP.NET 1.1, the Windows SharePoint Services team had to create their own .aspx page parser. Unfortunately, the .aspx parser of Windows SharePoint Services 2.0 does not support many of the richer features offered by the ASP.NET .aspx page parser.

ASP.NET 2.0 introduced a new pluggable component type known as a virtual path provider. A developer can write a custom component that retrieves .aspx pages for any location including a database such as SQL Server. Once a custom virtual path provider retrieves an .aspx page, it can then hand it off to ASP.NET to conduct the required parsing and compilation. ASP.NET also gives the virtual path provider a good deal of control as to how .aspx pages are parsed and whether they are compiled or run in a non-compile mode.

The Windows SharePoint Services 3.0 team has created their own virtual path provider named *SPVirtualPathProvider*, which is shown in Figure 1-1 on the next page. As you can see, the *SPVirtualPathProvider* is able to retrieve .aspx pages from SQL Server and then hand them off to the .aspx page parser supplied by ASP.NET 2.0. That means the Windows SharePoint Services 3.0 team was not required to evolve their .aspx page parser from the previous version. It also means that Windows SharePoint Services 3.0 does not suffer from a reduced feature set with respect to page parsing as it does with Windows SharePoint Services 2.0.

If you're familiar with the architecture of Windows SharePoint Services 2.0, you've probably heard the terms “ghosting” and “unghosting” used in conjunction with the .aspx pages of a Windows SharePoint Services 2.0 site. Page ghosting is a Windows SharePoint Services feature that allows a front-end Web server to store an .aspx page template on its local file system and to share that page template across many different sites. Page ghosting offers performance benefits because Windows SharePoint Services can serve up pages for thousands of sites using a page template stored on the local file system and loaded into memory a single time.

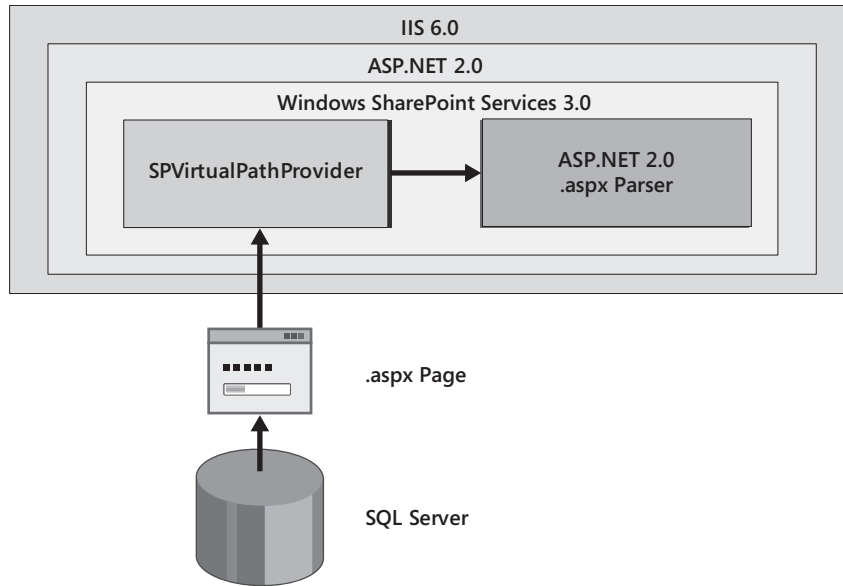


Figure 1-1 Windows SharePoint Services employs a custom virtual path provider to employ the .aspx page parser supplied by ASP.NET 2.0.

Windows SharePoint Services 2.0 supports user modifications to the page template using tools such as Microsoft Office FrontPage 2003. Once a user modifies a page template and saves the changes, a customized version of the page is stored on SQL Server for that particular site. In Windows SharePoint Services 2.0, this is often referred to as unghosting a page.

Windows SharePoint Services 3.0 still supports page templates that live on the Web server as well as customized versions of those page templates that are stored on SQL Server. However, the Windows SharePoint Services team and their documentation have stopped using the terms ghosting and unghosting because they do not translate well into other spoken languages. In Windows SharePoint Services 3.0, the term “uncustomized page” refers to a page template used directly from the local file system of the Web server, and the term “customized page” refers to a modified version of the page template that has been written to the content database for a particular site.

Another change to be aware of is that Microsoft Office FrontPage 2003 has been renamed in its new release as Microsoft Office SharePoint Designer 2007. Like the previous versions of FrontPage, Office SharePoint Designer 2007 is targeted more toward users than developers. However, it’s nonetheless a handy tool to have in your bag of tricks as a Windows SharePoint Services developer.

Office SharePoint Designer 2007 provides a code editor and WYSIWYG designer for customizing .aspx pages within Windows SharePoint Services 3.0 sites. You can also create new pages within a Windows SharePoint Services site that have no corresponding page template on the Web server. Office SharePoint Designer 2007 supports creating lists and document

libraries and even supplies a new wizard for creating custom workflows on a Windows SharePoint Services site. Workflows in Windows SharePoint Services 3.0 will be discussed in more depth later in this chapter.

Working with Master Pages

One of the most tedious aspects of customizing and branding sites in Windows SharePoint Services 2.0 is creating a consistent look and feel across pages. This is because ASP.NET 1.1 does not provide any suitable page templating technique that can be used across the pages within a Windows SharePoint Services 2.0 site. As a result, many developers and designers have resorted to copying and pasting HTML layouts from page to page. As you can imagine, this makes it very hard to customize and maintain a site whose layout requirements differ from the out-of-the box experience with a standard Windows SharePoint Services 2.0 site.

As you're probably aware, ASP.NET 2.0 has introduced a powerful page templating feature known as master pages. A master page is a template that allows you to define a standard page layout for an entire site with elements such as a banner, navigation controls, and menus. The pages that link to a master page are known as content pages.

The key concept is that each content page links to the master page to get the shared layout, and then extends the master page by adding customized content to replaceable named placeholders. For more information about how master pages work in ASP.NET, read "Master Your Site Design with Visual Inheritance and Page Templates" (<http://msdn.microsoft.com/msdnmag/issues/04/06/ASPNET20MasterPages/default.aspx>).

Windows SharePoint Services 3.0 was designed from the ground up to embrace the master page infrastructure of ASP.NET 2.0. Every Windows SharePoint Services 3.0 site is provisioned with a special catalog known as the Master Page Gallery containing a master page named `default.master`. This master page defines a common layout for every site's home page (`Default.aspx`) as well as all the standard Windows SharePoint Services form pages associated with lists and document libraries (for example, `AllItems.aspx`, `NewItem.aspx`). The master page layout includes standard Windows SharePoint Services menus and navigation controls. An example of a page based on the standard layout defined by `default.master` is shown in Figure 1-2 on the next page.

The definition of `default.master` includes several different named placeholders such as *PlaceholderPageTitle*, *PlaceholderMain*, and *PlaceholderLeftNavigation*. This makes it relatively simple to create a new custom content page that has the same layout as the other pages in a site. Take a look at the simplicity of the content page definition shown in the following code.

```
<%@ Page language="C#" MasterPageFile="~masterurl/default.master" %>
<asp:Content ContentPlaceHolderId="PlaceholderMain" runat="server">
    <h3>welcome to customization with WSS v3</h3>
    This is so much easier than it was in WSS v2!
</asp:Content>
```

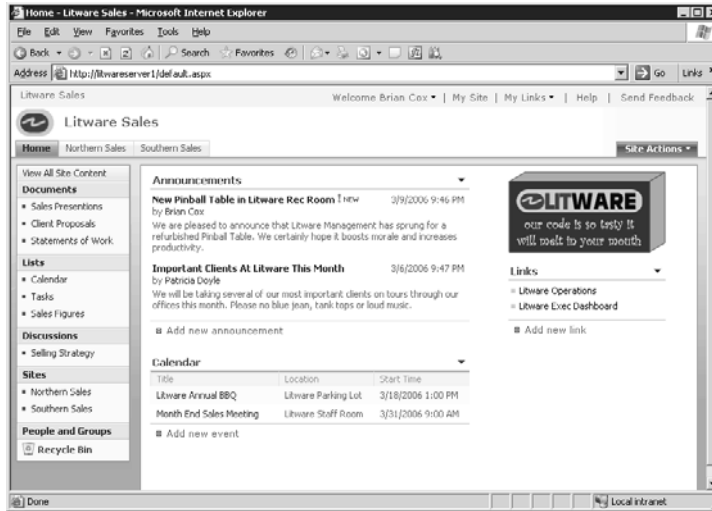


Figure 1-2 The standard layout of pages within a Windows SharePoint Services site is controlled through `default.master`.

The content page definition shown in the code could not be any simpler. It does nothing other than add a minimal fragment of HTML content inside `PlaceHolderMain`. However, it produces the standard Windows SharePoint Services page layout with site icons, menus, and navigation bars, as shown in Figure 1-3. Once you learn how to use the standard set of placeholders defined in `default.master`, you can easily swap out standard Windows SharePoint Services elements like menus and navigation bars and replace them with your own ASP.NET controls and Web Parts. This can be done for pages at the scope of a site, a site collection, or even on a farm-wide basis.

Master pages and content pages are stored and loaded using the same concepts of page templates and page customization discussed earlier. There are page templates defined for the master page as well as for content pages that reside on the local file system of the front-end Web server. Each site initially uses an uncustomized (that is, ghosted) version of the master page template and content page templates. However, once a user customizes and saves one of these pages for a particular site using a tool such as Office SharePoint Designer 2007, a customized (unghosted) version is saved in the SQL Server database.

It's possible to customize the master page for a site while leaving the content pages uncustomized. Likewise, it's possible to customize one or more content pages while leaving the master page uncustomized. Furthermore, if you customize either the master page or a content page and later wish to undo your changes, both the browser-based UI of Windows SharePoint Services 3.0 and Office SharePoint Designer 2007 provide simple menu commands to discard customization changes from the SQL database and revert back to the original page template.

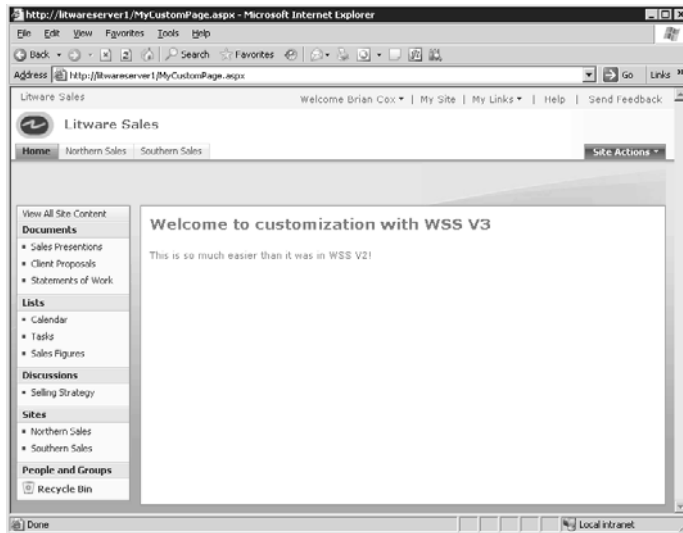


Figure 1-3 It's easy to create custom content pages that use the standard Windows SharePoint Services layout.

You might have noticed in the code on page 5 that the content page shown in Figure 1-3 links to the master page using a special syntax in the form of `~masterurl/default.master`. This is a tokenized reference to a master page that can be changed programmatically on a site-wide basis. You can accomplish this by acquiring an *SPWeb* reference to the current site and then updating the *MasterUrl* property.

```
using System.Windows.Forms;
using Microsoft.SharePoint;

namespace MyHandyPageCustomizer {
    public partial class frmMain : Form {
        void cmdSetMasterUrl_Click(object sender, EventArgs e) {
            SPSite siteCollection = new SPSite(@"http://localhost");
            SPWeb TopLevelSite = siteCollection.OpenWeb();
            TopLevelSite.MasterUrl = txtMasterUrl.Text;
            TopLevelSite.Update();
        }
    }
}
```

Note that each site has its own Master Page Gallery with a `default.master` and its own *MasterUrl* property. That means all the sites in a site collection do not automatically use the same master. However, with the use of recursion it's pretty easy to write some code against the Windows SharePoint Services 3.0 object model to synchronize a top-level site and all the child sites in a site collection to use the same master pages.

```
void cmdSynchronizeChildren_Click(object sender, EventArgs e) {
    SPSite siteCollection = new SPSite(@"http://localhost");
    SPWeb TopLevelSite = siteCollection.OpenWeb();
```

```
    UpdateMasterUrlRecursive(TopLevelSite, txtMasterUrl.Text);
}

void UpdateMasterUrlRecursive(SPWeb site, string Url) {
    site.MasterUrl = Url;
    site.Update();
    foreach (SPWeb child in site.Webs) {
        UpdateMasterUrlRecursive(child, Url);
    }
}
```

In addition to `~masterurl/default.master`, there is another dynamic token for master pages in the form of `~masterurl/custom.master`. This dynamic token works in conjunction with the `CustomMasterUrl` property of a site and provides a secondary target master page that can be redirected programmatically. There are also two static master page tokens that start with either `~site` or `~sitecollection`. These static tokens allow you to hard-code a relative path to a master page from either the root of the current site or the root of the current site collection.

Web Parts in Windows SharePoint Services 3.0

One of the most popular ways for developers to extend Windows SharePoint Services 2.0 sites has been to create custom Web Parts. Web Parts are great because they add the extra dimensions of user customization and personalization. As a consequence, many teams at Microsoft and third-party companies alike have built custom Windows SharePoint Services 2.0 solutions using Web Parts.

Because of the popularity of Web Parts in Windows SharePoint Services 2.0, Microsoft decided to add support for custom Web Part development to ASP.NET 2.0. This strategy to reach a wider audience of developers was accomplished by creating a new Web Part infrastructure for ASP.NET 2.0 that is similar yet distinct from the Web Part infrastructure created for Windows SharePoint Services 2.0.

Consequently, there are now two different styles of Web Parts. The older WSS-style Web Parts depend on `Microsoft.SharePoint.dll` and must inherit from the `WebPart` base class defined by the Windows SharePoint Services 2.0 teams in the `Microsoft.SharePoint.WebPartPages` namespace. The newer ASP-style Web Parts depend on `System.Web.dll` and must inherit from a different base class also named `WebPart` defined by the ASP.NET 2.0 team in the `System.Web.UI.WebControls.WebParts` namespace.

It is an important design goal for Windows SharePoint Services 3.0 to run both the older WSS-style Web Parts as well as the newer ASP-style Web Parts. This has been accomplished by building the Windows SharePoint Services 3.0 support for Web Parts on top of the ASP.NET Web Part infrastructure, and then making changes to `Microsoft.SharePoint.dll` so

that WSS-style Web Parts written for the Windows SharePoint Services 2.0 environment would be forwardly compatible with the Windows SharePoint Services 3.0 run-time environment.

To explain how Web Parts are loaded and run in Windows SharePoint Services 3.0, this section discusses how the Windows SharePoint Services 3.0 architecture was redesigned to layer on top of the ASP.NET 2.0 Web Part infrastructure. First, I will cover how Web Part Pages are laid out in Windows SharePoint Services 3.0, and then I get into the details of how to develop custom Web Parts for a Windows SharePoint Services 3.0 environment.

To run Web Parts in an ASP.NET 2.0 application, you must create an .aspx page that contains exactly one instance of the *WebPartManager* control and one or more *WebPartZone* controls. The *WebPartManager* control is responsible for serializing Web Part–related data as well as storing and retrieving it from the tables in the ASP.NET services database.

The .aspx page serving as a Web Part Page can also contain Editor Parts, which allow users to customize and personalize persistent Web Part properties. Web Part Pages can also contain Catalog Parts, which allow users to add new Web Parts to zones. To acquire more background about how the ASP.NET 2.0 Web Part infrastructure works, read “ASP.NET 2.0: Personalize Your Portal with User Controls and Custom Web Parts” (<http://msdn.microsoft.com/msdnmag/issues/05/09/WebParts/>).

The Web Part infrastructure of Windows SharePoint Services 3.0 is built on a control named *SPWebPartManager* that is derived from the ASP.NET 2.0 *WebPartManager* control. The *SPWebPartManager* control overrides the standard behavior of the *WebPartManager* control to persist Web Part data inside the Windows SharePoint Services content database instead of the ASP.NET services database. In most cases, you don’t have to worry about dealing directly with the *SPWebPartManager* control because the one and only required instance is already defined in the standard default.master page. When you create a content page that inherits from default.master, the *SPWebPartManager* control is already there.

The other controls that appear on a typical Windows SharePoint Services 3.0 Web Part Page are shown in Figure 1-4 on the next page and include Web Part zones, Editor Parts, and Catalog Parts. Note that Web Part zones for a Web Part Page in Windows SharePoint Services 3.0 should be created using the *WebPartZone* control defined in the *Microsoft.SharePoint.WebPartPages* namespace and not the standard *WebPartZone* control from ASP.NET 2.0.

Instances of the *WebPartZone* control are usually defined in content pages. The following code shows a simple example of creating a content page designed to act as Web Part Page in a Windows SharePoint Services 3.0 site. As you can see, this .aspx file links to default.master just like the example you saw earlier. However, it also explicitly inherits from the *WebPartPage* base class and adds two *WebPartZone* controls into *PlaceHolderMain*.

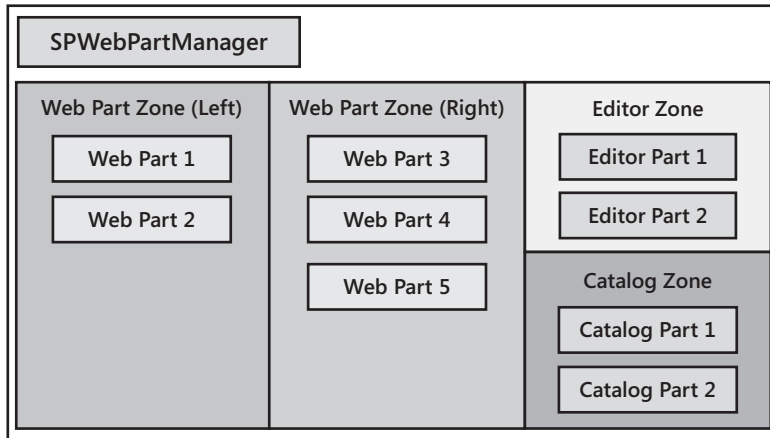


Figure 1-4 A Web Part Page in Windows SharePoint Services 3.0 requires the *SPWebPartManager* control and one or more *WebPartZone* controls. If you create a content page that inherits from the *WebPartPage* class, you also get the benefit of Windows SharePoint Services 3.0 supplying Editor Parts and Catalog Parts.

```
<%@ Assembly Name="[Fully-qualified name for Microsoft.SharePoint.dll]" %>

<%@ Page language="C#" MasterPageFile="~masterurl/default.master"
    Inherits="Microsoft.SharePoint.WebPartPages.WebPartPage" %>

<%@ Register Tagprefix="WebPartPages"
    Namespace="Microsoft.SharePoint.WebPartPages"
    Assembly="[Fully-qualified name for Microsoft.SharePoint.dll]" %>

<asp:Content ContentPlaceHolderId="PlaceHolderMain" runat="server">
  <h3>My Custom Web Part Page</h3>
  <table border="5" cellpadding="5" cellspacing="0">
    <tr>
      <td valign="top">
        <WebPartPages:WebPartZone runat="server" ID="Left"
          Title="Left Zone" />
      </td>
      <td valign="top">
        <WebPartPages:WebPartZone runat="server" ID="Right"
          Title="Right Zone" />
      </td>
    </tr>
  </table>
</asp:Content>
```

When you create a Web Part Page for a standard ASP.NET 2.0 application, you are required to add logic that interacts with the *WebPartManager* control to manage the Web Part display mode. Typically, you also need to explicitly add Editor Parts and Catalog Parts to the page along with the HTML layout to accommodate them. Fortunately, you don't have to do these things when creating content pages for a Windows SharePoint Services 3.0 site. Instead, you

inherit from the *WebPartPage* class that's defined in the *Microsoft.SharePoint.WebPartPages* namespace and let it do all this work for you behind the scenes.

Figure 1-5 shows a screen shot of a custom Web Part Page in action. This is the display generated by the custom Web Part Page definition shown in the previous code when the user has entered edit mode. Notice that the page allows users to add Web Parts into zones and to modify existing Web Parts using standard Editor Parts.

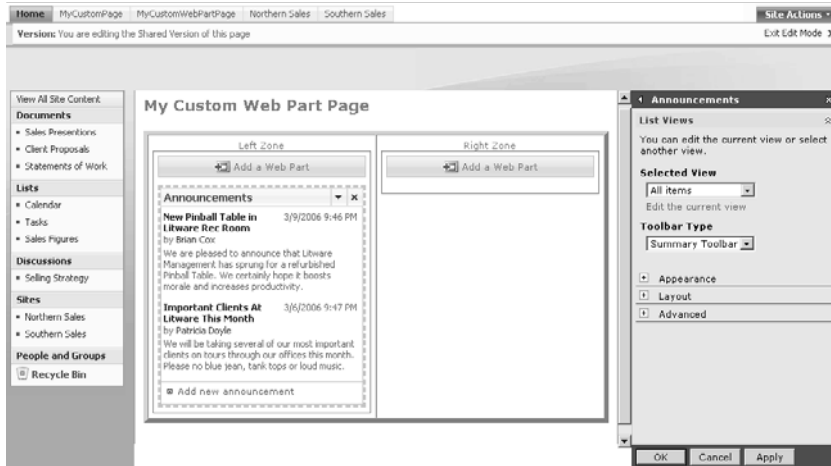


Figure 1-5 Custom Web Part Pages that inherit from the *WebPartPage* class provide automatic support for managing display mode as well as providing Editor Parts and Catalog Parts.

Developing Custom Web Parts

The preferred approach for creating Web Parts for Windows SharePoint Services 3.0 sites is to create ASP-style Web Parts. At a minimum, this involves creating a class that inherits from the *WebPart* base class defined in the *System.Web.UI.WebControls.WebParts* namespace and overriding the *RenderContents* method. If you want to add persistent Web Part properties, you use the same technique that is used in ASP.NET, as shown in the following code.

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls.WebParts;

namespace LitwareWebParts {
    public class SimpleWebPart : WebPart {

        // backing field for caching property value
        protected string _zipCode;

        [ // persistent web part property
        Personalizable(),
        WebBrowsable(true),
        WebDisplayName("Zip Code"),
        WebDescription("used to track user zip code")
```

```
    ]  
    public string ZipCode {  
        get{ return _ZipCode; }  
        set{ value = _ZipCode; }  
    }  
  
    protected override void RenderContents(HtmlTextWriter writer) {  
        writer.Write("You live at " + _ZipCode);  
    }  
}  
}
```

The Web Part definition shown in this code has no dependencies on Microsoft.SharePoint.dll so it can be used in either a standard ASP.NET application or a Windows SharePoint Services 3.0 site. In many cases, you will want to add a reference to Microsoft.SharePoint.dll with a Web Part library project so the code behind your Web Parts can program against the Windows SharePoint Services 3.0 object model.

In addition to supporting ASP-style Web Parts, Windows SharePoint Services 3.0 was also designed to support Web Parts created for the Windows SharePoint Services 2.0 environment. These older WSS-style Web Parts inherit from the *WebPart* base class from Microsoft.SharePoint.dll that is defined in the *Microsoft.SharePoint.WebPartPages* namespace.

The *WebPart* class in the Windows SharePoint Services 2.0 version of Microsoft.SharePoint.dll was designed to inherit from the ASP.NET *Control* class as shown in Figure 1-6. However, you can also see that the *WebPart* class in the Windows SharePoint Services 3.0 version of Microsoft.SharePoint.dll has been modified to inherit from the ASP.NET *WebPart* class instead. This versioning technique of changing a base class in a later version of an assembly is known as rebasing. The rebasing of the *WebPart* base class in Microsoft.SharePoint.dll is one of the keys to supporting older WSS-style Web Parts in a Windows SharePoint Services 3.0 environment.

If you look at the standard Web.config file for a Windows SharePoint Services 3.0 Web application, you'll see that it contains configuration elements to redirect references from the Windows SharePoint Services 2.0 version of Microsoft.SharePoint.dll to the Windows SharePoint Services 3.0 version. This redirection in combination with the rebasing of the Windows SharePoint Services *WebPart* base class allows Web Part DLLs written for the Windows SharePoint Services 2.0 environment to run in the Windows SharePoint Services 3.0 environment without any need for recompilation.

If you decide to move a Windows SharePoint Services 2.0 Web Part project to Microsoft Visual Studio 2005, you can continue to evolve your code using the same style as you have in the past and things will still work. However, you have another option once you have moved to Visual Studio 2005 and switched the project reference to Microsoft.SharePoint.dll to the new Windows SharePoint Services 3.0 version. Since the ASP.NET *WebPart* base class is now a part

of the inheritance hierarchy, you can change some aspects of your WSS-style *WebPart* class as if it were an ASP-style Web Part. This is known as a hybrid Web Part.

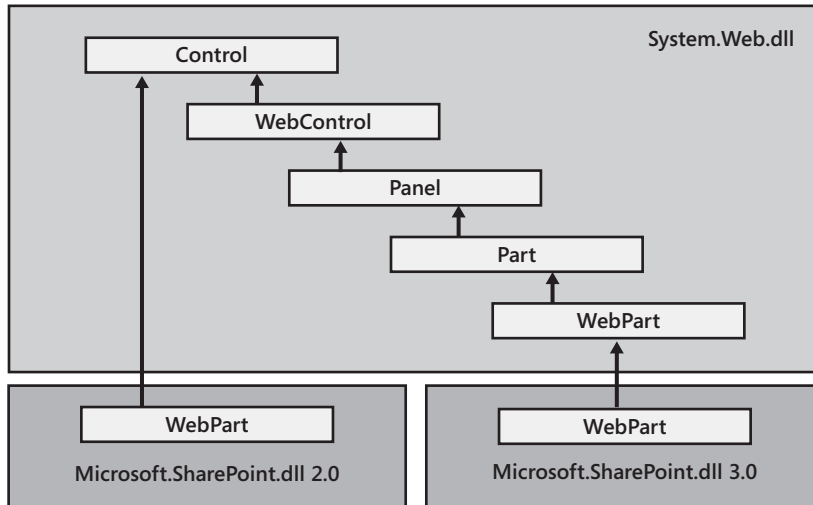


Figure 1-6 The Windows SharePoint Services *WebPart* class has been rebased on Windows SharePoint Services 3.0 to inherit from the ASP.NET *WebPart* class.

Enhancements in Content Storage

One criticism that developers have had with Windows SharePoint Services 2.0 is that several valuable features supported with document libraries do not extend to lists. For example, document libraries support versioning and events, but lists do not. To address this criticism, the Windows SharePoint Services teams worked hard to extend the functionality of lists and bring them up to par with document libraries. With Windows SharePoint Services 3.0, lists support many of the same features as document libraries including versioning, events, and folders. There are also some new features in Windows SharePoint Services 3.0 supported by both lists and document libraries such as exposing data through automatic RSS feeds.

Performance with large lists and document libraries has also been another concern with Windows SharePoint Services 2.0. For example, lists often start showing degraded performance when the number of items exceeds 2000. Document libraries have similar performance concerns. However, the rule of thumb for not exceeding 2000 documents applies to each folder within a document library as opposed to the document library itself. Therefore, coming up with a scheme to partition documents across multiple folders within a document library has come to be a popular approach in Windows SharePoint Services 2.0 for dealing with a large number of documents.

Windows SharePoint Services 3.0 introduces a new column indexing feature to alleviate some of the performance problems just mentioned. From a list settings page or a document library settings page, you can add an index to any column. Doing this does not actually create a

physical index in SQL Server. Instead, it creates a table with the integer ID of the list item or document and the value of the indexed column. Windows SharePoint Services then uses this table to improve the performance of data returned from views, especially a view with a filter based on the indexed column.

Many developers have expressed a desire to work with Windows SharePoint Services fields at a lower level to obtain more control over field rendering and validation. The Windows SharePoint Services team responded by adding extensible field types in Windows SharePoint Services 3.0. You can create an extensible field type by writing a class in C# or Visual Basic .NET that inherits from one of the built-in Windows SharePoint Services field types such as *SPFieldText*, *SPFieldNumber*, or *SPFieldUser*. An extensible field type can also utilize an ASP.NET *User* control that contains your favorite Web Controls. This allows you to use the same techniques for control initialization and validation that you have used in ASP.NET applications.

Another nice innovation added to Windows SharePoint Services 3.0 is custom site columns. A site column is a reusable definition that can be used across multiple lists. A site column defines the name for a column, its underlying field type, and other characteristics such as its default value, formatting, and validation. Once you have defined a site column, you can then use it as you define the structure of your user-defined lists. An obvious advantage is that you can update the site column in a single place and have that update affect all the lists where the site column has been used.

A site column is defined within the scope of a single site, yet it is visible to all child sites below the site in which it has been defined. Therefore, you can create a site column that is usable across an entire site collection by defining it in the top-level site.

One convenient technique made available with the introduction of site columns is the ability to perform field lookups across sites. For example, you can create a site column in a top-level site that performs a lookup on a list in the same site. Then you can create other lists within child sites that use this site column to perform lookups on the list in the top-level site. There was no way to accomplish this task in Windows SharePoint Services 2.0 short of writing custom code.

Imagine you want to store several different types of documents in the same document library. For example, say you need to store customer presentations, customer proposals, and customer reports. What if you want each of these document types to have its own unique set of custom columns and its own unique event handlers? In Windows SharePoint Services 2.0, you can add extra columns and event handlers only to the document library itself, which always affects every document in that document library. Furthermore, a Windows SharePoint Services 2.0 document library can have only one associated document template. Windows SharePoint Services 3.0 introduces a powerful new storage mechanism known as content types to solve this problem.

A content type is a flexible and reusable Windows SharePoint Services type that defines the shape and behavior for an item in a list or a document in a document library. For example,

you can create a content type for a customer presentation document with a unique set of columns, an event handler, and its own document template. You can create a second content type for a customer proposal document with a different set of columns, a workflow, and a different document template. Then you can create a new document library and configure it to support both of these content types. The introduction of content types is significant to Windows SharePoint Services 3.0 because it provides an ability that did not exist in Windows SharePoint Services 2.0 to deal with heterogeneous types of content in lists and document libraries.

Event Handlers

While Windows SharePoint Services 2.0 provides support for event handlers, this support is fairly minimal and has generated a number of complaints from the developer community. Events in Windows SharePoint Services 2.0 are supported for document libraries but not for lists. Furthermore, Windows SharePoint Services 2.0 supports only asynchronous events that fire after the user's action has been committed to the SQL Server database. This means there is no way for the developer to cancel a user's action inside an event handler.

Support for event handling improves in Windows SharePoint Services 3.0 by an order of magnitude. In addition to supporting the asynchronous events that exist in Windows SharePoint Services 2.0, there is now also support for synchronous events, which enables the developer to cancel the user's action. For example, you can stop a user from deleting a document once it has been approved or from creating an order with an order date in the future. Furthermore, events are supported on list items as well as documents in a document library.

You create an event handler by writing a custom class that inherits from one of the Windows SharePoint Services receiver classes and overriding methods to handle events. For example, to handle insert events for items in a list, you should create a class that inherits from the *SPIItemEventReceiver* class and override the *ItemAdding* method, as shown in the following code.

```
using System;
using Microsoft.SharePoint;

namespace Litware {
    public class MyReceiver : SPIItemEventReceiver {
        public override void ItemAdding(SPIItemEventProperties properties) {
            SPWeb web = properties.OpenWeb();
            Guid ListId = properties.ListId;
            int ListItemId = properties.ListItemId;
            SPListItem Orders = web.Lists[ListId].GetItemById(ListItemId);
            // check to make sure order date is not day in future
            DateTime OrderDate = Convert.ToDateTime(Orders["OrderDate"]);
            if (OrderDate.CompareTo(DateTime.Today) > 0) {
                string msg = "You cannot enter orders for future days";
                properties.ErrorMessage = msg;
                properties.Cancel = true;
                return;
            }
        }
    }
}
```

```
    }  
  }  
}
```

Now that you have seen how to write the code for a simple event handler, you'll learn how to bind it to a particular list or document library. You can accomplish this either through writing code against the object model or by defining an event receiver in a Windows SharePoint Services feature. For example, you can bind this event receiver to a list using the following code.

```
SPList list = web.Lists["Orders"];  
list.EventReceivers.Add(SPEventReceiverType.ItemAdding,  
    "[Fully-qualified Assembly name]",  
    "Litware.MyReceiver");
```

Windows SharePoint Services receiver classes use a naming convention for overridable methods that differentiates synchronous and asynchronous events. For example, the *ItemAdding* event is a cancellable, synchronous event that fires before the change is made to the content database. These synchronous events provide a great way to validate column values as shown in the preceding event handler code on page 15.

There is a complementary asynchronous event named *ItemAdded* that occurs after the change has been written to the content database. Unlike synchronous events, asynchronous events do not support cancelling the user's actions. Instead they provide the means to perform a required operation after a change has been made to a document or list item such as assigning the value of a calculated column or sending an e-mail notification.

In addition to supporting events on list items and documents, Windows SharePoint Services 3.0 supports many other types of new events that were not available in Windows SharePoint Services 2.0. There are list events that fire when someone changes a list definition. For example, you can cancel the user's action whenever someone tries to remove or rename a column in one of your custom lists. There are also events that fire whenever someone deletes or moves a site or an entire site collection.

Workflows in Windows SharePoint Services 3.0

Workflow applications have been getting quite a bit of attention at Microsoft lately. The WinFX run-time components to be released in fall 2006, named the Windows Workflow Foundation (WinWF), add a complete infrastructure for building workflow-style applications. The WinWF infrastructure includes a workflow engine, pluggable components to persist workflow state, and a Visual Studio designer that makes it easy to create custom workflows by dragging components known as activities onto a workflow design surface.

Windows SharePoint Services 3.0 builds on WinWF to provide a foundation for attaching business logic to list items and documents. Windows SharePoint Services 3.0 extended the basic workflow model of WinFX by associating a task list and history list with each workflow.

Windows SharePoint Services 3.0's extensions to WinFX add a degree of responsibility and accountability to workflows that are human-oriented in nature such as a workflow for reviewing or approving a document.

Both Windows SharePoint Services 3.0 and Office SharePoint Server 2007 ship with workflows that are installed and ready to use out of the box. Windows SharePoint Services 3.0 includes some simple routing workflows for things such as moderation and approval. Office SharePoint Server 2007 supplies workflows that are more complex and are used to support features such as its Web Content Management approval process.

The creation of custom workflows represents an obvious extensibility point for developers creating business solutions with Windows SharePoint Services 3.0 and Office SharePoint Server 2007. In addition to the standard support of the Visual Studio Extensions for WinWF, the Office team also plans to ship a Windows SharePoint Services-specific workflow SDK and a workflow starter kit including Visual Studio project templates for creating custom workflows targeted at Windows SharePoint Services 3.0 sites.

Office SharePoint Designer 2007 also provides support for creating custom workflows in Windows SharePoint Services 3.0 sites. This support is designed more for the power user than developers because it provides a wizard to attach ad hoc business logic to list items and documents in a production Windows SharePoint Services 3.0 site.

Site Definitions, Features, and Solutions

Developers and companies that have used Windows SharePoint Services 2.0 as a platform to build business solutions have found that working with low-level site definitions provides the greatest amount of control and reusability. A site definition is a directory on the front-end Web server containing XML files and .aspx page templates that define a blueprint for a site, including list schemas and page layouts. The XML-based language that is used inside many site definition files is called Collaborative Application Markup Language (CAML).

Developers who have worked with Windows SharePoint Services 2.0 site definitions have voiced several criticisms. First, the XML files inside Windows SharePoint Services 2.0 site definitions are poorly factored, making them unwieldy. Second, Microsoft doesn't support evolving a site definition in production once it has already been used to create sites. That means there isn't a supported technique for site definitions to be used to add functionality to an existing Windows SharePoint Services 2.0 site. Third, site definitions and the custom assemblies they depend on create deployment issues, because the files have to be pushed out to each front-end Web server in the farm without any assistance from the Windows SharePoint Services infrastructure. Finally, Windows SharePoint Services 2.0 provides no means to localize a site definition. This has been frustrating for companies that have wanted to internationalize the business solutions they have built on top of Windows SharePoint Services 2.0.

The first significant enhancement in this area is the introduction of features. A feature is like a site definition in that it is a directory containing CAML-based XML files and page templates. However, features offer a much more modular approach because they are not required to define the blueprint for an entire site. Instead, a simple feature can define a single site element such as a custom list definition or a custom menu command that is to be displayed in one of the standard Windows SharePoint Services menus.

A very attractive aspect of features is that they can be activated on an existing site. For example, you can create a feature that defines a custom list type, an instance of that list type, and an event handler or a workflow on that list instance. Once the feature has been installed, it can be activated in any site within a farm. Feature activation can be accomplished through the Windows SharePoint Services user interface, from the command line, or through custom code written against the Windows SharePoint Services object model. Once the feature has been activated, the site will have your new custom list and any behavior you want to attach to it. The key point is that features now allow you to add new types, storage, and functionality to existing sites.

Features also serve to make developing site definitions less unwieldy. In Windows SharePoint Services 2.0, each list definition had to be defined inside the context of a site definition. Now every list definition can be factored out into its own feature. Site definitions are now easier to create, as they can be composed using feature references. This is the approach that the Windows SharePoint Services team has taken with all the list types that ship as part of their collaboration services.

Windows SharePoint Services 3.0 introduces a new deployment mechanism called a solution. A Windows SharePoint Services solution is similar to Web Part packages from Windows SharePoint Services 2.0 in the sense that it is an aggregate CAB file containing XML instructions and files that need to be deployed on each front-end Web server. However, Windows SharePoint Services solutions go beyond Web Part packages to support the deployment of features, site definitions, and related assemblies used for event handlers and workflows.

Windows SharePoint Services 3.0 support for solutions also assists in pushing deployment files to each Web server in a farm. An administrator adds a solution to a Windows SharePoint Services farm, which copies the solution CAB file into the configuration database. Next, the administrator runs a command to deploy the solution, at which time Windows SharePoint Services starts a timer job to push the solution CAB file out to each front-end Web server and install it.

One of the most welcome changes in Windows SharePoint Services 3.0 for developers is the new support for localization. Windows SharePoint Services 3.0 support for localization has been built on the localization infrastructure of ASP.NET 2.0. It is now possible (and recommended) to create site definitions and features in a language-neutral fashion and to maintain string literals in .resx files. Within the XML files and .aspx page template files of a feature or

site definition, you can acquire the value for a named string that has been localized into a .resx file using standard ASP.NET syntax.

```
<%$Resources:Litware,MyString%>
```

Internet-Style Security

Authentication in Windows SharePoint Services 2.0 is based on Windows accounts and their associated Security IDs (SIDs). In a practical sense, this means that Windows SharePoint Services 2.0 is tightly coupled with Active Directory when used in anything but the smallest deployments. This dependency hasn't been a problem for companies that have already deployed Active Directory when they started using Windows SharePoint Services 2.0 and Windows SharePoint Portal Server 2003 for building Intranet-based solutions.

Using Windows SharePoint Services 2.0 and Windows SharePoint Portal Server 2003 has been more challenging for companies building extranet-based solutions. The tight coupling of Windows SharePoint Services 2.0 to Active Directory forces companies to create and maintain domain accounts for external users such as vendors and customers. It can also be said that Windows SharePoint Services 2.0 has gained a reputation as a product that isn't appropriate for Internet-facing sites because most companies refuse to create Active Directory accounts for unknown users coming to the site for the first time from across the Internet.

All this changes with Windows SharePoint Services 3.0 because authentication has been redesigned on top of the new authentication provider infrastructure introduced with ASP.NET 2.0. If you don't want to maintain user accounts for Windows SharePoint Services 3.0 and Office SharePoint Server 2007 sites inside Active Directory, you can build or acquire an ASP.NET authentication provider that's been designed to store and manage user accounts in a different identity repository.

For example, ASP.NET 2.0 ships with the forms authentication provider that allows you to maintain user accounts in a SQL Server database. This authentication provider can be configured for use in a Windows SharePoint Services 3.0 site. With little effort on your part, you can put a Windows SharePoint Services 3.0 site on the Internet that allows unknown users to register themselves as members. ASP.NET 2.0 provides you with convenient support for creating and maintaining user accounts and even allows users to change and reset their passwords.

Summary

This chapter covered many of the significant advancements to Windows SharePoint Services 3.0 that were targeted at developers. Once you to begin to dig in and work with Windows SharePoint Services 3.0, you will discover there are even more that could not be covered here. However, it should be clear that the Windows SharePoint Services 3.0 team has done a great job of listening to the criticism and feedback from developers about Windows SharePoint Services 2.0 and responding with some really significant advancements.

The next chapter takes a similar pass through new the developer-centric features introduced in Office SharePoint Server 2007. As you will see, there is a great deal of synergy between Windows SharePoint Services 3.0 and Office SharePoint Server 2007. The more you know about Windows SharePoint Services 3.0 development, the more power and control you will have when building custom solutions of top of Office SharePoint Server 2007.

Chapter 2

Building Solutions with Office SharePoint Server 2007

By **Ted Pattison**

With permissions to reprint from *MSDN Magazine*

In this chapter:

Building Office SharePoint Server

2007 Portal Sites	23
Web Content Management	30
Business Intelligence Features	35
Managing Documents and Business Processes	37
Enterprise Content Management	39
Summary	40

Microsoft Office SharePoint Server 2007 provides portal and search features that represent the next generation of Microsoft Office SharePoint Portal Server 2003. However, Office SharePoint Server 2007 is much more than that. The Office team has made significant investments in many other areas to give Office SharePoint Server 2007 extra dimensions in Web content management, business intelligence, business process management, and enterprise content management, as shown in Figure 2-1 on the next page. This chapter describes the basic architecture of Office SharePoint Server 2007 and explores the various opportunities for developers building portal sites and business solutions.

One of the most appealing aspects for developers is that Office SharePoint Server 2007 is built on top of Windows SharePoint Services 3.0 and ASP.NET 2.0. Office SharePoint Server 2007 portal sites are created using Windows SharePoint Services features and site definitions, and familiar ASP.NET building blocks such as master pages and Web Parts. This means that Office SharePoint Server 2007 solutions can easily be extended with ASP.NET components such as server-side controls and custom Web Parts as well as custom Windows SharePoint Services features containing things like custom list definitions, document libraries, event handlers, and workflows.

It's important to understand the extent of the synergy that exists between Office SharePoint Server 2007 and Windows SharePoint Services 3.0. Many of the new features in Office

SharePoint Server 2007 are made possible by new Windows SharePoint Services innovations described in Chapter 1, such as master page integration, content types, versioning, and workflows.

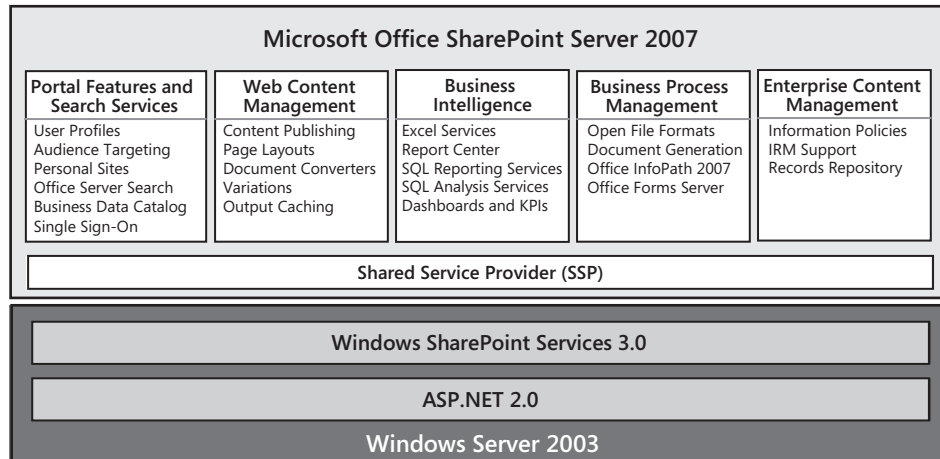


Figure 2-1 Office SharePoint Server 2007 builds its functionality on top of ASP.NET 2.0 and Windows SharePoint Services 3.0.

As a developer, it's also important to set your expectations accordingly about what you'll need to do to get an Office SharePoint Server 2007 solution up and running. Unlike many other development projects, you don't usually start by firing up Microsoft Visual Studio. In some scenarios you can get an entire business solution up and running without ever writing any custom code. This is quite different from developing directly on the ASP.NET platform by itself where you need Visual Studio to put together the simplest site with a couple of pages and a navigation infrastructure.

The Office team promotes a philosophy in which you begin building an Office SharePoint Server 2007 business solution with an “assemble and configure” mentality. Once you determine the Office SharePoint Server 2007 features and services you want to utilize, you create a new portal site and do as much as possible by configuring services through browser-based administrative pages. If you are going to succeed at building Office SharePoint Server 2007 solutions, it's important to embrace this “assemble and configure” mindset to get as far as you can using out-of-the-box features.

However, there will inevitably be times when you determine that the out-of-the-box experience provided by Office SharePoint Server 2007 doesn't cut it for a particular business solution. Then you can put your developer's hat back on and crank up Visual Studio 2005 to create many of the custom component types that are discussed throughout this chapter.

Building Office SharePoint Server 2007 Portal Sites

Companies and developers can expect to find all the portal features in Office SharePoint Server 2007 that they have become accustomed to in SharePoint Portal Server 2003, such as user profiles, audience targeting, personal sites (that is, My Sites), search, and single sign-on. Office SharePoint Server 2007 also includes several new portal features, like the Business Data Catalog.

If you have a background in building portal solutions on top of SharePoint Portal Server 2003, you will find that Office SharePoint Server 2007 is different in many ways. For example, SharePoint Portal Server provides its own separate administrative Web application for provisioning and configuring portal sites. Office SharePoint Server 2007, on the other hand, doesn't require its own separate administrative application. Instead, Office SharePoint Server 2007 integrates its administrative features and configuration links into the Windows SharePoint Services Central Administration application using the light-up features of Windows SharePoint Services 3.0.

Office SharePoint Server 2007 is also quite different from SharePoint Portal Server under the hood. SharePoint Portal Server builds its portal site infrastructure around the concepts of "areas" and "listings." However, areas in SharePoint Portal Server 2003 represent a strange and undocumented layer on top of Windows SharePoint Services 2.0 that has proved hard for developers to extend using standard Windows SharePoint Services development techniques. The concepts of areas and listings from SharePoint Portal Server 2003 have been eliminated in Office SharePoint Server 2007 and replaced with a portal infrastructure that was designed and implemented much more in line with Windows SharePoint Services best practices.

An Office SharePoint Server 2007 portal site is a Windows SharePoint Services site collection containing a top-level site along with several child sites below it. Unlike SharePoint Portal Server 2003, a portal site does not have to be created at the root of an IIS Web site. This provides more flexibility because you can host hundreds of portal sites inside a single IIS Web site. Remember that in Windows SharePoint Services 3.0 terminology, an IIS Web site extended with Windows SharePoint Services functionality is known as a Web application.

You create a new Office SharePoint Server 2007 portal site the same way that you create any other new site collection through the Windows SharePoint Services Central Administration application, the Stsadm.exe command-line utility, or through custom code. When creating a new Office SharePoint Server 2007 portal site, you use one of the portal site templates created by the Office team. Here are some examples of site templates that can be used to create a new Office SharePoint Server 2007 portal site:

- Corporate Internet Site
- Internet Presence Web Site
- Publishing Site

The Office team has provided new site templates to create child sites within a portal site collection such as Report Center and Search Center. The Office team has also evolved some of the older SharePoint Portal Server 2003 site templates for use within Office SharePoint Server 2007 portal site collections such as News, Topics, and the Sites Directory.

Shared Service Providers

The architecture of Office SharePoint Server 2007 is based on Shared Service Providers (SSPs). An SSP represents a set of services that can be configured a single time and shared across many different Office SharePoint Server 2007 portal sites and Windows SharePoint Services sites. Understanding SSPs is critical to being able to take advantage of Office SharePoint Server 2007 features and services.

After you have finished installing Office SharePoint Server 2007 in a farm, you must explicitly create and configure one or more SSPs to take advantage of Office SharePoint Server 2007 features such as user profiles, audiences, personal sites, Excel Services, the Business Data Catalog, and search. After you have initially created an SSP through the Windows SharePoint Services Central Administration pages, you can then configure the individual Office SharePoint Server 2007 services you need from the main SSP administration page shown in Figure 2-2.

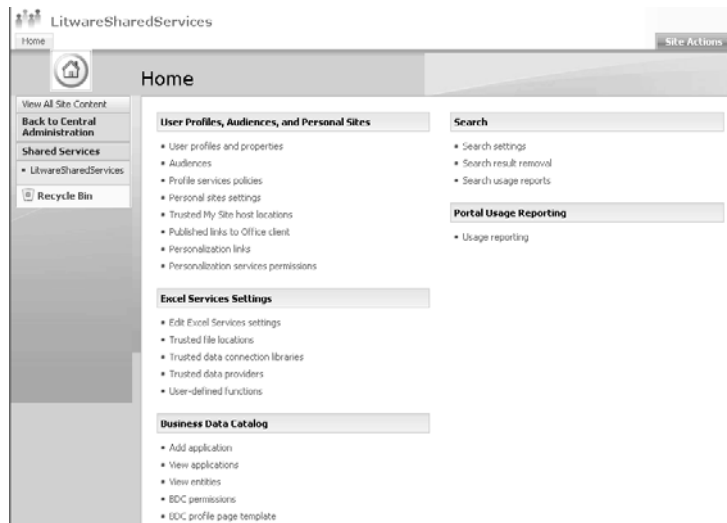


Figure 2-2 An SSP allows you to configure Office SharePoint Server 2007 services.

This new SSP architecture was designed to replace the Shared Services infrastructure of SharePoint Portal Server 2003 to provide greater flexibility in deployment and configuration. For example, it's possible to create two different SSPs within the same farm and configure them differently, as shown in Figure 2-3.

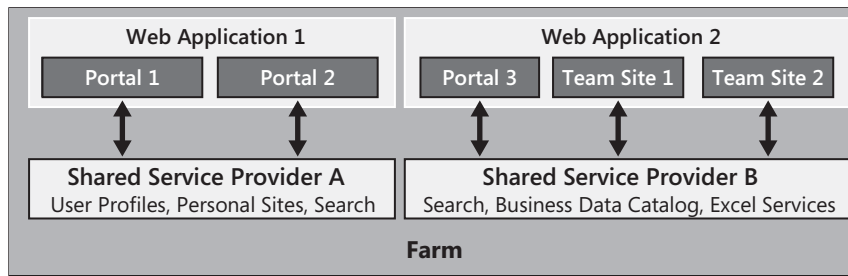


Figure 2-3 Each Web application is associated with a Shared Service Provider.

Each Web application in an Office SharePoint Server 2007 farm along with its Office SharePoint Server 2007 portal sites and Windows SharePoint Services sites is associated with exactly one SSP. One Web application can be associated with one SSP while a different Web application can be associated with a second SSP. The search results from within one portal site might be very different from the search results within another portal site if they are associated with different SSPs that have been configured to have different content sources.

User Profiles

User profiles are an Office SharePoint Server 2007 feature that has been carried over from SharePoint Portal Server 2003 to track information about the users within an organization. User profiles allow users to discover and learn more about the people they work with and to see how everyone fits into their company's organization chart. User profiles also provide the foundation for other Office SharePoint Server 2007 features such as audience targeting and personal sites.

Once you have configured the user profile service through an SSP, Office SharePoint Server 2007 stores the data for user profiles in Microsoft SQL Server. Office SharePoint Server 2007 makes it possible to import and synchronize user profile data from external sources such as an Active Directory domain as well as other LDAP-based identity management systems.

Office SharePoint Server 2007 provides several standard pages and Web Parts out of the box to display the information tracked within user profiles. Figure 2-4 on the next page shows an example of how Office SharePoint Server 2007 displays a graphic representation of an organization chart on a personal site from user data imported from Active Directory. Note also that the Office SharePoint Server 2007 object model provides an API to read and modify user profile data from components such as Web Parts, event handlers, and custom workflows.

As with SharePoint Portal Server 2003, user profiles can be extended with custom properties to track user data for domain-specific business solutions. However, Office SharePoint Server 2007 enhances custom properties by allowing for multivalued properties and properties defined with open or closed vocabularies.

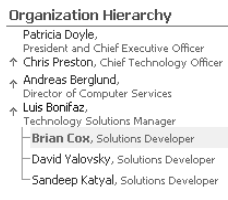


Figure 2-4 A user profile can be used to show a company's organization chart.

Audience targeting is another valuable Office SharePoint Server 2007 feature that has been carried over from SharePoint Portal Server 2003. You can define an audience by specifying criteria to define a subset of users. For example, you can create a Sales audience that is defined as all the users who are members of an Active Directory group named Sales. You could define another audience named Carpool as the set of users with the custom *Carpool* property's value set to True.

Once you have defined an audience, you can then configure a Web Part to conditionally display its content only when the current user is a member of that audience. This makes it straightforward to target content to those users who need it while hiding that content from users who would find it distracting. Audience targeting is also a great way to show privileged users links to secured pages while hiding these links from unprivileged users who would receive Access Denied errors when attempting to follow them.

One more feature carried over from SharePoint Portal Server 2003 that is built on top of user profiles is personal sites. When personal sites are enabled within an SSP, Office SharePoint Server 2007 will provide each user with their own personal site (that is, a My Site) that is provisioned on demand the first time it is used. Personal sites enable users to edit certain aspects of their user profile to better describe themselves to their coworkers. A personal site has a public aspect, which makes it easy for a user to share data and documents with other users within the organization. A personal site also has a private aspect, allowing a user to store data and documents that are not meant to be shared.

Office SharePoint Server 2007 Search

Many people feel that the search feature is one of the most valuable aspects of Office SharePoint Portal Server 2003. Office SharePoint Portal Server 2003 Search makes it possible to search through not only content and documents within Office SharePoint Portal Server portal sites and Windows SharePoint Services team sites but also through external content such as Windows file shares, public Microsoft Exchange Server folders, and standard Web sites. Office SharePoint Server 2007 Search was designed to give you these same features in a manner that is more performance-oriented and easier to configure.

With the previous version of SharePoint Technologies, Windows SharePoint Services 2.0 and SharePoint Portal Server 2003 each used a different underlying infrastructure to support

indexing and search. This created problems as companies upgraded from Windows SharePoint Services 2.0 to SharePoint Portal Server 2003. These upgrade problems have been addressed, as the Windows SharePoint Services Search Service and Office SharePoint Server Search Service are now based on the same underlying indexing and search infrastructure, which is an evolved version of what was provided by SharePoint Portal Server 2003.

So what are the most significant differences between Office SharePoint Server 2007 Search and Windows SharePoint Services 3.0 Search? Office SharePoint Server 2007 Search makes it possible to search external content across the network such as Windows file shares and BDC data while Windows SharePoint Services 3.0 Search is limited to searching through content and documents within the current site collection. Also, Office SharePoint Server 2007 Search can be configured to run the indexing service and search service on different servers within a farm to increase scalability and throughput. Windows SharePoint Services 3.0 Search is limited to running the indexing service and search service on the same physical server.

Configuring Office SharePoint Server 2007 Search is an administrative exercise in creating and configuring content sources within the scope of a particular SSP. A content source defines a set of searchable content. When you create a new SSP, Office SharePoint Server 2007 automatically creates a content source to search through user profile data as well as the content and documents within Office SharePoint Server 2007 portal sites and Windows SharePoint Services sites within the Web applications associated with the current SSP. However, the SSP administrator must explicitly create and configure additional content sources to support building indexes and searching through external content such as documents in a Windows file share or content from a public Web site.

Windows SharePoint Services 3.0 and Office SharePoint Server 2007 provide a rich user interface for searching by adding search boxes and search result pages to each Windows SharePoint Services site and each Office SharePoint Server 2007 portal site. Office SharePoint Server 2007 goes even further to supply a dedicated child site named Search Center within a portal site collection that provides a specialized user interface for searching, as shown in Figure 2-5 on the next page.

From what you have just seen, you could conclude that companies can take full advantage of Office SharePoint Server 2007 Search facilities without ever writing any custom code. However, Office SharePoint Server 2007 makes it possible for you to customize how search results look by modifying the XSLT transforms it uses to display search results. Office SharePoint Server 2007 also exposes its search engine through a programmable API that enables developers to extend either the Windows SharePoint Services 3.0 Search Service or the Office SharePoint Server 2007 Search Service using custom code.

For example, you can write a server-side component such as a Web Part or a custom workflow that queries the Office SharePoint Server 2007 Search Service programmatically and deals with the returned search results in a customized fashion. Likewise, you can create a Windows

Forms application that queries the Office SharePoint Server 2007 Search Service from across the network through a built-in set of Web Services that ship as part of Office SharePoint Server 2007.

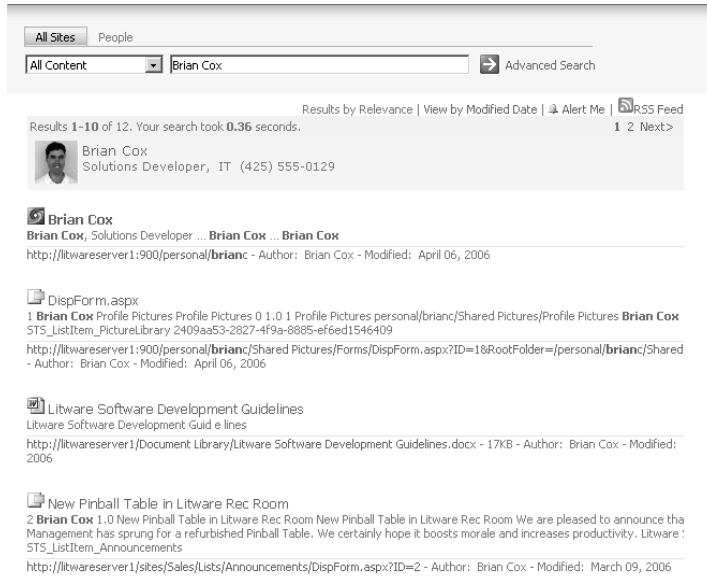


Figure 2-5 Search Center provides a sophisticated UI for user-initiated searches.

The Business Data Catalog

The Business Data Catalog (BDC) is a new innovative framework created by the Office team to provide Office SharePoint Server 2007 portal sites and standard Windows SharePoint Services 3.0 sites with integration into back-end line-of-business systems such as those created by SAP, Siebel, and PeopleSoft. The BDC additionally provides the means to integrate data directly from database systems such as SQL Server and Oracle.

While SharePoint Portal Server 2003 makes it possible to integrate portal sites with back-end systems, it requires you to write custom code to manage connections and retrieve the data you need to display. Furthermore, the code you must write changes significantly as you switch between back-end systems from vendors such as SAP and PeopleSoft. The Office team designed the BDC to make things much easier.

The BDC enables you to integrate data from back-end systems without requiring you to write custom code for managing connections and retrieving data. The design of the BDC is based on standardized metadata that describes the location and format of a back-end system and data entities defined inside. The BDC also provides an execution component capable of reading BDC metadata that is able to retrieve external data from any back-end system and return it to Office SharePoint Server 2007 in a standard format.

Figure 2-6 shows the high-level architecture of the BDC. As you can see, connectivity between the BDC and traditional line-of-business systems is achieved using standard Web services. Connectivity between the BDC and database systems is achieved using ADO.NET providers.

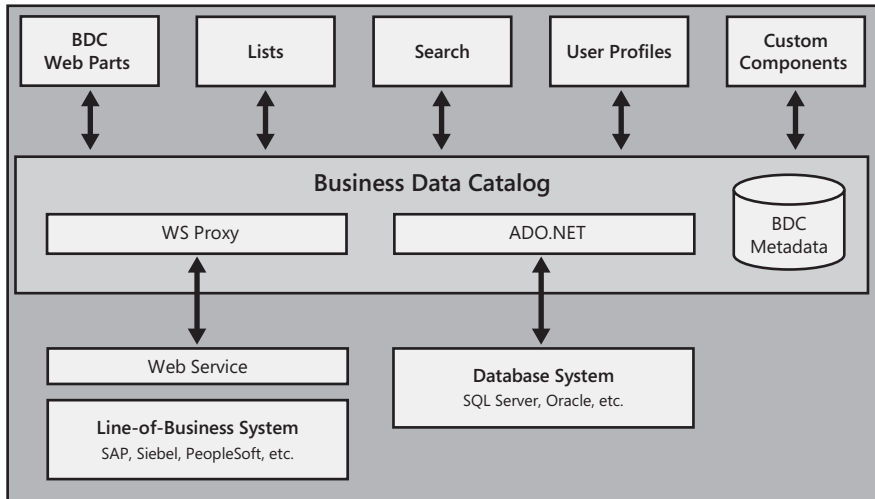


Figure 2-6 The BDC provides seamless access to data in back-end line-of-business systems and databases.

The first step in using the BDC is to author an XML file containing the metadata to connect to a back-end system. When you author metadata for the BDC, you define the data you want to retrieve in terms of entities. For example, you might define a customer as one entity and an invoice as another entity. The BDC metadata format also lets you define associations between entities in scenarios when there is a one-to-many relationship such as one that might exist between customers and invoices.

The definition of a BDC entity contains identifiers, properties, and methods. The methods define how the BDC interacts with entry points exposed by the back-end system. For a back-end system accessible through Web services, methods define the names of the Web service operations and the parameters required to call them. For a back-end system that is a database such as SQL Server or Oracle, methods define the names of stored procedure and SQL statements.

Entities can also define actions. A BDC action is used to dynamically parse together the URL behind a hyperlink that allows a user to navigate from a page in an Office SharePoint Server 2007 portal site to another location. For example, an action defined on a BDC customer entity could be written to redirect users to a Web page in an SAP application that supports updates to customer information. Actions were designed to support scenarios where the BDC is used to display read-only data and to bootstrap the user into another application when updating or some other type of external operation is required.

Once you have authored or acquired the XML file with the required BDC metadata for a back-end system, you must import it into the BDC within the scope of a particular SSP to create what is known as a “BDC application.” You can accomplish this importing process using the SSP administrative Web pages. You can alternatively import an XML file with BDC metadata using custom code written against the BDC administrative object model.

Once you have imported the required metadata to create a BDC application, there are several out-of-the-box techniques to leverage and display its data within a portal site. Office SharePoint Server 2007 ships with a set of Business Web Parts that can be quickly added to pages to query and display BDC data. You can also add new columns to lists and document libraries based on an entity defined in a BDC application. A user editing a column based on a BDC entity is automatically presented with a user interface making it possible to query the back-end system.

The BDC has been designed to integrate with the Office SharePoint Server 2007 Search Service. For example, a back-end system and its entities can be defined as a content source so that the Office SharePoint Server 2007 indexing service will crawl through its data and build indexes for the Office SharePoint Server 2007 search engine. This becomes a powerful feature because it allows users to discover data from back-end systems about things like customers and invoices when running standard search queries through both Office SharePoint Server 2007 portal sites and standard Windows SharePoint Services 3.0 sites.

The BDC provides convenient features to map data from a BDC entity to properties in a user profile and to synchronize this data at periodic intervals. For example, if your company has an SAP system that contains employee data you like to include in user profiles, such as phone numbers or social security numbers, you can configure this type of data importing without writing any custom code.

Finally, BDC entities can also be accessed programmatically using custom code written against the BDC object model. This makes it possible to write custom Web Parts as well as other server-side components and services that run their own BDC queries. One nice aspect of writing code to query BDC entities is that you don’t have to worry about managing connections or whether you are accessing the back-end system through Web services or ADO.NET. All those details are abstracted away by BDC metadata and the BDC execution engine.

Web Content Management

Over the past few years, Microsoft’s strategy for publishing content on the Web has been based on Microsoft Content Management Server 2002 (CMS). CMS provides a structured way for content authors to add content to a company’s public Web site using professionally formatted layout pages. CMS also provides a formalized scheme where a privileged user must approve any page modification before it can be seen by the Web site’s visitors.

In the past, many Microsoft customers have had to choose between CMS and SharePoint Portal Server 2003. While there is a connector that provides a certain degree of integration between CMS and SharePoint Portal Server 2003, these two products are built on very different architectures. This has resulted in frustration because you cannot build a site that fully benefits from both the CMS Web content management features and the SharePoint Portal Server 2003 portal features.

Microsoft made an important decision to discontinue evolving CMS as a stand-alone product and to migrate CMS Web content management features and the CMS customer base over to Office SharePoint Server 2007. This will obviously have a significant impact on customers who have already become familiar with CMS development. However, the good news is that Microsoft's Web content management strategy is now built on Windows SharePoint Services 3.0 and Office SharePoint Server 2007. That means you can mix Microsoft's best-of-breed Web content management features with its portal features in an Office SharePoint Server 2007 portal site.

If you have worked with CMS in the past, it's important to note that the CMS concepts of channels and postings are not used in the Office SharePoint Server 2007 Web content management infrastructure. Instead, the infrastructure has been designed using basic Windows SharePoint Services 3.0 building blocks such as child sites, page templates, content types, document libraries, and security groups. This newer approach lends itself to building custom solutions that extend the basic Web content management infrastructure using standard Windows SharePoint Services components such as custom event handlers and workflows.

When you need to brand an Office SharePoint Server 2007 portal site, you can modify a single ASP.NET master page to customize the basic look and feel of the entire Web site just as you would in a standard Windows SharePoint Services 3.0 site collection. However, Office SharePoint Server 2007 extends Windows SharePoint Services 3.0 by introducing a publishing scheme based on page layouts. A page layout provides a structured approach to collecting content from content authors and displaying it on a page within a portal site.

Examples of some of the page layouts provided by Office SharePoint Server 2007 out of the box include welcome pages, articles, and news items. An example of a page layout as seen by a content author in editing mode is shown in Figure 2-7 on the next page. As you can see, page layouts are designed to make it fairly straightforward to add and modify content from within the browser. You might also notice in Figure 2-7 that Office SharePoint Server 2007 provides a toolbar within the browser to give content authors and approvers a convenient way to move content pages through the editing and approval processes.

Each page layout is based on a Windows SharePoint Services 3.0 content type and an associated .aspx page template. By layering page layouts on top of content types, Office SharePoint Server 2007 makes it possible to add custom fields for storing different types of structured content such as HTML, links, and images. Once a custom field is defined inside the content

type associated with a page layout, it can be data-bound to the associated .aspx page template using another new Office SharePoint Server 2007 component known as a field control.

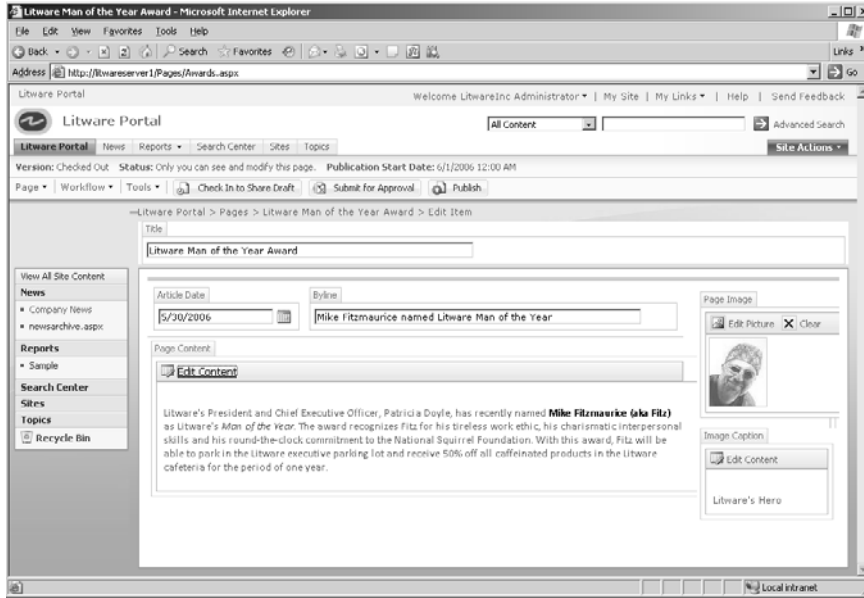


Figure 2-7 Page layouts provide an editing mode for content authors.

Office SharePoint Server 2007 ships with several field controls such as a rich HTML editor as well as others field controls for editing custom fields based on images and links. Many field controls also support adding extra declarative constraints to keep portal content within a highly structured format.

Note that the page layout infrastructure is very extensible because Windows SharePoint Services content types support inheritance. It's fairly straightforward to take one of the built-in page layouts and customize it by extending its underlying content type or .aspx page template.

In addition to field controls, an .aspx page template for a page layout can also contain ASP.NET server controls and Web Part zones. A page layout with Web Part zones provides the content author with the flexibility to add Web Parts displaying content outside the schema of the current page layout. In addition, Office SharePoint Server 2007 provides several Web Parts that have been designed for use in portal pages including the Table of Contents Web Part and the Content Query Web Part.

The .aspx page templates associated with page layouts are stored along with the portal site's master page in the Master Page Gallery. The Master Page Gallery also contains a metadata column to associate each .aspx page template with a content type. Note that it is possible to have multiple page layouts, each with its own .aspx page template, that are all associated with the

same content type. This is useful when you want to create several different views for the same set of structured content.

Whenever a content author creates a new content page from a page layout, Office SharePoint Server 2007 creates a new instance of the associated content type and stores it in a document library named Pages. When a content author updates content for custom fields within a page layout, Windows SharePoint Services stores the data within a structure defined by the underlying content type. The fact that content page instances are stored in a Windows SharePoint Services document library means that the Office SharePoint Server 2007 Web content management infrastructure can take advantage of basic document library features provided by Windows SharePoint Services 3.0 such as versioning, auditing, approval, and workflows as well as per-document security configuration and security UI trimming.

By default, Office SharePoint Server 2007 uses the basic document approval features of a Windows SharePoint Services document library to control when the updated content is shown to the site's visitors. However, the infrastructure was designed to make it straightforward to associate custom workflows with the Pages document library for scenarios where you need something more sophisticated than the Office SharePoint Server 2007 content approval functionality that comes out of the box.

Note that an instance of a content page stored in the Pages document library does not represent a copy of the page template. Instead, it contains redirection logic to link it to the .aspx page template at run time. That means updating the .aspx page template will always affect content pages that have already been created from the associated page layout.

A number of other Office SharePoint Server 2007 features focus on Web content management. I will conclude this section by quickly exploring what these features are and why they are valuable.

Office SharePoint Server 2007 provides a framework for document converters. A document converter is a component designed to read content from an external format such as a Microsoft Word document and convert it into a format that can be displayed within an Office SharePoint Server 2007 content page. Several document converters will ship with Office SharePoint Server 2007 as well as a framework for building and integrating custom document converters.

Office SharePoint Server 2007 provides content deployment features that allow you to transfer content from one site collection to another. This is valuable for companies that prefer to author content in a staging environment before moving it into their production environment. You take advantage of Office SharePoint Server 2007 content deployment features by configuring paths and jobs.

A path defines one site collection as a content source and another site collection as a content destination. Once you have defined a path, you can define one or more jobs to move content

from the source to the destination. Jobs can be run on demand or they can be scheduled to run at a future time or on a periodic basis.

Office SharePoint Server 2007 also supports a feature known as site variations for companies that need to duplicate a site's content for translation into multiple spoken languages or for targeting different types of rendering devices. For example, imagine you have configured Office SharePoint Server 2007 variation support for German and French in addition to Spanish. Office SharePoint Server 2007 maintains a parallel structure across these three different sites with respect to pages and child sites.

When a content author adds a new page to the master variation site maintained in Spanish, Office SharePoint Server 2007 automatically adds the same page into the structure of the other sites as well. Office SharePoint Server 2007 can also be configured to create a Windows SharePoint Services task marking the required translation as a to-do item for a language translator. While Office SharePoint Server 2007 will not actually convert your content from one spoken language to another, it does keep multiple sites in sync with respect to their content structure, which provides a good deal of value.

Office SharePoint Server 2007 provides several caching options. While Office SharePoint Server 2007 doesn't allow you to use ASP.NET output caching directives the same way you do in a standard ASP.NET page, it provides a more sophisticated framework to reach the same end.

You can enable Office SharePoint Server 2007 output caching at the site collection scope. When using these caching features, you configure caching profiles to control caching page items and complete pages in memory. Developers should take note that Office SharePoint Server 2007 supplies dedicated caches for navigation nodes and content returned from potentially expensive retrieval operations such as standard Windows SharePoint Services queries run using an *SPQuery* object and cross-site queries run using a SharePoint Portal Server *SPSiteDataQuery* object.

Office SharePoint Server 2007 also supports Web front-end (WFE) disk caching. If you enable the WFE disk cache, Office SharePoint Server 2007 will begin writing the large files it retrieves from the SQL Server database into a special cache on the local file system of the front-end Web server. This eliminates the need to continually move .jpg, .png, .gif, .css, and .js files from the SQL Server database server to front-end Web servers on a per-request basis.

Finally, it's important to note that Office SharePoint Server 2007 publishing sites benefit from Windows SharePoint Services 3.0 security advancements. In particular, Windows SharePoint Services 3.0 is built on top of the ASP.NET 2.0 authentication provider infrastructure. Unlike SharePoint Portal Server 2003, which is tightly coupled to Active Directory, you can configure an Office SharePoint Server 2007 portal site to use forms-based authentication. That means you can store the user credentials in a SQL Server database or another LDAP identity management system of your choosing.

Business Intelligence Features

In the past, the Office team has provided its customers with business intelligence (BI) features in SharePoint Portal Server 2003, the Office 2003 Web Parts and Components Add-in, and Microsoft Office Business Scorecard Manager 2005. Over the past few years, many developers have used these BI products as a platform for building dashboard-style applications that provide upper-level management with up-to-date data that reflects the health of a business and flags potential problems in a timely matter.

Leveraging their previous experience with these earlier BI components, the Office team has designed Office SharePoint Server 2007 to include a next-generation platform for building dashboards and integrating with other technologies in the greater Microsoft BI picture such as Microsoft Office Excel 2007, Microsoft SQL Server Reporting Services, and Microsoft SQL Server Analysis Services. As with all other aspects of Office SharePoint Server 2007, its BI platform builds on top of ASP.NET and Windows SharePoint Services 3.0 and provides many opportunities for extending what comes out of the box.

The Office team has heard consistent customer feedback telling them that a large percentage of corporations maintain a significant amount of business logic in Excel spreadsheets and that this business logic has been hard to leverage and reuse across a large organization. This feedback led the Office team to create Microsoft Office SharePoint Server 2007 Excel Services.

Excel Services represents a server-side version of the traditional Excel calculation engine that has been rewritten from the ground up on top of Windows SharePoint Services 3.0. Excel Services doesn't suffer from the same types of scalability problems that occur when you run the desktop version of Microsoft Excel on the server.

Excel Services also provides a server-side rendering engine that can display spreadsheets in the browser as HTML. That means a company can store all its Excel spreadsheets in a centralized document library and make them viewable by users who don't even have Excel installed on their desktop. Furthermore, users can see the numbers displayed by a spreadsheet within the browser without having any access to the business logic behind it that represents a company's intellectual property.

A key observation is that the Microsoft Office 2007 system introduces a new paradigm that recognizes that companies maintain business logic within Excel spreadsheets just as they maintain business logic within managed code inside compiled assemblies and within stored procedures in a SQL Server database. To support this new paradigm, the Office team has added many new features to Office 2007 products designed to expose and update this business logic as well as to protect its intellectual property from prying eyes.

The new desktop version of Office Excel 2007 has been enhanced to allow information workers with Excel expertise to publish and update their spreadsheets in a document library on an Office SharePoint Server 2007 portal site or a Windows SharePoint Services team site. Users

running a version of Excel can view these spreadsheets through a rich client experience while other users can rely on Excel Services to view the same spreadsheet inside the browser.

Note that this new spreadsheet publishing metaphor allows a company to maintain a single master copy of its critical spreadsheets. It also allows the spreadsheet author to post updates without the need to involve the development staff or the IT staff.

It's important to note that the use of Excel Services isn't restricted to the browser. You can create a Windows Forms application that leverages the server-side Excel calculation engine but doesn't use the rendering engine. For example, a Windows Forms application can use out-of-the-box Web services from Excel Services to load a spreadsheet on the server, enter input data, perform calculations, and return a result. This example furthers the analogy that Excel Services exposes the business logic defined in a spreadsheet just as SQL Server exposes the business logic defined in a stored procedure.

Office SharePoint Server 2007 provides a special site template named Report Center for companies that want to build dashboard-style applications. Report Center was designed to make the new BI features of Office SharePoint Server 2007 easy to discover and use. A Report Center site contains a document library named Reports Library that is tuned for storing and displaying BI reports such as Excel spreadsheets and reports built for SQL Reporting Services.

Another important aspect of Report Center is the built-in support for creating and importing Key Performance Indicators (KPIs). A KPI is a common term used in the BI space to describe a visual indicator that tells a manager how some aspect of the business is doing. For example, the KPI for a product inventory level might display a green light when there is enough inventory to supply all the orders for the coming week.

However, the light might turn from green to yellow when the inventory level drops below some predefined threshold such as the amount of inventory required to supply orders for the next four days. The light then might change from yellow to red when the inventory level drops to a point where it will run out within the next 48 hours. The main idea is that a KPI flags business problems that require immediate attention.

Office SharePoint Server 2007 provides out-of-the-box support for creating several different types of KPIs. For example, you can create a KPI whose indicator changes automatically depending on data it dynamically reads from a Windows SharePoint Services list or an Excel spreadsheet. Office SharePoint Server 2007 also provides integration support for KPI in SQL Server 2005. That is, if you have already created KPIs with SQL Server Analysis Services, you can import and display them on a Report Center site alongside other types of supported KPIs.

The last aspect of Report Center I want to discuss is the built-in framework for filtering data before it's shown to the user. This is a key component of the Office SharePoint Server 2007 dashboard framework because it makes dashboard pages more relevant to the user.

When a manager visits a Report Center site, the experience is enriched if the dashboard views have been customized with data that is relevant for that user. For example, a sales manager for

the Eastern region can be presented with a different view of sales figures than the sales manager for the Western region. Furthermore, managers like to be able to see high-level data at first and then be able to drill down into more specific categories on demand.

Filtering support is built into Office SharePoint Server 2007 dashboards at the page level using Web Part connections. Office SharePoint Server 2007 supplies Web Parts out of the box that allow page designers and users alike to specify criteria such as the name of the current user, a date range, or a product category. There are also many out-of-the-box Web Parts that can be configured to consume the filtering criteria supplied by other Web Parts such as the standard Windows SharePoint Services List View Web Part as well as the Web Parts designed for use with the Business Data Catalog, Excel Services, SQL Reporting Service, and SQL Analysis Services.

Managing Documents and Business Processes

In past versions of Microsoft Office Word, Excel, and PowerPoint, Microsoft has relied on a default file structure that is based on binary files written in a proprietary format. These formats have been very hard to read and modify unless you go through the object model of the hosting Office application such as Word or Excel. As a result, companies have tried to run Office desktop applications on the server, which poses serious problems with scalability and robustness.

Office 2000 and Office 2003 added some modest capabilities for creating Excel spreadsheets and Word documents using XML. In the 2007 Microsoft Office system, Microsoft has taken this idea much further by adopting the Open XML Formats for Word, Excel, and PowerPoint documents. The Open XML Formats is a new file standard for creating composite documents containing multiple inner XML files that factor out content from other aspects of the document such as formatting instructions, data, and code.

The top-level file in the Open XML Formats is known as a package and it is structured using standard ZIP file technology. The internal files contained within a package are known as parts. Many parts within Word, Excel, and PowerPoint files contain XML structured in accordance with published XML schemas. Other parts within a package can consist of binary files for items such as graphics, audio clips, and videos.

A major goal of the Open XML Formats is to provide a standard approach for reading, manipulating, and generating documents in server-side scenarios where using the object model of a desktop application such as Word or Excel isn't a viable option. Think about a scenario in an Office SharePoint Server 2007 portal site when you have created and configured an event handler to fire whenever someone uploads a new Word document. The new Open XML Formats make it significantly easier to extract data or to perform hygiene on the document such as removing comments and personal information. You can also leverage Open XML Formats to develop server-side components that generate Office documents on the fly using data pulled from content sources such as a Windows SharePoint Services list or the Business Data Catalog.

To get started working with the Open XML Formats, you need to learn how to program against the new WinFX packaging API. This is the recommended API to use when opening and creating packages. You must also learn the specific package structure and XML schemas for the type of Office document you are working with. These details will change as you move among Word, Excel, and PowerPoint documents. Microsoft has started a developer community effort around the Open XML Formats at <http://openxmldeveloper.org>.

Office Forms Server 2007

Since the introduction of Microsoft Office 2003, many companies have found that Microsoft Office InfoPath provides a quick and efficient solution for creating electronic input forms to collect XML-based data from users. The InfoPath 2003 forms designer provides high levels of productivity because it significantly reduces or eliminates the need to write custom code when developing forms. InfoPath also creates a very reliable way to collect data because InfoPath forms are built on top of XML schemas, allowing them to automatically validate the user's input.

InfoPath 2003 provides a convenient integration point with Windows SharePoint Services 2.0 where a form designer can publish an InfoPath form on a Windows SharePoint Services site, creating what is known as a forms library. A forms library is a hybrid Windows SharePoint Services document library that uses an InfoPath form template as its underlying document template and acts as a repository for XML documents containing form data entered by users.

One common complaint that Microsoft has received about this forms-based technology is that it requires a full version of InfoPath to be installed on the user's desktop. This is true not only for those designing forms but also for any users who need to read or modify data in an InfoPath form. The problem is that the features of InfoPath 2003 do not extend to users for whom deploying Office 2003 on their desktop isn't an available option. Office SharePoint Server 2007 introduces Microsoft Office Forms Server 2007 to solve this problem.

Office Forms Server 2007 has been designed to render InfoPath forms within the browser to reach users who are not running InfoPath. In fact, Office Forms Server 2007 doesn't even require users to be running a version of Microsoft Internet Explorer or the Windows operating system. The Office team is testing Office Forms Server 2007 for compatibility with browsers such as Firefox, Safari, and Netscape as well as several other HTML-enabled mobile devices.

You can use the Office InfoPath 2007 forms designer to create Web-enabled forms, which can then be deployed to Office Forms Server 2007. This forms designer provides a compatibility checker to ensure that your forms contain only controls and elements that are compatible with what Office Forms Server 2007 can render to the browser. It's also possible to create and deploy InfoPath forms intended for dual use. Such a form is downloaded to the desktop and loaded into the rich client environment when InfoPath is available on the desktop or otherwise rendered through the browser when necessary.

Enterprise Content Management

Over the past few years, governments and other regulatory organizations have placed increasingly restrictive requirements on companies that generate and store large numbers of documents. The Sarbanes-Oxley Act and emerging privacy laws within the U.S. have made it much harder for companies to stay within acceptable levels of compliance. Office SharePoint Server 2007 provides several enterprise content management features to address these problems.

Many of the Office SharePoint Server 2007 features for enterprise content management are built on top of the new information policy feature. An information policy is a Windows SharePoint Services component that can be enabled and configured within the scope of a list or document library. Out-of-the-box examples of information policies that ship with Office SharePoint Server 2007 include those for document expiration, auditing, and the automatic generation of bar code labels to identify physical documents and associate them with electronic copies maintained in a document library.

The Windows SharePoint Services framework for information policies was designed with extensibility in mind. For example, you can create a custom policy that checks the integrity of a digital signature on every document within a document library. You can create another policy that promotes document hygiene and privacy by removing all the comments and personal information from documents as they are uploaded to a document library.

Office SharePoint Server 2007 provides a dedicated site template named the Records Repository to assist with records management. A Records Repository site provides archiving support for companies that are required to keep certain types of business documents as official records of the company's activities. While archiving requirements vary across different regulated industries, keeping records is required to provide evidence of a company's activities in the event of a litigation dispute or an audit.

The Single Sign-On Service

The final topic I would like to cover is the Single Sign-On (SSO) Service. The SSO that ships with Office SharePoint Server 2007 is an enhanced version of the credential-mapping service that ships with SharePoint Portal Server 2003. The SSO is used to map the identity of a user who has logged on to an Office SharePoint Server 2007 portal site with another identity for the same user in a back-end system.

For example, imagine a user named Bob has two user accounts. First, he has a Windows account in Active Directory that he uses to log on to the local network and to authenticate against the Office SharePoint Server 2007 Web server. Bob also has a second user account with different credentials he needs to use when accessing the company's back-end SAP application.

SSO solves the problem of allowing server-side code running on behalf of Bob to seamlessly access the SAP system using his secondary account once he has logged on to the Office Share-

Point Server 2007 portal site using his primary Active Directory account. SSO accomplishes this by providing a credential-mapping database to store his SAP user name and password in an encrypted format. SSO also provides the means for custom Web Parts and other Office SharePoint Server 2007 services such as the BDC, Excel Services, and Office Forms Services to retrieve user credentials that are required when accessing various back-end systems.

Summary

The chapter has provided a broad survey of the most significant services and features provided by Office SharePoint Server 2007. You have seen that the Office team has invested heavily to provide Office SharePoint Server 2007 functionality in areas such as portal and search, Web content management, business intelligence, business process management, and enterprise content management.

You have also seen that Office SharePoint Server 2007 promotes an “assemble and configure” philosophy that enables you to construct rich business solutions without ever starting Visual Studio or writing a line of managed code. However, you should always remember that Office SharePoint Server 2007 is built on top of ASP.NET 2.0 and Windows SharePoint Services 3.0, providing ample opportunities for extending standard functionality with custom components such as Web Parts, event handlers, workflows, page layouts, document converters, and server-side forms.

Building a Basic SharePoint Site

In this chapter:

Creating a Site Collection and a Top-Level Site	41
Creating a List for Tracking Project Profiles	46
Creating a Document Library	48
Customizing the Home Page	49
Creating Child Sites	51
Creating a “Hello World” Web Part	52

This chapter serves as a bridge to the chapters that follow, in which you’ll see examples that illustrate some of the features, architectural underpinnings, and development possibilities related to Microsoft Windows SharePoint Services 3.0 and Microsoft Office SharePoint Server 2007.

In this chapter, we’ll walk through how to set up and build a basic SharePoint site. The example will let you see some of the administrative pages and other elements of the user interface in the latest version of Windows SharePoint Services. In the example, we’ll create a site collection and a top-level team site, a list for tracking projects, and a document library for storing files related to projects. The site will also include two child sites. You’ll see how to change the look and feel of a SharePoint site and, in the last section of the chapter, you’ll see some sample code for a custom Web Part.

Creating a Site Collection and a Top-Level Site

The first step in creating a site collection and a top-level site is to open the SharePoint Central Administration page. (Choose Start, Administrative Tools, SharePoint 3.0 Central Administration.) On the Central Administration home page, click Application Management at the top of the page, and then, on the Application Management page, under SharePoint Site Management, click Create Site Collection.

After the Create Site Collection page appears, the Select Web Application dialog box (shown in Figure 3-1 on the next page) opens and prompts you to select the target Web application (formerly known as a virtual server) that will be used to provision the new site collection. Choose the default Web site as the target Web application or a different target if necessary.

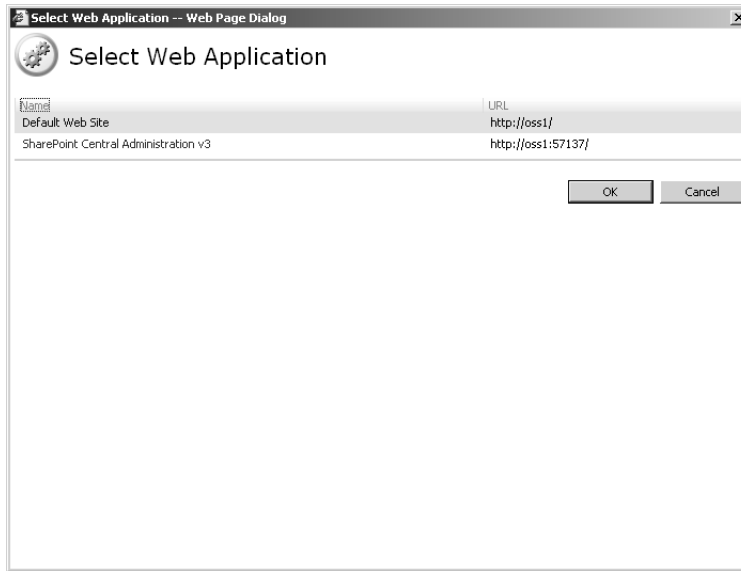


Figure 3-1 Use the Select Web Application dialog box to specify the Web application that will provision the site collection.

Back on the Create Site Collection page, fill in the information required to create a new site collection, using settings similar to the following. When filled in, the Create Site Collection page would look something like Figure 3-2.

- The new site collection's URL (<http://oss1/sites/projectmanagement>).
- The primary site collection owner (`litwareinc\administrator`, with an e-mail address of `administrator@litwareinc.com`).
- The Quota Template option set to No Quota, the default setting.
- The site template that will be used for the sites in the site collection. (Blank Site is used in this example as the site template for the new top-level site that is automatically created.)

After you click OK to create the site collection and top-level site, you'll see a page confirming that the site has been created. You can then navigate to the top-level site by using the browser. Figure 3-3 shows the bare-bones site.

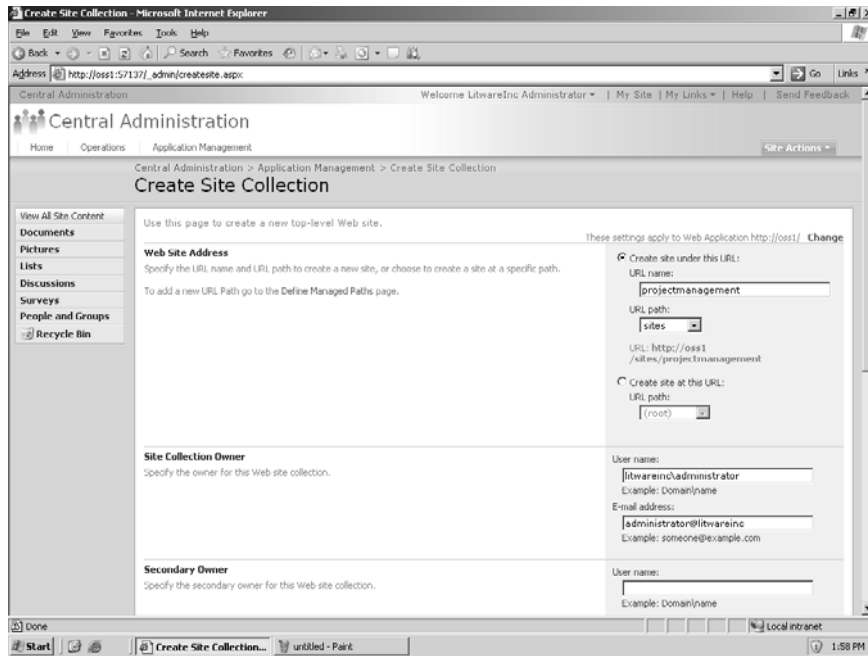


Figure 3-2 You specify information such as this to define a site collection.

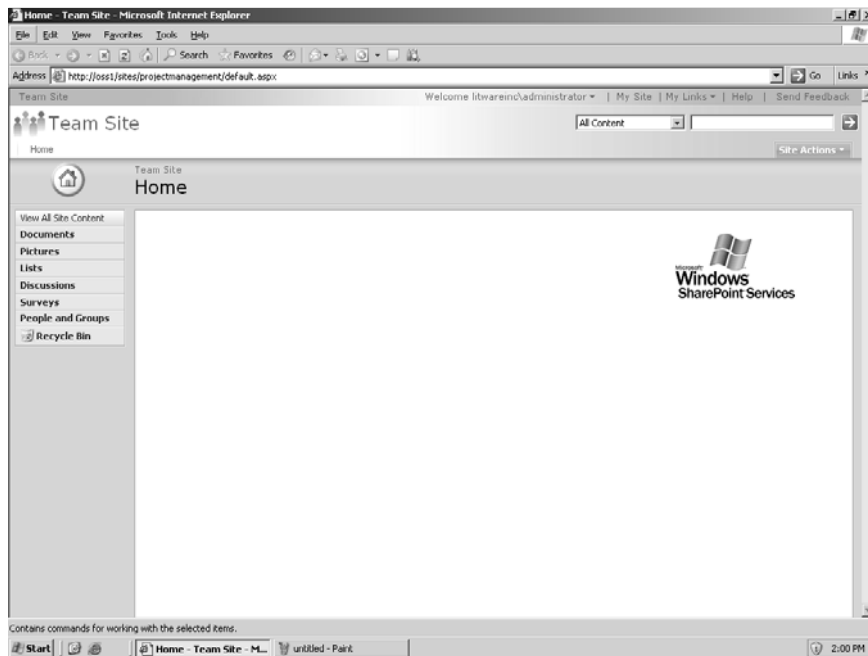


Figure 3-3 The top-level site of the site collection

At this point, we'll make a few straightforward changes to the settings and appearance of the top-level site. For example, we'll change the site's title and add our own graphics.

Start by clicking Site Settings on the Site Actions menu. On the Site Settings page, in the Look And Feel section, click Title, Description And Icon. On the General Settings page, enter the title for the new site: **LW Project Management**.

We can also use this page to point to different graphics files to replace the standard Microsoft graphics. For example, you can modify the site icon to use a custom graphic. The graphics files you want to use for a site should be located within the Windows SharePoint Services `_layouts` directory so that the graphics files are accessible to all SharePoint sites within a farm. In particular, copy the graphics to the following location:

```
C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\
    TEMPLATE\IMAGES
```



Note Windows SharePoint Services configures the IMAGES directory with Internet Information Services so that it is a child virtual directory of the `_layouts` virtual directory. This configuration means that your graphics files should be accessible within any SharePoint site using a relative URL that looks like the following:

```
/_layouts/images/LitwareGraphics/LitwareLogo.png
```

While still on the General Settings page, we can modify the address for the logo image file, entering a value something like `/_layouts/images/LitwareGraphics/LitwareLogo.png`. Click OK on the General Settings page, and you can then confirm that pages within the site now display the new title and the new site icon, as you can see in Figure 3-4.

We can also change the look and feel of the site by modifying the Site Image Web Part on the site's home page so that it displays a custom graphic instead of the default Microsoft graphic. To do this, we place the home page of the site in edit mode by choosing Edit Page from the Site Actions menu. After the page is editable, select Modify Shared Web Part from the Site Image Web Part's action menu. You'll then see a task pane in the browser that allows you to modify the Image Link property. Assign the Image Link property a value such as `/_layouts/images/LitwareGraphics/LitwareSlogan.png`, and then click OK. Figure 3-5 shows the site's home page displaying the new site graphic.

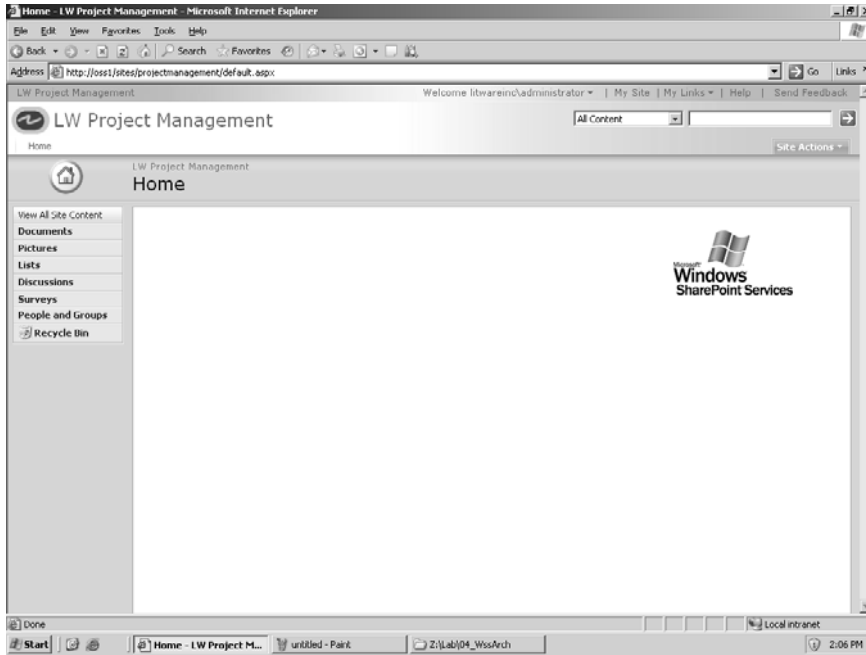


Figure 3-4 Simple customizations to the titles and graphics of the top-level site



Figure 3-5 A custom graphic can be used in the Site Image Web Part.

Creating a List for Tracking Project Profiles

In this example, we'll create and modify a custom SharePoint list that will be used to manage information about consulting projects. Start by clicking Create on the Site Actions menu to open the Create page. In the Custom Lists section, click Custom List to open the page shown in Figure 3-6 that you use to name the list. We'll name the list Projects, and then click Create.

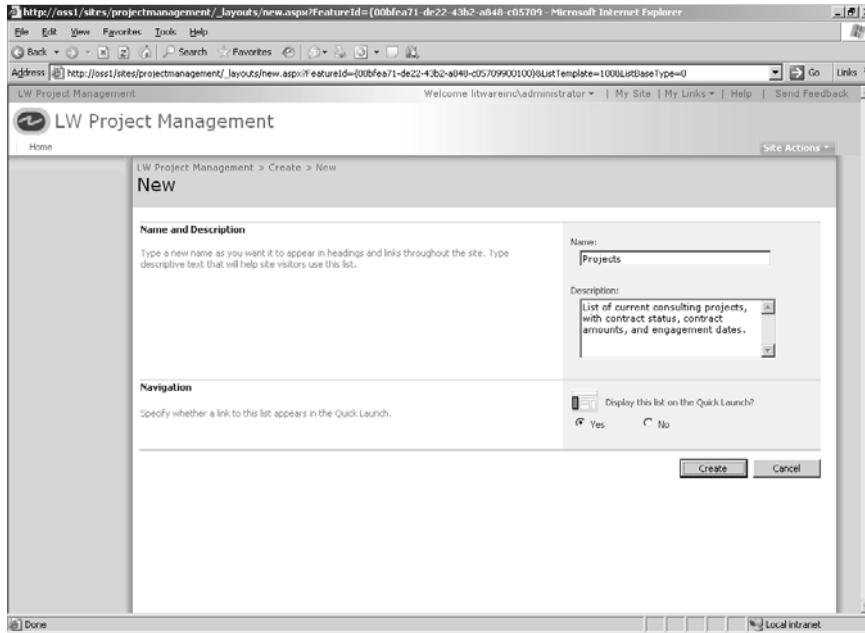


Figure 3-6 This page is used to create a custom list.

Now we want to modify some properties of the Projects list. On the list's AllItems.aspx page, click List Settings on the Settings menu to open a page with the helpful title Customize Projects. On this page, click Advanced Settings in the General Settings section. On the Advanced Settings page for the Projects list, shown in Figure 3-7, take note of the different modifications you can make to the list, such as enabling item-level permissions, allowing attachments to list items, and so on. The one change we'll make to the Projects list is to disable attachments because the structure and purpose of the list doesn't require them.

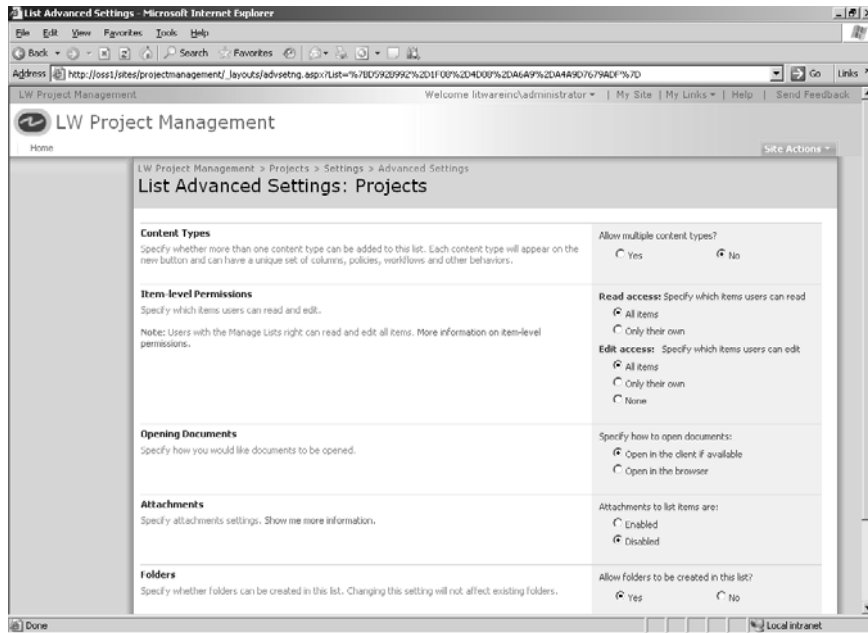


Figure 3-7 Use this page to set properties for a list.

We'll next use the List Settings page to add and modify the columns that will make up the Projects list. Using the links in the Columns section of the List Settings page, we'll structure the Projects list with the columns shown in the following table.

Column Name	Type	Notes
Project	Single Line of Text	You do not need to create this column. Instead, you can rename the Title column (one of the default columns) as Project.
Client	Single Line of Text	This is a required column.
Contract Signed	Yes/No	The default value is set to No.
Contract Amount	Currency	Currency formatting can be set to whatever seems most appropriate.
Begin Date	Date and Time	This column is formatted to show only the date.
End Date	Date and Time	This column is formatted to show only the date.
Created By	Person or Group	You do not have to create this column. It is created when you create a new list.
Modified By	Person or Group	You do not have to create this column. It is created when you create a new list.

Figure 3-8 shows the Projects list with some test data entered.

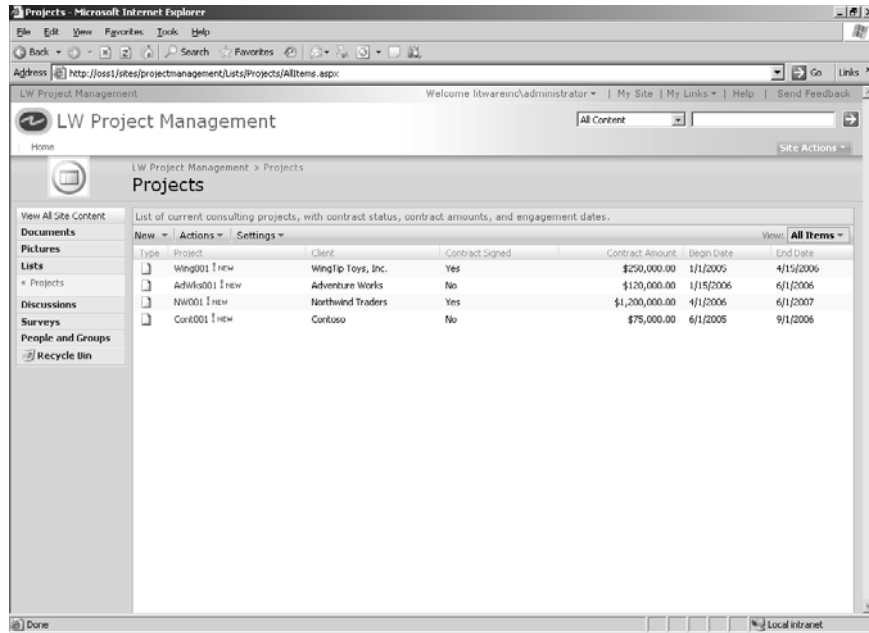


Figure 3-8 The Projects list with its column definitions and some test data

Creating a Document Library

In the following example, we'll add a document library to the Project Management site that can be used for storing and managing documents.

We start by clicking Create on the Site Actions menu to open the Create page. In the Document Libraries section, click Document Library to open the page on which you provide a name for the document library. We'll name the library Project Documents. Now that we've created the document library, we can make the use of it more particular by adding a column to associate custom metadata with each document that is added to the library.

On the document library's AllItems.aspx page, on the Settings menu, click Document Library Settings. This command takes you to a page named Customize Project Documents. We'll add a column named Project to the document library that is a lookup column. The source for this column will be the Project column from the Projects list. In creating this column, we also select the option to make the Project column required. Figure 3-9 shows the settings for the Project lookup column.

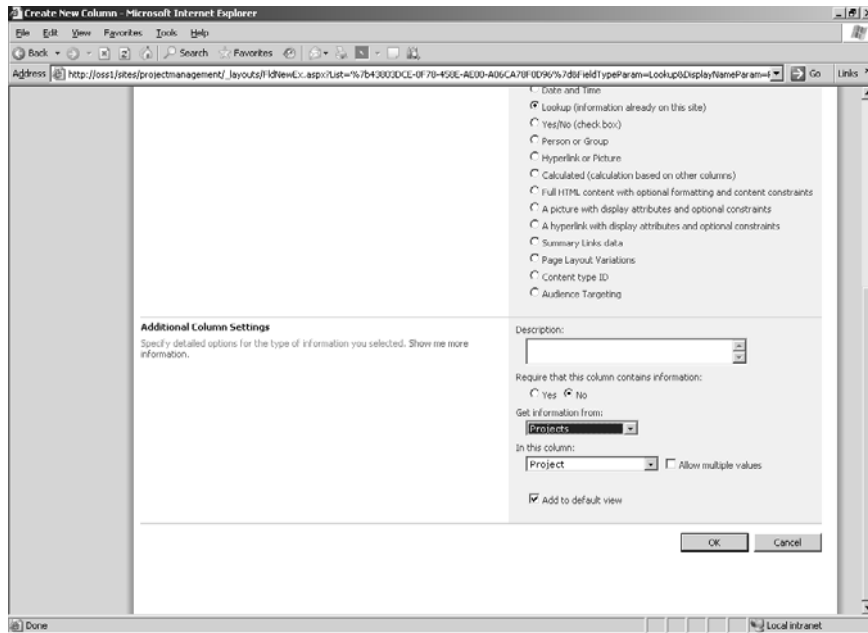


Figure 3-9 Settings for a document library column that looks up project names

We can now upload a few documents to test out the Project Documents document library. As you might expect, whenever a document is uploaded to the Project Documents document library, Windows SharePoint Services displays a page that prompts a user to assign a value to the Project column. Windows SharePoint Services forces users to associate each and every document that is stored in this document library with a specific project.

Customizing the Home Page

One of the steps you'll often want to take when designing and developing navigational aspects of a SharePoint site is to specify the items that appear on the Quick Launch toolbar on the left side of the home page.

To customize this control, start by clicking Site Settings on the Site Actions menu. On the Site Settings page, click Quick Launch in the Look And Feel section. On the Quick Launch page, you can add, edit, and delete links and headings from the Quick Launch control. In this case, we'll remove the headings for Pictures, Discussions, and Surveys. After these modifications, the Quick Launch toolbar looks like Figure 3-10 on the next page.

Next we'll add two Web Parts to the home page, one to display the Projects list, and a second to show the Project Documents document library. To add these Web Parts, we place the site's home page in edit mode by choosing Edit Page from the Site Actions menu. Next we click Add A Web Part at the top of the left Web Part Zone. In the Add Web Parts dialog box, select the items for the Projects list and the Project Documents document library, and then click Add.

After you have added Web Parts to a page, as shown in Figure 3-11, you can experiment by moving them between zones and changing some of their Web Part properties.

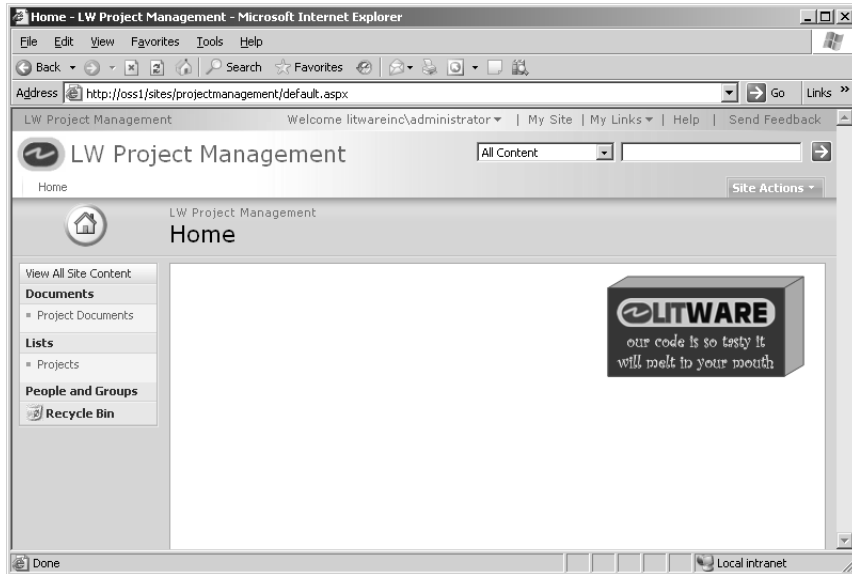


Figure 3-10 The customized Quick Launch toolbar

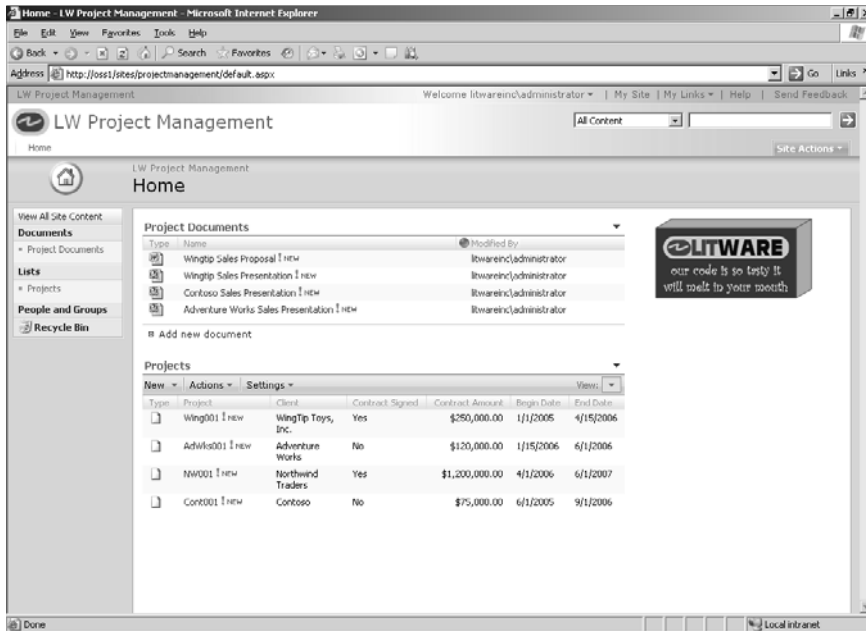


Figure 3-11 The home page with the document library and list Web Parts

Creating Child Sites

We'll continue building the Project Management team site by adding two child sites that are used to track departmental or divisional information within the organization. Keep in mind that when you create child sites, you want to make it easy for users to navigate between the top-level site and a child site, between child sites, and back again. To handle this need, you can create a top link bar that appears on the parent site and on child sites, similar to the one shown in Figure 3-12.

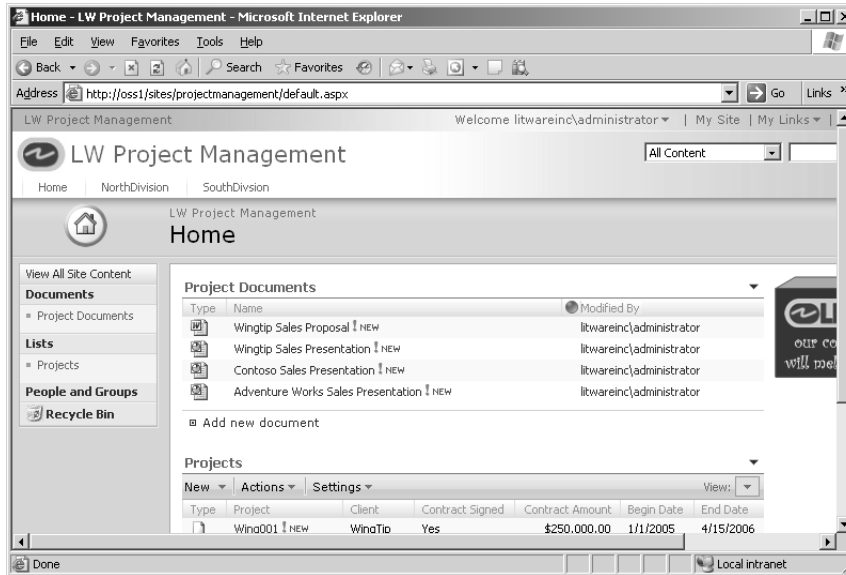


Figure 3-12 The top link bar is used to navigate between sites in a collection

Start by clicking Create on the Site Actions menu. On the Create page, under Web Pages, click Sites And Workspaces. This link takes you to a page on which you can create a child site. In our example, we've named the first child site North Division. The site's URL is <http://oss1/sites/ProjectManagement/NorthDivision>. We've used the Blank Site as the site template. In the Navigation section of the New SharePoint Site page, select Yes for the option Use The Top Link Bar From The Parent Site. Also select Add To Top Link Bar. As we did with the Project Management home page, we can modify the Site Image Web Part for the North Division site so that it displays a custom graphic instead of the default Microsoft graphic.

Setting up the child site for the South Division follows the same steps. At this point, you should be able to navigate between the top-level site and either child site with a single click on the top link bar.

Creating a “Hello World” Web Part

In this section, you’ll see how to develop a simple custom Web Part in Microsoft Visual Studio 2005. Figure 3-13 shows a view of the class library DLL project that we’ll work with, which is included in a solution named LitwareWebPartsLab.

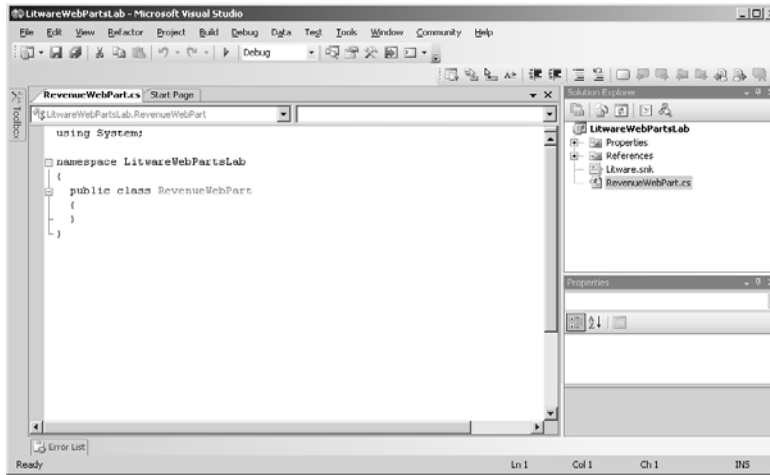


Figure 3-13 The LitwareWebPartsLab solution

The LitwareWebPartsLab project is a class library project that is configured to build an assembly DLL with a strong name. Here is the code from the AssemblyInfo.cs file that describes the format for the name of the output assembly DLL:

```
using System.Reflection;

// If needed, copy Assembly format name from below to paste elsewhere
// LitwareWebPartsLab, Version=1.0.0.0, Culture=neutral,
// PublicKeyToken=d4e5777b16a5749f
[assembly: AssemblyVersion("1.0.0.0")]

// this was used to get around a beta 1 bug for importing Web Parts
[assembly: System.Security.AllowPartiallyTrustedCallers()]
```

The file RevenueWebPart.cs contains a class named *RevenueWebPart*. We’ll transform this standard class into a Web Part for use with the Project Management site we’ve been working on.

The first step is to add a reference to *System.Web.DLL*. This system assembly has types that are required to write ASP.NET-style Web Parts. Next we modify the *RevenueWebPart* class so that it inherits from the *WebPart* class defined within the *System.Web.UI.WebControls.WebParts* namespace. Then, inside the *RevenueWebPart* class, we override the *RenderContents* method with a minimal “Hello, world” implementation. Here’s the code:

```
using System;
using System.Web.UI;
```

```

using System.Web.UI.WebControls.WebParts;

namespace LitwareWebPartsLab {
    public class RevenueWebPart : WebPart {
        protected override void RenderContents(HtmlTextWriter writer) {
            writer.Write("Hello, world");
        }
    }
}

```

A Web Part DLL must be placed in a location where the Windows SharePoint Services run time can find it. When you deploy a Web Part DLL, you have the option of putting it in the local \bin directory or the Global Assembly Cache (GAC). In this example, we're going to compile the assembly DLL output for the LitwareWebPartsLab project directly to the \bin directory of the default Web site. To do this, we can modify the Output path for the LitwareWebPartsLab project as shown in Figure 3-14. Now we can build the LitwareWebPartsLab project, and after compiling the project, we can verify that the assembly DLL has been created in the C:\InetPub\wwwroot\bin directory.

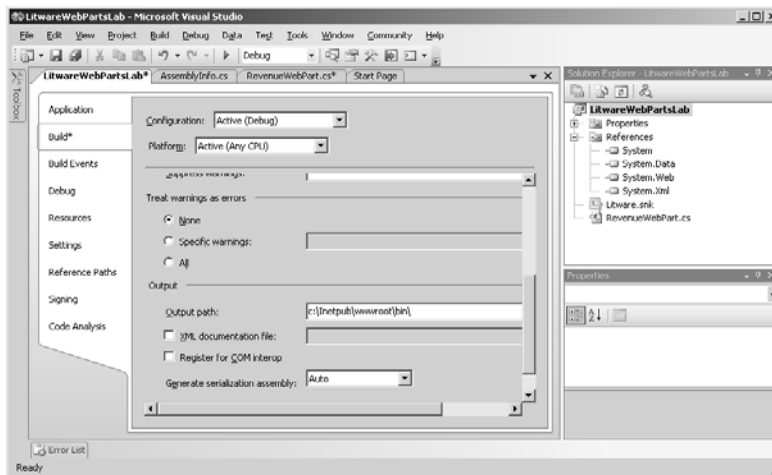


Figure 3-14 Setting the Output path in the project's properties window

We also need to modify the Web.config file in the C:\InetPub\wwwroot directory by adding a *SafeControl* setting for the new Web Part. The setting uses the following format:

```
<SafeControl Assembly="[assembly name]" Namespace="[namespace]" TypeName="*" />
```

In this example, the namespace is *LitwareWebPartsLab* and the four-part assembly format string looks like this:

```
LitwareWebPartsLab, Version=1.0.0.0, Culture=neutral, PublicKeyToken=d4e5777b16a5749f
```

After compiling the Web Part DLL to the \bin directory and configuring it in the *SafeControls* list of the default Web site, we can add the Web Part to the Web Parts gallery of the top-level site.

To do this, go to the Project Management site home page and click Site Settings on the Site Actions menu. Click the Web Parts link in the Galleries section to navigate to the Web Part Gallery page. Click the New button on the toolbar to open the page that allows you to add Web Parts to the gallery. Click the check box to the left of LitwareWebParts.LitwareRevenueWebPart to select it, and then click the Populate Gallery button at the top of the page.

You can now add the Web Part to any page in the current site collection. Navigate to the home page of the Project Management site, and then add the Web Part to a Web Part zone. Figure 3-15 shows the home page with the Web Part added above the Litware slogan graphic.

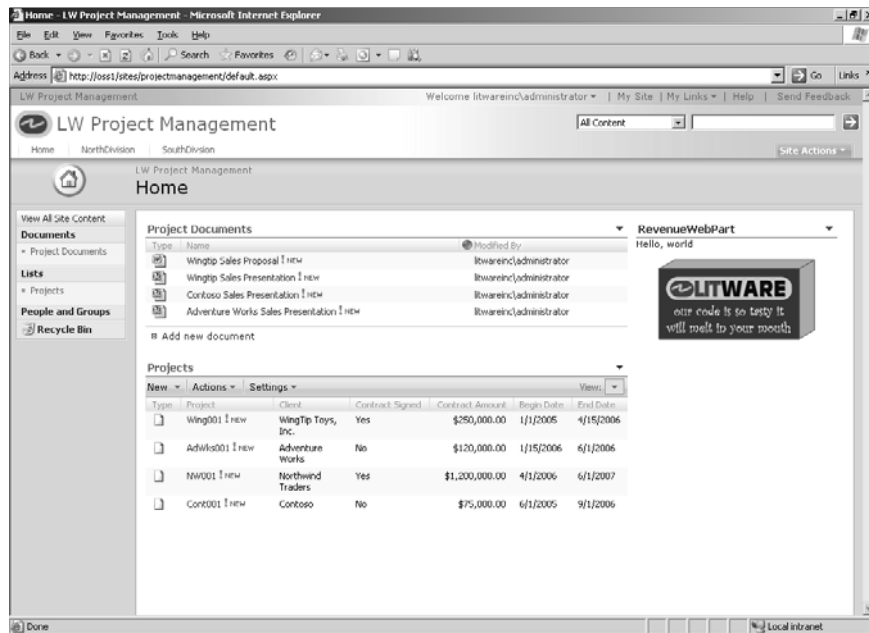


Figure 3-15 The top-level site with the custom Web Part

The Revenue Web Part could be modified in a number of ways. For example, you could add code that provides for a persistent Boolean personalizable property named *ShowTextAsBold*. The property would give a user the ability to toggle bold text on or off. When the user has the property disabled, the Revenue Web Part renders its output using a standard font. When the property is enabled, the Revenue Web Part renders its output using a bold font with a larger size. If you're in the mood, you could also change the color of the font.

The Revenue Web Part could also be modified to generate some helpful business information from a SharePoint list. In the code you've seen to this point, the Web Part has been written as a standard ASP.NET Web Part that can run on any ASP.NET site. To enable the Web Part to

access data within a SharePoint list, you need to add Windows SharePoint Services–specific code to the DLL.

First you need to add a reference to *Microsoft.SharePoint.dll*. Next, in the *RenderContents* method, you would write code to acquire a reference to the current site (an *SPWeb* object) and then obtain a reference to the Projects list (an *SPList* object). Next the code would enumerate through the list items one by one and add together the contract amounts to calculate a total. The sum of the contract amounts will be displayed as the total revenue.

Code you could use to make modifications such as these is as follows:

```
npublic class RevenueWebPart : WebPart {

    protected bool _ShowTextAsBold = true;
    [
        Personalizable(),
        webBrowsable(true),
        webDisplayName("Show Text As Bold"),
        webDescription("Enable to turn on bold font output")
    ]
    public bool ShowTextAsBold {
        get{ return _ShowTextAsBold; }
        set{ value = _ShowTextAsBold; }
    }

    protected override void OnPreRender(EventArgs e)
    {
        this.Title = "Litware Project Revenue";
    }

    protected override void RenderContents(HtmlTextWriter writer)
    {
        SPWeb ProjectManagementSite = SPContext.Current.Web;
        SPList Projects = ProjectManagementSite.Lists["Projects"];
        decimal TotalRevenue = 0;
        foreach (SPListItem Project in Projects.Items) {
            TotalRevenue += Convert.ToDecimal(Project["Contract Amount"]);
        }

        if (_ShowTextAsBold) {
            writer.AddStyleAttribute(HtmlTextWriterStyle.FontWeight, "Bold");
            writer.AddStyleAttribute(HtmlTextWriterStyle.FontSize, "12pt");
            writer.AddStyleAttribute(HtmlTextWriterStyle.Color, "Red");
            writer.RenderBeginTag(HtmlTextWriterTag.Span);
            writer.Write("Total Revenue: " + TotalRevenue.ToString("$#,###"));
            writer.RenderEndTag(); // </Div>
        }
        else {
            writer.Write("Total Revenue: " + TotalRevenue.ToString("$#,###"));
        }
    }
}
```


Chapter 4

Organizing Lists and Documents with Site Columns and Content Types

In this chapter:

Content Types	57
Site Columns	66
Working with Site Columns and Content Types	67

Large enterprises often encounter problems managing the volume of content they produce and store. Information workers create and rely on many different types of documents, but their organizations have no clear way to enforce standards with respect to the templates on which users base a document or the properties that are used to identify a document or classify the content it contains. In addition, an organization's content management applications should be able to tie the list of actions that are available to users to the type of content or document. Finally, organizations need to store different types of documents in a central location rather than parcel out documents according to type or file format.

This chapter provides an overview of two features in Microsoft Windows SharePoint Services that organizations can use to store and manage documents and other content: site columns and content types. The first sections of the chapter provide background about these features, and in the last part of the chapter we'll demonstrate some examples using a sample SharePoint site. We'll look at content types first and then review site columns, which have much in common with content types.

Content Types

In Windows SharePoint Services 2.0, a list was defined by a single set of data requirements (or schema) that applied to each item in the list. Having a single set of requirements meant that a list item was tied to its location. All the items in that location had to adhere to the columns defined for the list or document library. In Windows SharePoint Services 3.0, lists can include multiple schemas, in the form of content types. *Content types* provide the means to encapsulate a data schema and make it independent of a location on a SharePoint site.

Content types help users organize the documents and other material stored on a SharePoint site. For developers, a content type provides a way to define and use metadata to distinguish one type of list item from another. A content type is a collection of settings and metadata that is applied to a certain category of content. For example, a content type named Specifications might include columns that record metadata such as Project, Priority, Developer, and Test Lead. A content type named Contract might use columns such as Approver, Signed (Yes/No), and Amount. Even with these differences, specifications and contracts can be items in the same list or document library within a SharePoint site.

Because content types are independent of a specific list or document library, a content type can be used in lists on multiple SharePoint sites, which enables the types of content stored in a site collection to be defined and managed more centrally. For example, the Specification content type could be applied to a number of relevant lists, even if specification documents are stored on multiple SharePoint sites, to ensure that an organization keeps track of the same metadata for all software and other project specifications. A content type can be extended by including in its definition settings such as a workflow or custom attributes that are applied to the items on a site.

More Info For more information about using workflows on a SharePoint site, see Chapter 7, “Creating Workflows: The Missing Piece of Office Productivity.”

Content types can be defined through the Windows SharePoint Services user interface, by using the Windows SharePoint Services object model, or by deploying a feature that installs the content type based on an XML definition file.

More Info For more information about features, see Chapter 5, “Working with Features in Windows SharePoint Services.”

Content Type Settings

A content type can include the following information:

- Metadata, or properties, assigned to the type. These properties are represented by the columns added to the list or document library.
- A template on which to base documents of this type.
- Custom New, Edit, and Display forms to use with the content type.
- Event handlers.
- Workflows available for items of this content type. A workflow can be initiated automatically, based on a selected event or condition, or through a user’s action.

- Retention policies.
- Information necessary for custom solutions associated with the content type. You can store this information in the content type as one or more XML documents.

File Formats

Content types are independent of file formats. In document libraries, a document template can be specified for a content type. Windows SharePoint Services uses this template when a user creates a document of this type. Users can still upload into the document library a document based on a different template or a file of a different file type.

Suppose, for example, that you create a content type that will be applied to the documentation required for a project. The Project content type can be applied to any file format, including a Word document that contains project planning information, goals, risk assessments, and similar information; an Excel workbook that contains the project budget; a Microsoft Project document used for scheduling; and SharePoint list items that store the names and roles of team members, tasks, and the like. All these files and list items can be assigned the Project content type.



Note You can assign content types to SharePoint items that do not have a file format at all, such as list items or folders. A content type created for documents can be applied only to document libraries; likewise, a content type created for list items can be applied only to lists. A content type created for folders can be applied to either document libraries or lists.

Site and List Content Types

A content type that you create at the site level is called a *site content type*. A site content type is a template that is independent of any specific list or library. Any child site can inherit a site content type. If a site content type is created at the root site of a site collection, any site in that site collection can inherit the site content type.

Site content types are inherited by reference. When a child site inherits a site content type, Windows SharePoint Services includes a reference to the site content type locally in the child site. The site content type is then available to provision lists within the child site.

When a site content type is added to a list, Windows SharePoint Services stores a local copy of the site content type in the list itself. This local instance, called a *list content type*, is applicable only to the list it is copied to. Because Windows SharePoint Services stores a copy of the site content type as a list content type in each list the content type is added to, changes to list-specific instances can be made without affecting the site content type itself. Changes to a list content type are limited to that list and do not affect the parent site content type or any other list content types that inherit from that site content type.

Creating Content Types Based on Other Content Types

You can create content types based on other content types. Using this relationship, you can create general site content types that contain attributes for several different types of content and then derive more specific site content types from them. At the site level, you can create site content types based on other site content types. At the list level, you can create list content types based on site content types, provided that the site content type is within the scope of the site or list.

You can edit the attributes of a site content type that you derive from another site content type by adding, altering, or removing columns; specifying different New, Edit, or Display forms; or specifying a different document template.



Note The content type of an item is set explicitly by selecting the content type for that item. However, even items that haven't been assigned a specific content type must comply with the metadata requirements of the list or document library in which the items are located. The list schema functions as the default content type of the list itself.

Figure 4-1 shows an example of the scope of content types created for a SharePoint site collection. Site content types created at the root site of the collection, the Project Management site, are available for its direct child sites (North Division and South Division) and also for sites lower in the site hierarchy, such as North Region 1, North Region 2, and the Assignments list. Site content types created within the North Division site are available for the regional sites and the Assignments list, but these content types are not available in the South Division site because it is not a child of the North Division site. In addition, any content types created for the Assignments list are list content types and available only to that list.

The base content type hierarchy included in Windows SharePoint Services corresponds to the types of lists that you can create. The base content type hierarchy includes types such as Document, Event, Issue, Contact, Task, Announcement, and so on. When you create a list, Windows SharePoint Services creates a list content type based on the appropriate base site content type. For example, when you create an announcement list, Windows SharePoint Services copies the Announcement content type locally to the list.

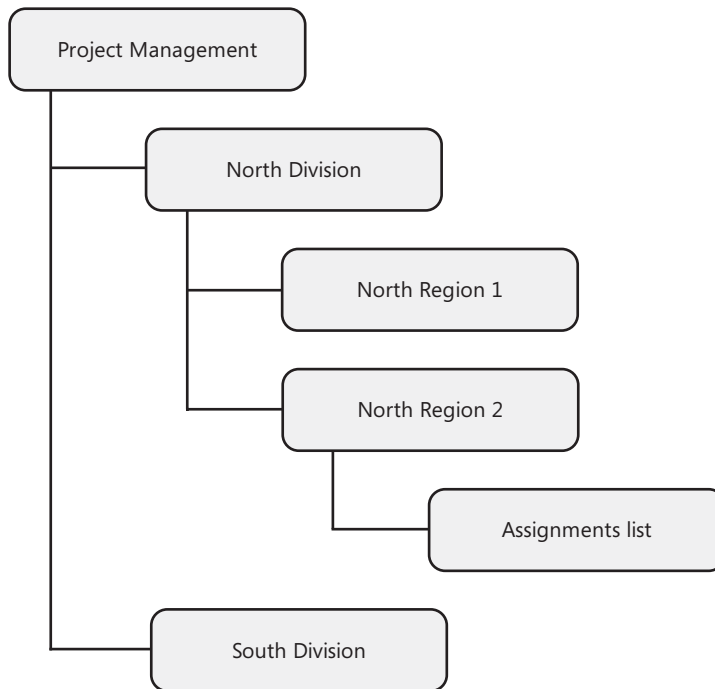


Figure 4-1 An example of content type scope

Controlling Changes to Content Types

You can prevent users from making changes to content types in two ways: specify a content type as read-only or define it as sealed. (There is no way, however, to prevent users from creating content types based on a specific site content type.)

Read-Only Content Types

A read-only content type prevents users from making changes to the content type through the user interface in Windows SharePoint Services. The content type can still be altered programmatically, using the Windows SharePoint Services object model. Specifying that a content type is read-only also affects how changes made to the parent content type can be pushed down. You'll learn more about updating content types and propagating changes in "Updating Content Types" on the next page.

Sealed Content Types

Defining a content type as sealed provides more control of the content type. Sealed content types cannot be changed through the Windows SharePoint Services user interface or the object model. To change a sealed content type, you need to change the definition of the

content type that is specified in the XML file used to provision the content type. Sealed content types are not updated through push-down operations.

Controlling Access to Content Types

One way to control access to a content type is to specify that it be hidden. Hidden content types are not displayed in the Windows SharePoint Services user interface, so users are prevented from selecting or editing the content type. This approach is useful when defining site content types that will be used as the base for other content types that users can alter. Hidden content types can be accessed through the Windows SharePoint Services object model.

Content types can also be added to a built-in content type group, named `_hidden`. This group is not displayed in the group lists from which users select content types to apply to lists or to use as the basis for other content types. Users also cannot select content types from this group to use as a parent for new content types. An administrator can choose to add a content type from this group to a list.

Updating Content Types

Each content type contains a reference to the site content type on which it is based. This reference enables Windows SharePoint Services to push down changes that are made to a parent content type to site and list content types that are derived from it. Push-down operations are optional so that users who consume a content type can make their own customizations and so that an administrator can seal a content type to prevent others from making changes.

When you choose to push down changes to a content type to all of its children, the operation does not overwrite the entire content type. The scope of what is overwritten depends on whether the changes and push-down operation are executed through the Windows SharePoint Services user interface or through the Windows SharePoint Services object model.

If a push-down operation fails on a given child content type, Windows SharePoint Services continues propagating changes to other child content types. Windows SharePoint Services also returns a list of errors it encounters during a push-down operation.



Note To create or manage a site content type on a site, you must have access rights to that site. If you do not have the appropriate access rights to a child site, push-down operations to content types contained in that child site will fail.

Updating Content Types Through the User Interface

When editing a site content type by using the Windows SharePoint Services user interface, all the settings contained on the content type settings page are overwritten during a push-down operation. The level of detail of the changes that can be pushed down is defined within the

following areas. Each time you make a change in an area, the entire area is overwritten when changes are propagated:

- **Document template settings** This area contains the following settings:
 - Document template URL
 - The actual document template file, if you selected an existing template on the Web site or uploaded a template file
 - Read-only attribute
- **New column settings** This area involves adding a new column to the site content type.
- **Column settings** This area contains the following settings:
 - Status
 - Order
 - Removing a column from the site content type

Updating Content Types Through the Object Model

The Windows SharePoint Services object model provides more control for propagating changes made to a content type to the children of that type. Changes to a site content type made through the object model affect the in-memory representation of the site content type, but Windows SharePoint Services doesn't commit those changes to the site database until a program using the object model calls the *Update* method.

The following code example adds two columns to a site content type named SalesPresentation and then pushes down those changes to the content types based on SalesPresentation.

```
SPWeb web = GetSPWeb();
SPContentType myCT = web.ContentTypes["SalesPresentation"];
myCT.Fields.Add(Field1);
myCT.Fields.Add(Field2);
myCT.Update(SPUpdateType.ListsAndTemplates);
The following example changes the document template associated with the site content type.
SPWeb web = GetSPWeb();
SPContentType myCT = web.ContentTypes["SalesPresentation"];
myCT.DocumentTemplate = "SpecTemplate.doc";
myCT.Update(SPUpdateType.ListsAndTemplates);
```

Updating Custom Information in Content Types

You can push down custom settings defined for a content type in an XML document by using the Windows SharePoint Services object model. Each content type has an XML document collection that a third-party solution can use to store custom settings information. Specific XML documents can be overwritten when changes to a content type are propagated. Windows SharePoint Services does not attempt to determine whether an XML document is currently

being used or is needed for a process before overwriting it. XML documents can also be deleted entirely as part of a push-down operation.

Considerations for Pushing Down Changes

The following list summarizes some considerations to keep in mind for push-down operations:

- Propagating changes to a site content type will overwrite modifications that have been made to one or more child content types if the changes to the child content type fall within the scope of the push-down operation. For example, suppose you made a change to a column in a child content type. If you make changes to that column in the parent content type (or delete the column altogether) and then push down the changes, Windows SharePoint Services overwrites the changes that you made to that column in the child content type.
- Each push-down operation propagates only the changes made to the parent content type at that time. If you do not push down the changes at the time you make them, you cannot easily push down those changes later. In most cases, you would need to reinstate your previous changes, make the changes again, and then push down that set of changes. For example, suppose that you deleted a column from a parent content type but did not push down this change when you deleted the column. A push-down operation that occurs later would not delete that column from child content types. To remove the column from the child content types through a push-down operation, you would need to add the column to the parent content type, delete it again, and then push down the change.
- If you push down changes that are no longer applicable to a child content type, those changes are ignored. For example, if you push down column settings changes for a column that has been deleted from a child content type, those changes are ignored. Windows SharePoint Services does not add the column to the child content type.
- If you push down changes to the column order of a content type, and the child content type has more or fewer columns than its parent, Windows SharePoint Services does the following:
 - If a column isn't included in the child content type, the column is skipped in the column order of the child content type.
 - If a child content type includes a column that isn't included in the parent, Windows SharePoint Services places that column at the end of the column order. If a child content type includes more than one column that the parent does not have, these columns are placed at the end of the column order and sorted by their order number prior to the push-down operation.

- If you attempt to perform a push-down operation on a child content type that is marked as read-only, the push-down operation fails unless you set the template to be read-write as part of the push-down operation.

- If a child content type is defined as sealed, the push-down operation fails on that content type.



Note You cannot delete a site content type if it is being used as the basis for other site or list content types. You must first remove this content type from all lists using it and delete all child site content types. Windows SharePoint Services does not consider items sent to the Recycle Bin when making this determination. If those items are restored after their content type has been deleted from the list, those items are assigned the default content type for that list.

Extending Content Types

Developers can extend the definition and functions of content types by using XML documents. These documents can conform to any given schema; they only need to be valid XML. XML documents included in a site content type are also copied to any child types. If you make a change to an XML document and then perform a push-down operation, the entire XML document is overwritten in any child content types.

The XML file that defines a content type must adhere to the content type definition schema. The following XML shows the structure of the content type schema:

```
<Elements>
  xmlns:"http://schemas.microsoft.com/sharepoint/"
<ContentTypes>
  <ContentType>
    ID as CTID, required
    Name as string
    Group as string
    Sealed as boolean
    Version as integer
    ReadOnly as boolean
    Hidden as boolean
    Description as string
<FieldRefs>
  <FieldRef>
    ID as GUID, required
    Name as string
    DisplayName as string
    Required as boolean
    Hidden as boolean
    Sealed as boolean
    ReadOnly as boolean
    ReadOnlyClient as boolean
  <RemoveFieldRef>
    ID as GUID, required
    Name as string
```

```

<DocumentTemplate>
  TargetName as path
<XMLDocuments>
  <XMLDocument>
    NamespaceURI as string, required

```

To see an application of this schema, here is the XML that defines the Document content type in Windows SharePoint Services:

```

<?xml version="1.0" encoding="utf-8" ?>
<!-- _lcid="1033" _version="12.0.3008" _dal="1" -->

<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

  <ContentType ID="0x0101" Name="$Resources:Document"
    Group="$Resources:Base_Content_Types" Sealed="TRUE" Version="0">
    <FieldRefs>
      <RemoveFieldRef ID="{67df98f4-9dec-48ff-a553-29bece9c5bf4}"
        Name="Attachments" /> <!-- Attachments -->
      <FieldRef ID="{5f47e085-2150-41dc-b661-442f3027f552}"
        Name="SelectFilename" /> <!-- SelectFilename -->
      <FieldRef ID="{8c06beca-0777-48f7-91c7-6da68bc07b69}" Name="Created"
        Hidden="TRUE" /> <!-- Created -->
    </FieldRefs>
    <XmlDocuments>
      <XmlDocument NamespaceURI=
        "http://schemas.microsoft.com/sharepoint/v3/contenttype/forms">
        <FormTemplates>
          <Display>DocumentLibraryForm</Display>
          <Edit>DocumentLibraryForm</Edit>
          <New>DocumentLibraryForm</New>
        </FormTemplates>
      </XmlDocument>
    </XmlDocuments>
  </ContentType>
</Elements>

```

Site Columns

A site column is a column definition that can be used in multiple lists on multiple SharePoint sites. Site columns have much in common with content types. For example, like a content type, a site column can help keep metadata about an item consistent across sites and lists. Also like site content types, you define a site column at the site level, independent of any actual list or content type, and child sites inherit site columns by reference. When you add a site column to a list, the site column is copied locally to the list as a list column.

Site columns are scoped in the same manner as site content types. The site column is available to the site on which it is created and on any child sites. At the site level, the site column collection contains definitions for or references to the definitions for each site column available on the site, whether or not it has been added to a content type or list.

Site Column Properties

A site column has several properties that define it, including its name and its data type. A site column is also a member of a column group, which is a user-defined group that organizes columns into categories.

A site column's name must be unique among all column groups in the scope at which you create the site column. A site column can be defined with one of the data types supported by Windows SharePoint Services, including Single Line of Text, Multiple Lines of Text, Number, Date and Time, Lookup, Currency, and others.

When you add a site column to a specific list or content type, you can set properties for the column to define its behavior within that list or content type. For example, you can specify whether the column information is required, whether the column is read-only, and whether the column is hidden.

Each element manifest that defines a site column must follow the column definition schema. The column definition schema for Windows SharePoint Services 3.0 builds on the schema used in Windows SharePoint Services 2.0, retaining all its elements. The following XML shows the schema:

```
<elements>
  xmlns="http://schemas.microsoft.com/sharepoint/"
  <fields>
    <field>
      ID as guid, required
      Group as string, required
      Name as string, required
      DisplayName as string, required
      Type as fieldtype, required
      Sealed, as boolean
      ReadyOnly, as boolean
      Hidden, as boolean
      RowOrdinal, as integer
```

Working with Site Columns and Content Types

In the following sections, you'll see examples of how content types and site columns are used to manage and store information. First, we'll create a site column definition on a top-level site (the Project Management site created in the previous chapter) that performs a lookup in the Projects list. We'll use that site column to define a column in a list on a child site within the same site collection. Next we'll create several content types. As mentioned earlier, content types provide an additional layer of organization and access to the content that is stored in a document library, and offer a mechanism by which users can store documents in a document library in a more structured way.

Creating a Site Column for Project Lookups

The site column we create in this section is used to look up items in a list. To start, on the top-level site, click Site Settings on the Site Actions menu to open the Site Settings page. Under Galleries, click Site Columns, and then click Create on the Site Column Gallery page, which lists the built-in site columns and their types. When you click Create, the New Column page, shown in Figure 4-2, opens.

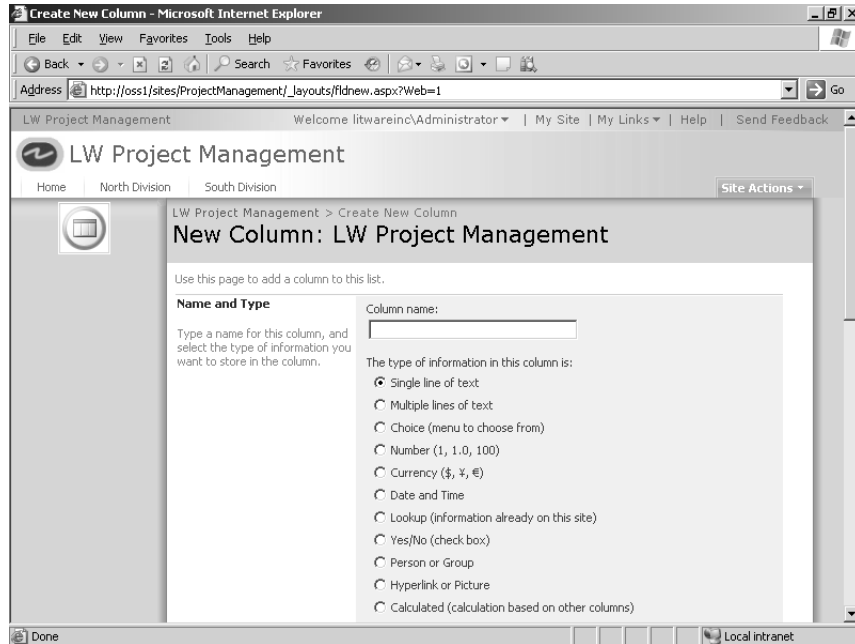


Figure 4-2 The New Column page

We'll name the site column Project and specify that its type is Lookup. We also add the column to a new group, named Litware. In the Additional Column Settings section, we can add a brief description and modify the site column so that it requires information. We assign the Projects list as the source of information for the lookup column and designate the Project column as the column to be used in the lookup. When we've finished filling out the New Column page, the Group and the Additional Column Settings sections include the information shown in Figure 4-3.

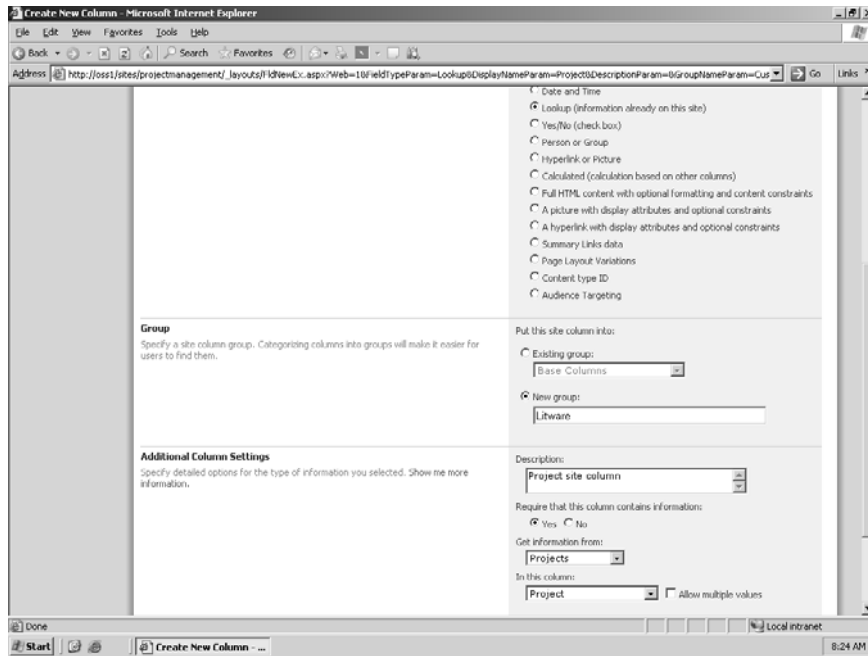


Figure 4-3 Information needed to create a lookup column

We can now use the Project site column in a list. We create a list that uses this site column on the North Division child site created in the previous chapter. The list will be used for tracking consultant time sheet information. This scenario demonstrates how a site column can easily enable lookups across sites within a site collection.

On the North Division site, click Create on the Site Actions menu to open the Create page. In the Custom List area, click Custom List, which takes you to a page that you use to name the new list. Name the list Timesheets.

When the Timesheets list is created, its AllItems.aspx page opens. On this page, click List Settings on the Settings menu to open the Customize Timesheets page. Click Versioning Settings in the General Settings section. On the Versioning Settings page, modify the list by selecting Yes for the option Create A Version Each Time You Edit An Item In This List? With this option, you can track changes made to time sheet items in the list by time and user.

The next step is to define columns for the Timesheets list, which are listed and described in the table on the next page.

Column Name	Type	Notes
TimesheetID	Single Line of Text	Renamed the Title column to TimesheetID.
Consultant	Choice	To simplify this example, this is a choice column with three choices for the consultant: Britta Simon, John Rodman, and Mike Fitzmaurice.
Project	Lookup	This column should be based on the Project site column created earlier in this example. It was created by clicking the link Add From Existing Site Columns.
Submitted On	Date and Time	This column is formatted to show the date only. It is also a required column. The default value is the current date.
Hours	Number	This is a required column. The minimum value is set to 1 hour and maximum to 24 hours. The maximum represents the fact that each time sheet item will account for one specific day.
Hourly Rate	Currency	This is a required column.
Created By	Person or Group	This column is automatically created when you create a new list.
Modified By	Person or Group	This column is automatically created when you create a new list.

Figure 4-4 shows the Timesheets list with some test data entered. You can see that the Project column has performed lookups using project names in the Projects list from the parent Project Management site.

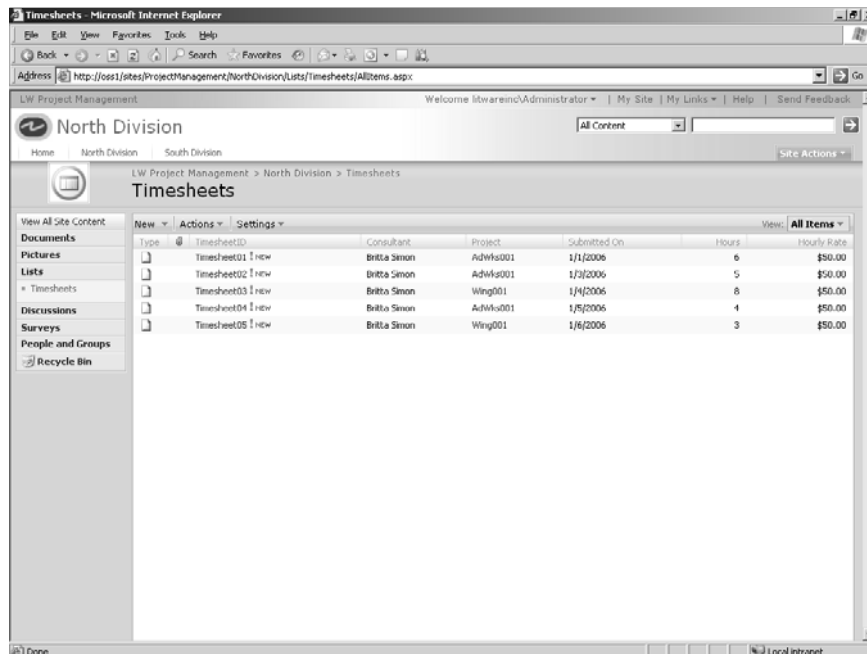


Figure 4-4 The custom Timesheets list includes the site column named Project.

Creating Custom Content Types

In this section, you'll see how to implement custom content types and use them within a document library. First, we'll create a content type named LitwareDocument. This content type serves as a base content type. We'll also create two content types, LitwareProposal and LitwarePresentation, that derive from LitwareDocument.

Start on the Project Management site's home page by clicking Site Settings on the Site Actions menu. On the Site Settings page, under Galleries, click Site Content Types to open the Site Content Type Gallery page, shown in Figure 4-5. Like the site column gallery, the site content gallery lists the standard document, folder, list, and other content types that Windows SharePoint Services provides.

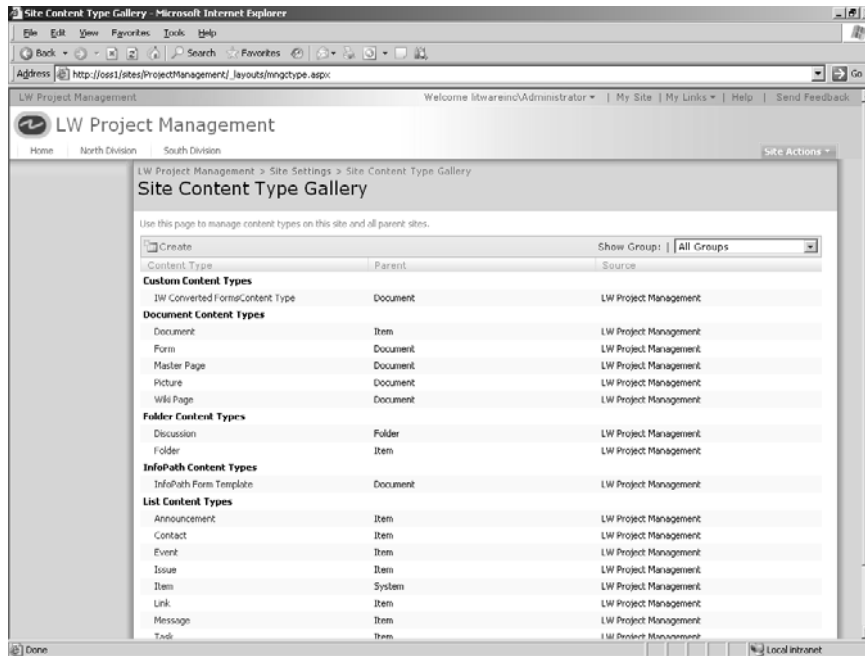


Figure 4-5 The Site Content Type Gallery page

Next, click Create to open the page used to create a content type. To create the LitwareDocument content type, fill in the page with the information shown in Figure 4-6. Notice that the content type was added to a new group, named Litware.

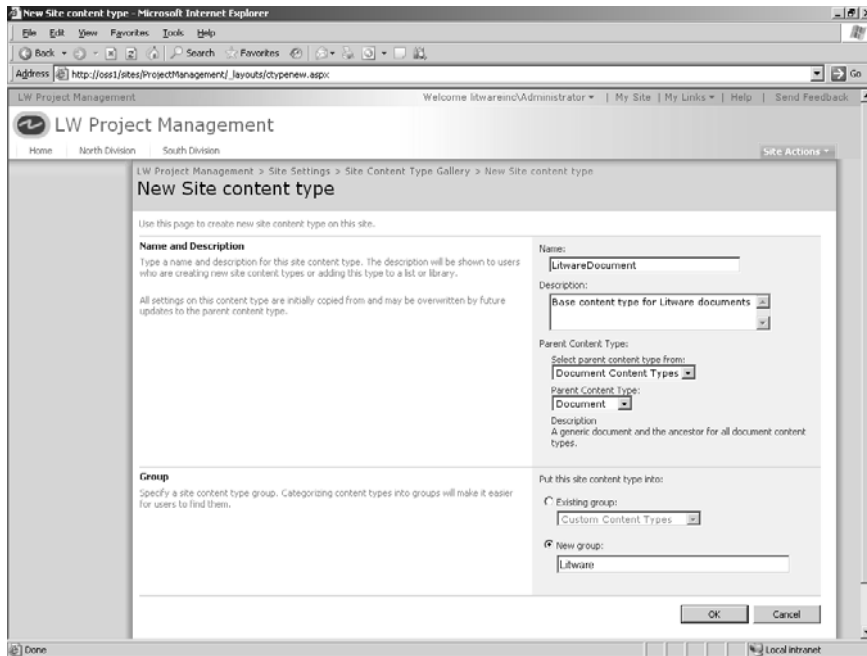


Figure 4-6 The LitwareDocument content type serves as a base content type.

To continue defining the LitwareDocument content type, we added three columns from the set of custom and built-in site columns: Author, Project, and Append-Only Comments. The Author column and the Append-Only Comments columns are provided as standard Windows SharePoint Services site columns. The Project column is the custom column created in the previous example. The Add Columns page is shown in Figure 4-7.



Note To get this example to work in beta 1, the Project column had to be configured as an optional column.

Next we created two derived content types: LitwareProposal and LitwarePresentation. These content types inherit from LitwareDocument, which means that they automatically contain the columns defined for the base content type. We can extend the derived content types by associating a document template with each of them.

Figure 4-8 shows the information used to define the content type named LitwareProposal. The settings for LitwarePresentation are essentially the same. As you can see, the derived content type contains the same columns that were defined in the base content type, LitwareDocument.

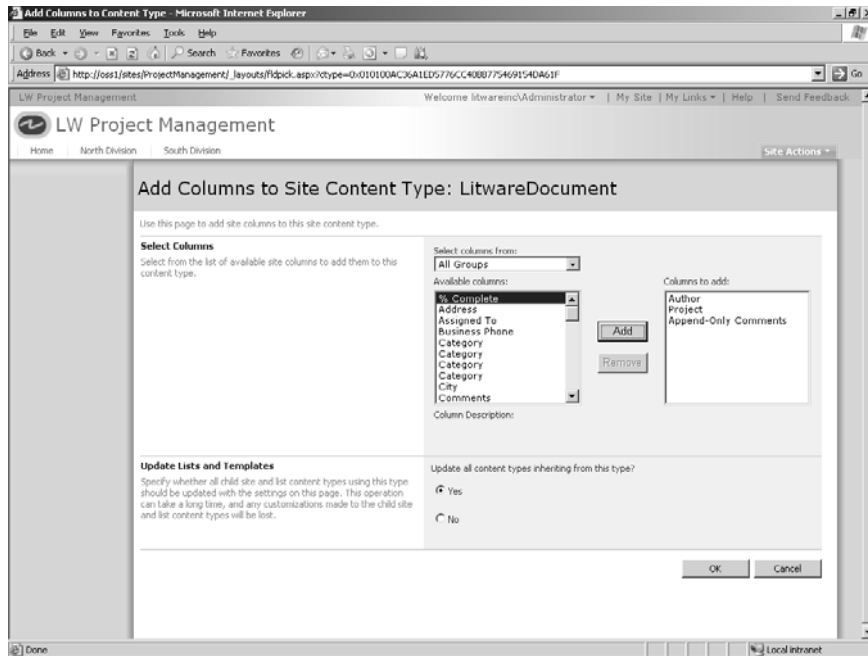


Figure 4-7 The LitwareDocument content type includes built-in and custom site columns.

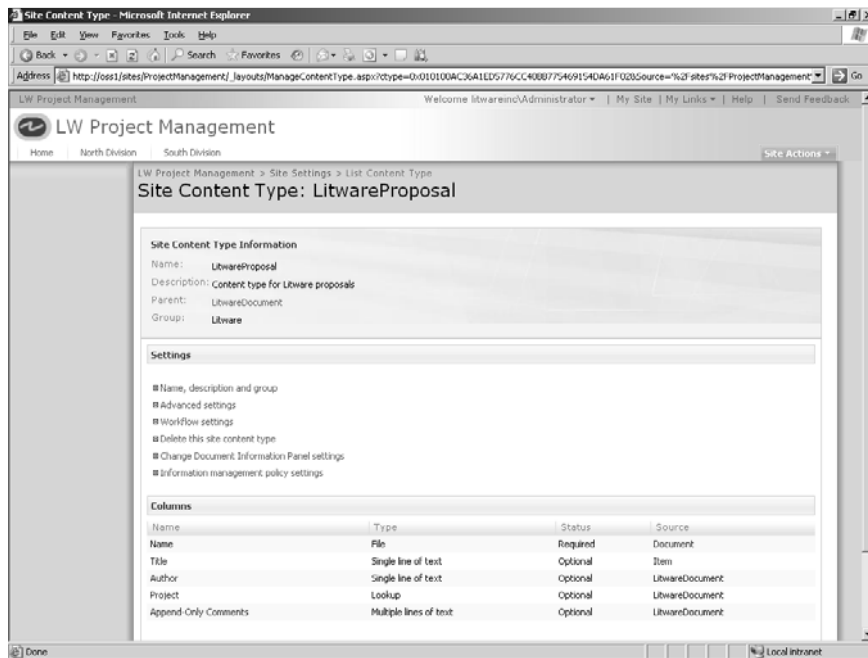


Figure 4-8 The LitwareProposal content type is derived from LitwareDocument.

By clicking the Advanced Settings link, we open a page that's used to upload a document template for this content type. In this example, the template file for proposals is named Litware-ProposalTemplate.dotx, as you can see in Figure 4-9.

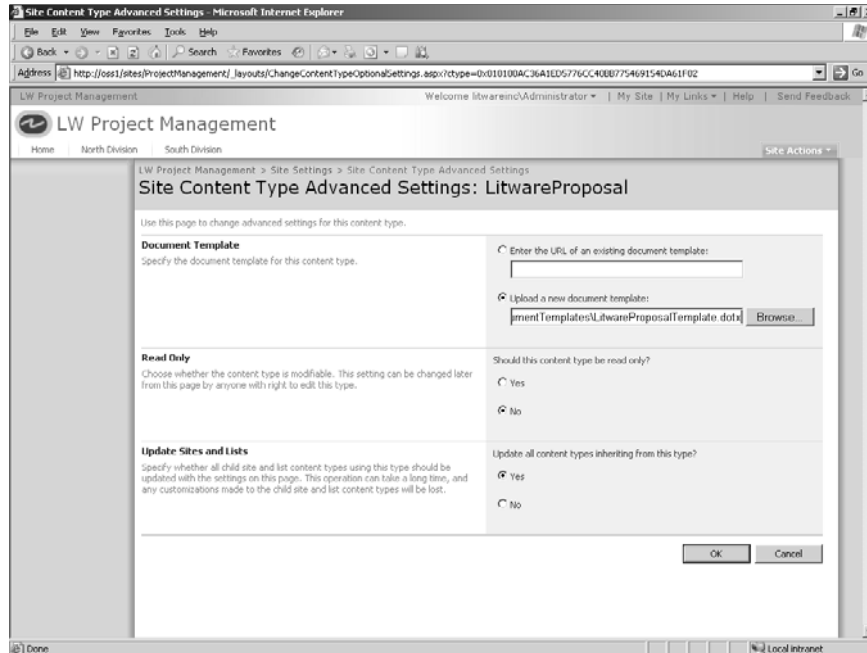


Figure 4-9 The LitwareProposal content type is associated with a particular document template.

After completing these steps, there are two content types that can be used within the Project Documents document template library.

Before this set of content types can be used, we need to configure the Project Documents document library so that it allows multiple content types. We do this by opening the Document Library Settings page and then clicking the Advanced Settings link. Figure 4-10 shows the Document Library Advanced Settings page and the option that allows for multiple content types.

We also use the Document Library Settings page to add the content types LitwareProposal and LitwarePresentation to the document library. In the Content Types section of the page, click Add From Existing Site Content Types, and then add LitwareProposal and LitwarePresentation.

Now we can go back and examine the files that are already in the Project Documents document library by choosing Edit Properties in the drop-down menu for each individual document. This command opens a page that allows you to change the content type for each document. When we've finished with the test data, there should be no documents still assigned to the standard content type named Document. Figure 4-11 shows the properties for a sales presentation with the LitwarePresentation content type specified. Notice that the content type columns—Author, Project, and Append-Only Comments—are included in the document's properties.

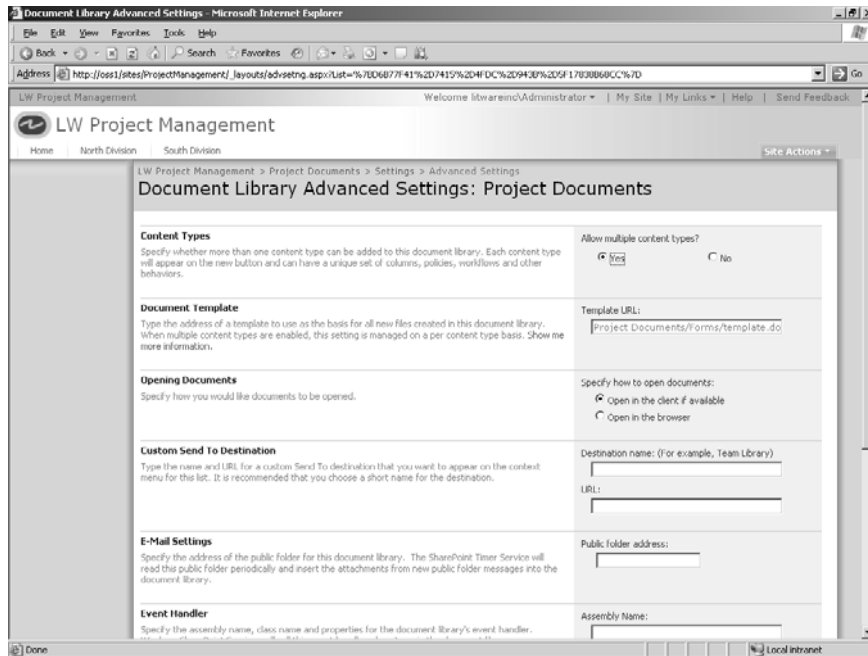


Figure 4-10 Document libraries need to be configured to allow multiple content types.

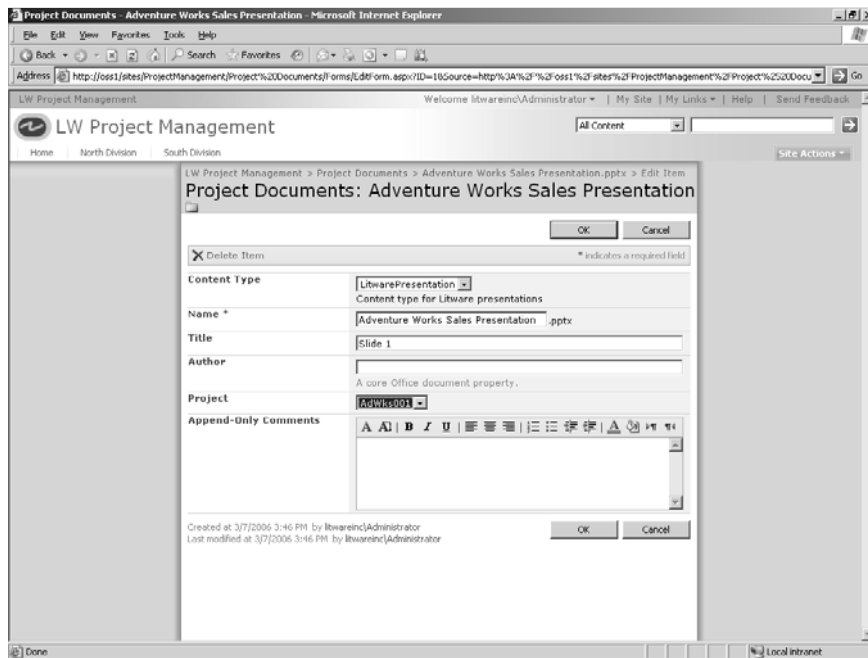


Figure 4-11 Document properties now include the columns defined for the LitwarePresentation content type.

You can test whether a particular content type has been specified for each file in a document library by opening the Document Library Settings page for the document library and

removing the original content type named Document. You will not be able to remove the Document content type if one or more documents in the document library still are assigned to this content type.

Users of the Project Management site can now use the New menu in the Project Documents document library to create a Litware proposal or a Litware presentation based on the templates assigned to these content types, as shown in Figure 4-12.

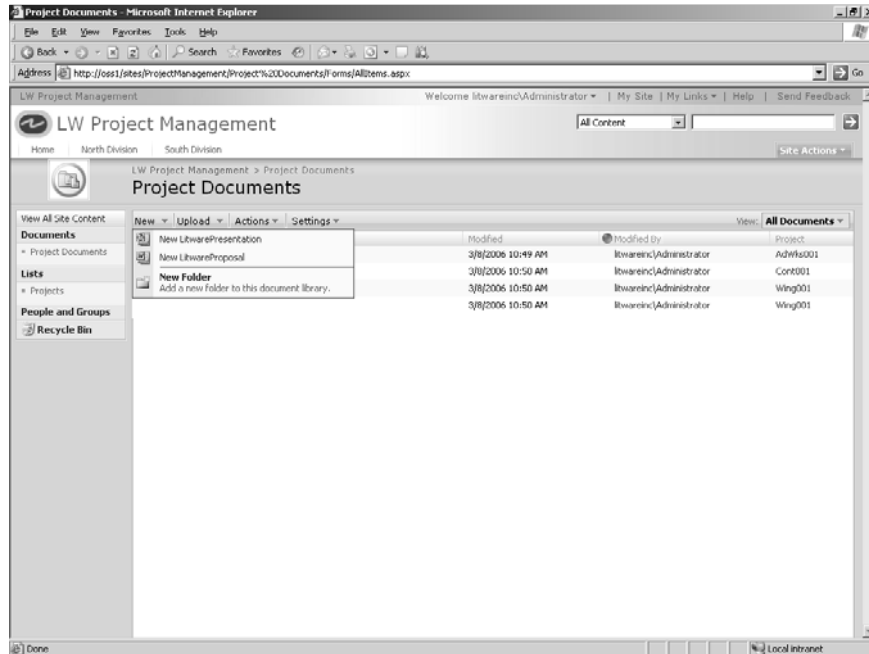


Figure 4-12 The New menu includes commands for creating documents of particular types using a specific template.

Working with Features in Windows SharePoint Services

In this chapter:

Implementing Features	78
Feature Elements	78
Element Scope	79
Activation Dependencies and Scope	79
The Structure of Feature.xml	80
Features and the Windows SharePoint Services Object Model	82
Designing Windows SharePoint Services Applications Using Features	84

Microsoft Windows SharePoint Services *features* are collections of logically related items and operations that can be added to and used repeatedly across site definitions. Features are a way to enhance the modular provisioning of SharePoint sites and help lessen the work required to make simple site customizations. For example, features eliminate the need to copy large blocks of code to change simple functionality. They reduce versioning and inconsistency issues that can arise among front-end Web servers, and they make activating or deactivating functionality in the course of a SharePoint deployment easier. An administrator, for example, can transform the template or definition of a site by toggling a particular feature on or off in the Windows SharePoint Services user interface.

Features provide the following capabilities :

- Scoping semantics for determining where custom code runs
- Pluggable behavior for installing or uninstalling features within a deployment
- Pluggable behavior for activating or deactivating features at a given scope
- A scoped property bag for storing data required by a feature within its scope
- Versioning and upgrade semantics that enhance code consistency and safeguard custom code within a deployment
- A framework for distributed deployment of SharePoint solutions

This chapter touches on the basics of working with Windows SharePoint Services features, including how to implement a feature, the structure of the XML file that defines a feature, and the Windows SharePoint Services classes that are used to work with features programmatically. The last main section of the chapter demonstrates how to create and implement a simple feature on a SharePoint site.

Implementing Features

To implement a feature, you need to add a subfolder that contains the feature's definition to the Features setup directory at the following location:

```
\Program Files\Common Files\Microsoft Shared\web server extensions\  
12\TEMPLATE\FEATURES
```

The subfolder for a feature includes a file named `Feature.xml`, which defines the base properties of the feature and lists elements that are bound to the feature, such as XML files that contain element manifests and other supporting files. The folder for a feature might contain only the `Feature.xml` file, or it might contain `Feature.xml` and any number of supporting element files, including other XML files or `.aspx`, `.htm`, `.xsn`, `.resx`, `.dll`, or other file types.

After creating the feature's folder, you can install and activate the feature through the Windows SharePoint Services user interface, command-line operations using `stsadm.exe`, or the Windows SharePoint Services object model. (You'll see examples of batch files that use `stsadm.exe` later in this chapter.) Installing a feature makes the feature definition and its elements known throughout a server farm. Activating a feature makes the feature available at a particular scope.

Feature Elements

As mentioned previously, a feature includes the `Feature.xml` file and any number of files describing elements of the feature. The *Feature* element is used in `Feature.xml` to define a feature and to specify the location of assemblies, files, dependencies, or properties that support the feature. The *Feature* element can also be used in an `Onet.xml` file to specify that a feature be included within a site definition.

A `Feature.xml` file usually points to one or more XML files whose top-level *Elements* tag contains definitions of elements that support the feature. Elements in Windows SharePoint Services 3.0 correspond to what were discrete nodes in the `Onet.xml` or `Schema.xml` file in the previous version.

A feature can include several types of elements, such as a custom menu item, an event handler, a content type, a list instance, and so on. The collective operation of these elements and

their relationship defines the functionality of a feature. For example, a feature named “My Favorite Items” might provide its functionality by including the following elements:

- A custom list that stores each user’s favorite items. The list is created as a single hidden list per workspace when the feature is enabled.
- A custom menu item named Add To Favorites that is attached to all lists.
- A Web Part that implements usage and link tracking to display the user’s top 10 favorites as the first items in the list.

Element Scope

Features can be scoped at the level of a site, site collection, Web application (virtual server), or server farm. A particular feature can be activated for a single scope. The scope of a feature is determined by the value of the *Scope* attribute of the *Feature* element.

- **Site and site collection scope** A site feature is scoped at the individual site level. A site collection feature contains items that apply to the site collection as a whole (Web Parts or item profiles that apply to the site collection, for example) in addition to items that can be activated per site. Elements with a site or site collection scope include list definitions (templates, views, and instances), modules (file sets), and item content type behaviors (for example, per-item custom menu options and per-item events).
- **Web application scope** A feature scoped for a Web application, or virtual server, can be activated or deactivated and can contain the elements for virtual server assemblies and virtual server administrative links.
- **Server farm scope** A farm feature is scoped for an entire server farm and, unlike a feature with site or site collection scope, is always activated by default in the farm. A farm feature contains a number of elements that are required for implementing applications and logic anywhere within a deployment of SharePoint. A farm feature can contain links to *_layouts* pages and files, *_admin* pages, and other elements.

Activation Dependencies and Scope

A feature can be defined so that it is dependent on another feature or features for activation. Activation dependencies can be expressed for features of the same scope, or a feature at a particular scope can be dependent on a feature at a different scope (a cross-scope activation dependency). For example, a feature at the site level can be dependent on a feature that is scoped for the site’s site collection. Cross-scope activation dependencies cannot be formed if the dependent feature has a more restrictive scope. In other words, a feature scoped for a site collection cannot have an activation dependency on a feature scoped at the site level.

A feature can have an activation dependency on either hidden or visible features. (A hidden feature is determined by the setting of the *Hidden* attribute of the *Feature* element.) Hidden features themselves, however, cannot have any activation dependencies.



Note Windows SharePoint Services 3.0 deactivates a dependent hidden feature if the last visible feature at the same scope that has an activation dependency on the hidden feature is deactivated.

Scope and Feature Globalization

Two types of resources can be implemented in features for support files:

- Local resources that are stored in a subdirectory within the Features folder in the \TEMPLATE\FEATURES directory.
- Shared application feature and site definition resources that are stored in \web server extensions\12\Resources and are designed to be used across features and site definitions.

The *LocaleId* attribute of the *Feature* element specifies the language of the feature. This attribute can be set to 0, which makes the feature available to all languages and cultures. The developer of a feature can declare a feature to be culture-neutral, even if the feature contains potentially localizable resources, which will be displayed in all languages.

The set of language support files plus the default language of the feature constitute the set of available languages for a feature. A feature scoped at the farm or Web application level does not activate if the feature does not support the language of the administrative user interface. A feature scoped at the site collection level always activates because a site collection is assumed to be inherently multilingual. A feature with a site scope is not available for a site if it does not support the language of the site.

The Structure of Feature.xml

In the Feature.xml file, the *Feature* element defines a feature and specifies the location of assemblies, files, dependencies, or properties that support the feature. A Feature.xml file can contain the following elements:

```
Feature
  ActivationDependencies
    ActivationDependency
  ElementManifests
    ElementFile
    ElementManifest
```

The *ActivationDependencies* element specifies a list of the features on which this feature depends for activation. The *ElementManifests* element specifies a list of element container files that include definitions for the operations that make up the feature.

The following table lists and describes the attributes for these elements.

Name	Data Type	Description
Feature Attributes		
<i>Id</i>	Guid	The unique identifier for the feature.
<i>Title</i>	Text	Returns the title of the feature; limited to 255 characters.
<i>DefaultResourceFile</i>	Text	Optional. By default, if a developer specifies a resource in the Feature.xml file, Windows SharePoint Services looks in FeatureName\Resources\Resources.Culture.resx. However, through <i>DefaultResourceFile</i> , you can specify an alternative file from which to grab resources.
<i>Description</i>	Text	Optional. Returns a description of the feature's function.
<i>Version</i>	Version	Specifies a <i>System.Version</i> -compliant representation of the version of a feature. This value can be up to four numbers, delimited by decimals, that represent a version.
<i>ReceiverAssembly</i>	Text	Optional. If this attribute is set along with <i>ReceiverClass</i> , it specifies the assembly from which to load a receiver to handle feature events.
<i>ReceiverClass</i>	Text	Optional. If this attribute is set along with <i>ReceiverAssembly</i> , it specifies the class that implements the feature event processor.
ActivationDependency Attributes		
<i>FeatureId</i>	Guid	The ID of the feature that a dependency refers to.
<i>Title</i>	Text	Optional. Title of the feature on which this feature is dependent. This field is usually localized.
<i>Description</i>	Text	Optional. Description of the feature on which this feature is dependent. This field is usually localized.
<i>Url</i>	Text	Optional. A URL that contains more information about the feature in question. This field is usually localized.
ElementFile Attributes		
<i>Location</i>	Text	Specifies the relative file path to the root element manifest file.
ElementManifest Attributes		
<i>Location</i>	Text	Specifies the relative file path to the root element manifest file.

The following example shows a Feature.xml file that specifies installation dependencies, special fields, and relative paths to element definition files.

```
<Feature
  Id="1111111-11111-11111-11111"
  Title="Location Services"
  Description="This feature contains lists and parts that let you link
    location data to your customer lists."
  Scope="Web">
  <InstallationDependencies>
    <InstallationDependency
      Id="FEEDBADA-11111-111111"
      Title="Portal Search"
      Description="This feature contains portal search functionality"
      Url="http://www.microsoft.com/sharepoint">
    </InstallationDependency>
  </InstallationDependencies>
  <PropertySchema>
    <Fields>
      <Field
        Type="Text"
        Name="ProjectName"
        DisplayName="Project Name">
      </Field>
    </Fields>
  </PropertySchema>
  <ElementManifests>
    <ElementManifest Location="Location\LocationPart.xml"/>
    <ElementManifest Location="CustomerLocation\CustomerLocationList.xml"/>
    <ElementFile Location="test.aspx"/>
  </ElementManifests>
</Feature>
```

Features and the Windows SharePoint Services Object Model

Windows SharePoint Services provides an object model that can be used to determine the list of installed features within a given scope and for controlling whether features are enabled at the farm and site levels. The following sections summarize the classes used in working with features.

Feature Classes

The following are the main classes you use to work with SharePoint features programmatically:

- *SPFeature* (*SPFeatureCollection*). This class returns an object that represents the state of a feature at its corresponding level. The presence of a feature in a collection at the virtual server, site collection, or site level scope means that the feature is activated. The absence of an *SPFeature* object indicates that the object is not added. The *SPFeature* class is included in the *Microsoft.SharePoint* namespace.

- *SPFeatureProperty* (*SPFeaturePropertyCollection*). Objects of this class represent a single feature property. The *SPFeatureProperty* class is included in the *Microsoft.SharePoint* namespace.
- *SPFeatureScope*. This class provides an enumeration of the possible scopes that can be specified for a feature, including *Farm*, *WssWebApplication*, *Site*, and *Web*. The *SPFeatureScope* class is included in the *Microsoft.SharePoint* namespace.
- *SPFeatureDefinition* (*SPFeatureDefinitionCollection*). This class contains the base definition of a feature, including its name, type, and the version of the feature. Also, you can store properties about the feature globally per feature. For example, a feature could be associated with a central administrative page for setting configurations for a feature. The *SPFeatureDefinition* class is included in the *Microsoft.SharePoint.Administration* namespace.
- *SPElementDefinition* (*SPElementDefinitionCollection*). This class represents an element to be provisioned when the feature is activated or used. The *SPElementDefinition* class is included in the *Microsoft.SharePoint.Administration* namespace.

Accessing Feature Collections

You can get the collection of features for a Web application, farm, site collection, or site by using one of the following properties to access the collection:

- *Microsoft.SharePoint.Administration.SPWssWebApplication.Features* returns a list of features activated at the virtual server level for the Windows SharePoint Services Web application.
- *Microsoft.SharePoint.Administration.SPWssService.Features* returns the administrative features that have been activated at the server farm scope.
- *Microsoft.SharePoint.Administration.SPFarm.FeatureDefinitions* returns the list of all installed features in the server farm.
- *Microsoft.SharePoint.SPSite.Features* returns the list of features for the site collection.
- *Microsoft.SharePoint.SPWeb.Features* returns the list of activated features for a site.

The following example demonstrates how to use several of these classes and properties to display the list of names and the GUIDs of the features that are activated on a specified site:

```
SPSite siteCollection = SPControl.GetContextSite(Context);
SPWeb site = siteCollection.AllWebs["Site"];
SPFeatureCollection siteFeatures = site.Features;
System.Globalization.CultureInfo cultureInfo = new System.Globalization.CultureInfo(1033);

foreach (SPFeature siteFeature in siteFeatures)
{
    Response.Write("Title: " + siteFeature.Definition.GetTitle(cultureInfo) +
        "<BR>ID:" + siteFeature.DefinitionId.ToString() + "<BR><BR>");
}
```

The next example uses information returned through the previous example to add a feature to a subsite:

```
SPWeb subSite = site.Webs[""];  
System.Guid guid = new System.Guid("6e005f62-f8b2-4073-a673-c035c9129946");  
subSiteFeatures.Add(guid);
```

Features and Events

Windows SharePoint Services also provides classes for responding to feature events. These classes allow you to trap and respond to an event that occurs when a feature is installed in a server farm, added to a new virtual server, or removed. Such an event is a *post event*, meaning that it occurs after the respective behavior has been committed and the feature definition has been created in the collection of feature definitions for the farm. You cannot cancel the installation or removal of a feature through a feature event.

- *SPFeatureReceiver* is the base abstract class that can be overridden to trap the activation, deactivation, installation, or removal of a feature. Each of these events is a post event. The *SPFeatureReceiver* class is included in the *Microsoft.SharePoint* namespace.
- The *SPFeatureReceiverProperties* class provides access to event properties such as the feature instance that has been created, the feature definition, or the parent object of the feature (for example, an *SPWeb* object). The *SPFeatureReceiverProperties* class is included in the *Microsoft.SharePoint* namespace.

Designing Windows SharePoint Services Applications Using Features

The following sections present examples of how to work with features on a SharePoint site. We'll look at features from the end-user perspective, to see how a feature is activated or deactivated, and then we'll look at the XML file structure for a feature and at a batch file that can be used to install and activate a custom feature using `stsadmin.exe` commands. We'll complete the set of examples by creating and testing a custom feature using XML files and Microsoft Visual Studio 2005.

Activating and Deactivating Features

Figure 5-1 shows the Create page for a new Windows SharePoint Services site at the root of the default Web site. This top-level site is based on the Blank Site template. As you can see, the Document Libraries section lists four types of libraries that can be created for the site: Wiki Page Library, Document Library, Data Connection Library, and Form Library. Slide Library is not among the items in this list. It first needs to be activated as a feature before one can be created for this site.

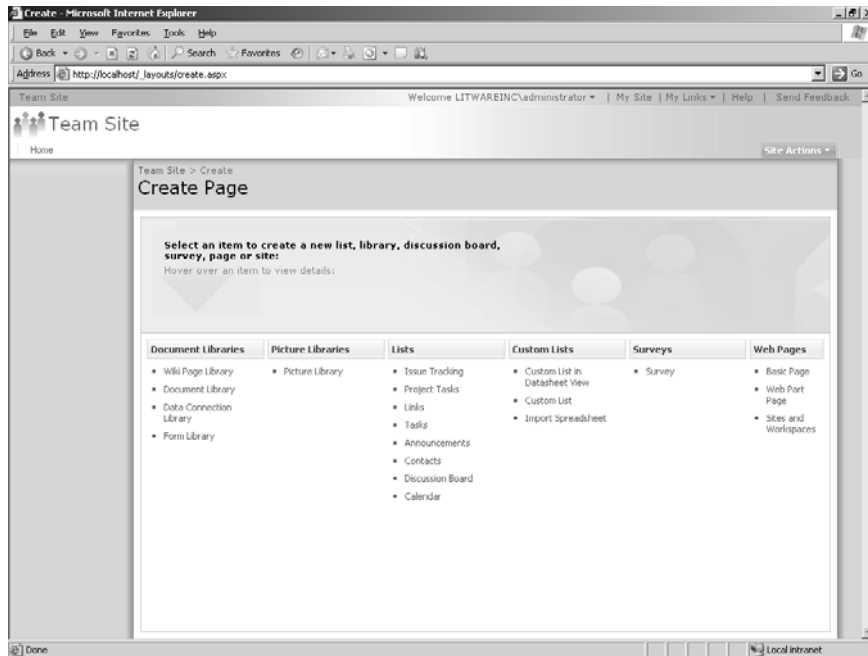


Figure 5-1 The Create page shows the types of document libraries available for this site.

Here are the steps for activating a feature for a site collection:

1. On the home page, click Site Settings on the Site Actions menu.
2. On the Site Settings page, in the Site Administration section, click Site Features to open the Site Features page for the current site. (See Figure 5-2 on the next page.) You can use this page to examine the set of installed features available at the site level, including those that are activated and deactivated for the current site. Also, you can determine which features are automatically activated for the site template you used.
3. Locate the feature you want to activate (Slide Library, for example), and then click Activate. Click Deactivate to deactivate a feature.



Note After you have activated a feature, return to the site's Create page (click Create on the Site Actions menu), and you should be able to confirm that the feature you activated now appears.

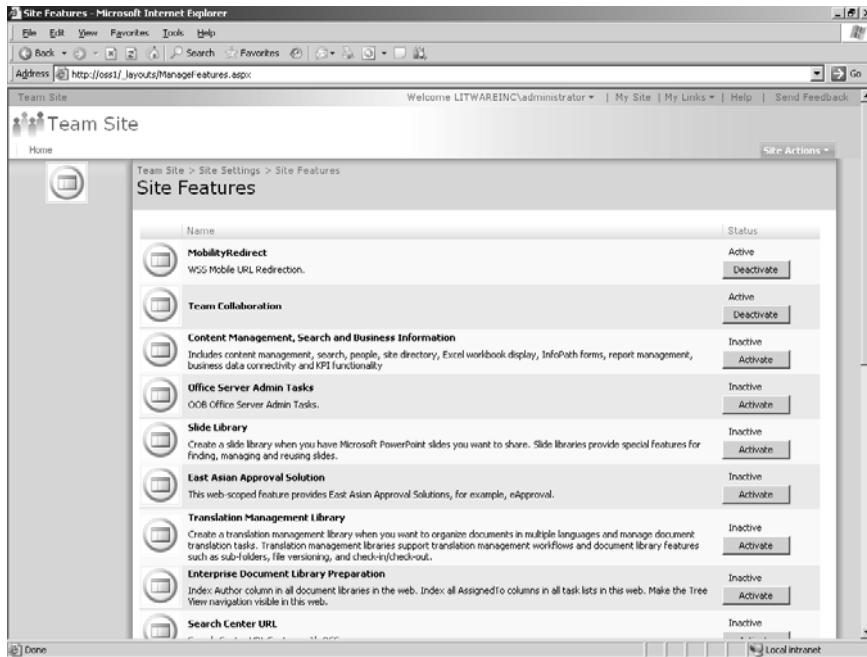


Figure 5-2 Use the Site Features page to activate or deactivate a feature through the Windows SharePoint Services user interface.

Working with a Custom Feature

Next we'll look at some of the files used to define a feature and at a batch file that can be used to install and activate a feature that you might create. The feature we'll examine is a custom list type named LitwareTimesheets. This feature has been designed to create a list instance and add some initial data items to the list.

Figure 5-3 shows the Feature.xml file, which is the feature manifest. As you can see, this file gives the feature its name and identifying GUID and defines its scope. This feature is scoped at the site level because its scope has the value "Web". You can also see that an *ElementManifest* element points to the location of a file named Timesheets.xml, which contains the data that will be included in the list once the feature is installed.

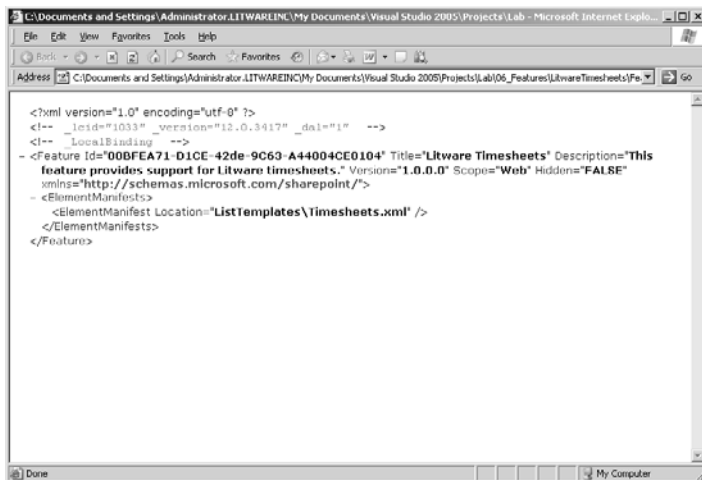


Figure 5-3 The feature manifest file, Feature.xml

The following XML, from the schema.xml file, defines the list's fields. This file also defines list views and other information.

```

<?xml version="1.0" encoding="utf-8" ?>

<List xmlns:ows="Microsoft SharePoint"
  Name="Timesheets"
  Title="Timesheet List"
  Direction="0"
  Url="Lists/Timesheets"
  BaseType="0"
  >
  <Metadata>
    <Fields>
      <Field Name="Title" FromBaseType="TRUE" Type="Text"
        DisplayName="TimesheetID" Required="FALSE"
        MaxLength="24" ID="{fa564e0f-0c70-4ab9-b863-
        0177e6ddd247}" Version="1" StaticName="Title"
        SourceID="http://schemas.microsoft.com/sharepoint/v3"
        ColName="nvarchar1" RowOrdinal="0"/>
      <Field Type="Choice" DisplayName="Consultant"
        Required="TRUE" MaxLength="50" ID="{96fe5f63-bd85-4f87-
        b423-5437e803b242}" SourceID="{e9b34e79-429b-42a4-bc6a-
        8b766aa59987}" StaticName="Consultant"
        Name="Consultant" ColName="nvarchar3" RowOrdinal="0">
        <Default>Britta Simon</Default>
        <CHOICES>
          <CHOICE>Britta Simon</CHOICE>
          <CHOICE>John Rodman</CHOICE>
          <CHOICE>Mike Fitzmaurice</CHOICE>
        </CHOICES>
      </Field>

```

```

<Field Type="Choice" DisplayName="Project" Required="TRUE"
  MaxLength="50" ID="{e18a2fdc-c40a-4e33-9e84-
  10a5df587da4}" SourceID="{e9b34e79-429b-42a4-bc6a-
  8b766aa59987}" StaticName="Project" Name="Project"
  ColName="nvarchar4" RowOrdinal="0">
  <Default>FI-23-123</Default>
  <CHOICES>
    <CHOICE>FI-23-123</CHOICE>
    <CHOICE>MR-23-123</CHOICE>
    <CHOICE>PR-10-232</CHOICE>
  </CHOICES>
</Field>
<Field Type="DateTime" DisplayName="Submitted On"
  Required="TRUE" Format="DateOnly" ID="{4676b294-0c00-
  476a-8693-6012028ececfc}" SourceID="{e9b34e79-429b-42a4-
  bc6a-8b766aa59987}" StaticName="Submitted_x0020_On"
  Name="Submitted_x0020_On" ColName="datetime1"
  RowOrdinal="0">
  <Default>[today]</Default>
  <DefaultFormulaValue>2005-12-
  30T00:00:00Z</DefaultFormulaValue>
</Field>
<Field Type="Number" DisplayName="Hours" Required="TRUE"
  Min="1" Max="24" Decimals="0" ID="{98fbe207-eedb-4903-
  bbb9-d84016dfa0d0}" SourceID="{e9b34e79-429b-42a4-bc6a-
  8b766aa59987}" StaticName="Hours" Name="Hours"
  ColName="float1" RowOrdinal="0"/>
<Field Type="Currency" DisplayName="Hourly Rate"
  Required="TRUE" Min="40" Max="225" LCID="1033"
  ID="{d77c5ea1-6646-4b18-90e4-16a23fb8dcd7}"
  SourceID="{e9b34e79-429b-42a4-bc6a-8b766aa59987}"
  StaticName="Hourly_x0020_Rate" Name="Hourly_x0020_Rate"
  ColName="float2" RowOrdinal="0"/>
</Fields>

```

And here is the XML that shows list properties and some of the data used to populate the list, from the Timesheets.xml file:

```

<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <ListTemplate
    Name="Timesheet"
    Type="10201"
    BaseType="0"
    OnQuickLaunch="TRUE"
    SecurityBits="11"
    DisplayName="Timesheets"
    Description="Litware Time Sheet List Type"
    Image="/_layouts/images/CHNGCOL.GIF"/>
  <ListInstance
    Id="Timesheets"
    Description="Litware Time Sheet List Instance"
    TemplateType="10201"
    Title="Timesheets"
    Url="Timesheets"
  >

```

```

<Data>
  <Rows>
    <Row>
      <Field Name="TimesheetID">Britta Simon FI-23-123
        1/2/2006</Field>
      <Field Name="Consultant">Britta Simon</Field>
      <Field Name="Project">FI-23-123</Field>
      <Field Name="Submitted On">1/2/2006</Field>
      <Field Name="Hours">6</Field>
      <Field Name="Hourly Rate">50</Field>
    </Row>
    <Row>
      <Field Name="TimesheetID">Britta Simon FI-23-123
        1/3/2006</Field>
      <Field Name="Consultant">Britta Simon</Field>
      <Field Name="Project">FI-23-123</Field>
      <Field Name="Submitted On">1/3/2006</Field>
      <Field Name="Hours">5</Field>
      <Field Name="Hourly Rate">50</Field>
    </Row>
  </Rows>
</Data>
</ListInstance>
</Elements>

```

Finally, here is the code from a batch file named Install.cmd. This batch file copies files into the Windows SharePoint Services Feature directory and runs stsadm.exe commands to install the feature and activate it on the site at the URL <http://localhost>.

```

@SET SPDIR="c:\program files\common files\microsoft shared\web server extensions\12"
Echo Deactivating feature
%SPDIR%\bin\stsadm -o deactivatefeature -filename
  LitwareTimesheets/feature.xml -url http://localhost
Echo Uninstalling feature
%SPDIR%\bin\stsadm -o uninstallfeature -filename LitwareTimesheets/feature.xml
IISRESET
Echo Copying files
rd /s /q %SPDIR%\Template\Features\LitwareTimesheets
xcopy /e /y Features\LitwareTimesheets\* %SPDIR%\Template\Features\LitwareTimesheets\
Echo Installing feature
%SPDIR%\bin\stsadm -o installfeature -filename LitwareTimesheets/feature.xml -force
IISRESET
Echo Activating feature
%SPDIR%\bin\stsadm -o activatefeature -filename LitwareTimesheets/feature.xml
  -url http://localhost -force

```

After this batch file is run, users can navigate to the site at the root of the default Web site and see that a list has been created named Timesheets. (The Timesheets list will not show up on the Quick Launch toolbar, so users would need to click View All Site Content on the Quick Launch toolbar to see the list, which is shown in Figure 5-4.)

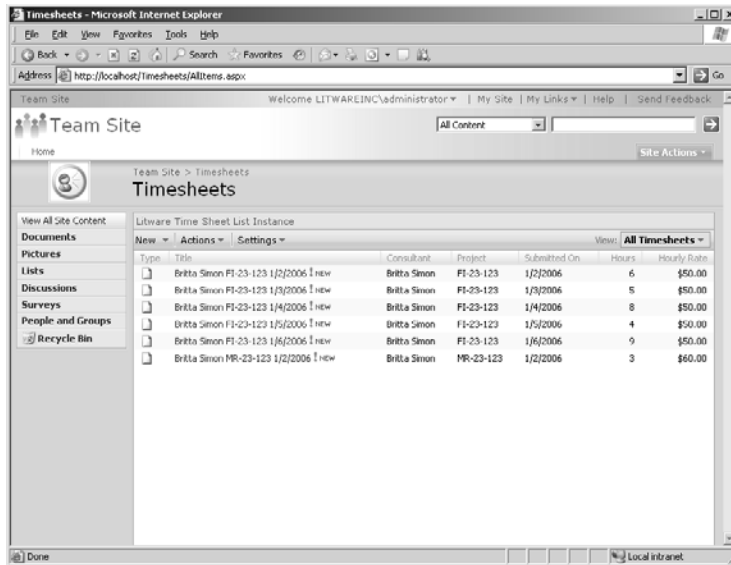


Figure 5-4 The Timesheets list and its data were created and installed through files that define a feature.

Creating a Custom Feature

Now we'll walk through the creation of a custom feature. We'll work with a Visual Studio project named `LitwareFeatureLab.sln`, which contains starter XML files for a feature named `LitwareFeatureLab` and an assembly DLL project that we'll use to create receiver classes for the feature.

Figure 5-5 shows the feature manifest file, `Feature.xml`, open in Visual Studio 2005. As you can see, this manifest file contains the essential feature attributes, such as the `Id`, `Title`, and `Scope`, but nothing else at this point.

First we'll add an element manifest to `Feature.xml` to reference a file named `Lightup.xml`. `Lightup.xml` contains the feature's functionality. Here's the markup included in `Lightup.xml` so you can examine the feature elements defined within it. We'll simply reference `Lightup.xml` in `Feature.xml`, which will add the functionality defined in `Lightup.xml` to the feature.

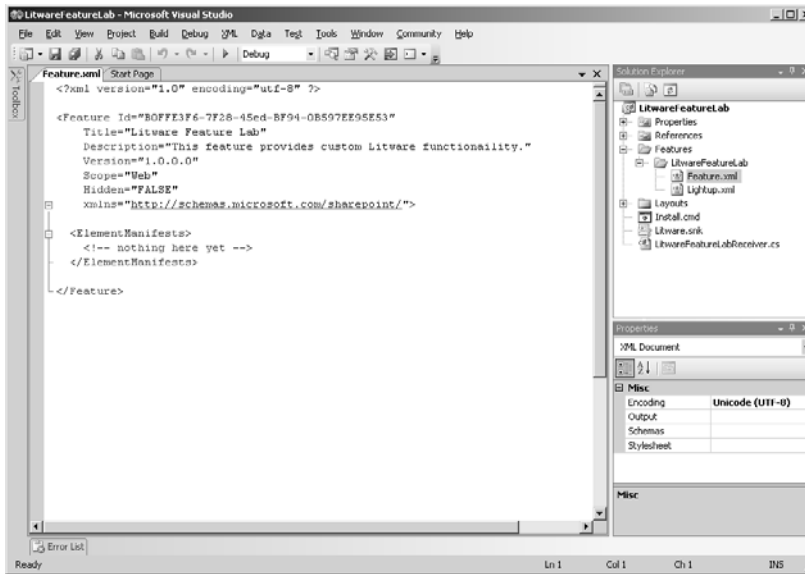


Figure 5-5 A bare-bones Feature.xml file

```
<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <!-- Create Command Link Site Settings Page -->
  <CustomAction
    Id="SiteSettings"
    GroupId="Customization"
    Location="Microsoft.SharePoint.SiteSettings"
    Sequence="106"
    Title="Custom Litware Site Setting Command">
    <UrlAction Url="/_layouts/LitwareFeatureLab.aspx?Command=SiteSettingCommand"/>
  </CustomAction>
  <!-- Add Command to Site Actions Dropdown -->
  <CustomAction Id="SiteActionsToolbar"
    GroupId="SiteActions"
    Location="Microsoft.SharePoint.StandardMenu"
    Sequence="1000"
    Title="Litware Custom Action"
    Description="Custom Litware Site Action"
    ImageUrl="/_layouts/images/ACL16.GIF">
    <UrlAction Url="/_layouts/LitwareFeatureLab.aspx?Command=SiteActionCommand"/>
  </CustomAction>
  <!-- Document Library Toolbar New Menu Dropdown -->
  <CustomAction Id="DocLibNewToolbar"
    RegistrationType="List"
    RegistrationId="101"
    GroupId="NewMenu"
    Rights="ManagePermissions"
    Location="Microsoft.SharePoint.StandardMenu"
    Sequence="1000"
    Title="Litware Custom New Command"
    Description="This command creates a new Litware doc"
```

```

    ImageUrl="/_layouts/images/ACL16.GIF">
    <UrlAction Url="/_layouts/LitwareFeatureLab.aspx?Command=NewDocCommand"/>
</CustomAction>
<!-- Document Library Toolbar Actions Menu Dropdown -->
<CustomAction Id="DocLibActionsToolbar"
    RegistrationType="List"
    RegistrationId="101"
    GroupId="ActionsMenu"
    Location="Microsoft.SharePoint.StandardMenu"
    Sequence="1000"
    Title="Litware Command on Document Library"
    Description="This command performs a custom command on the document
    library"
    ImageUrl="/_layouts/images/ACL16.GIF">
    <UrlAction Url="/_layouts/LitwareFeatureLab.aspx?Command=DocLibCommand"/>
</CustomAction>
<!-- Per Item Dropdown (ECB)-->
<CustomAction
    Id="ECBItemToolbar"
    RegistrationType="List"
    RegistrationId="101"
    Type="ECBItem"
    Location="Bugworkaround:LocationShouldEqualEditControlBlock"
    Sequence="106"
    Title="Litware ECB Item Command">
    <UrlAction Url="/_layouts/LitwareFeatureLab.aspx?Command=SiteSettingCommand"/>
</CustomAction>
</Elements>

```

In Feature.xml, we add an *ElementManifest* element for Lightup.xml as follows:

```

<ElementManifests>
    <ElementManifest Location="Lightup.xml"/>
</ElementManifests>

```

After building the project, we can install the feature using a batch file similar to the Install.cmd file shown earlier, but with one small difference. This batch file also copies an application page named LitwareFeatureLab.aspx to the _layouts directory. You might notice that this page is referenced in the *UrlAction* attribute in all the *CustomAction* elements in Lightup.xml.

After the feature has been installed, we can determine that it is working properly by examining the Site Actions menu. We should see the Litware Custom Action command as shown in Figure 5-6.

Go back and examine each of the *CustomAction* elements in Lightup.xml. Each of these commands shows up in the Windows SharePoint Services user interface. We would need to add a document library to the site to test the last three custom actions.

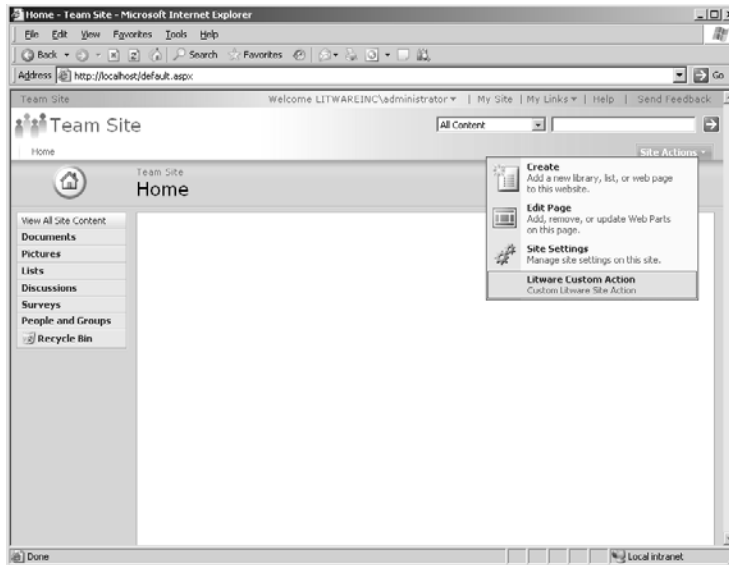


Figure 5-6 The Site Actions menu with a custom action added through a feature

Creating a Callback Receiver Class for a Feature

Now we'll write and configure a custom receiver class that fires custom event handlers whenever our feature is activated or deactivated within a site. We'll use the class library DLL project named `LitwareFeatureLab` as a starting point. This project has been configured to compile the output assembly, `LitwareFeatureLab.dll`, with a strong name and install it into the Global Assembly Cache (GAC) any time the project is built.

We need a reference to `Microsoft.SharePoint.dll` so that code in the `LitwareFeatureLab` project can use the core Windows SharePoint Services types. The receiver class is contained in the source file, `LitwareFeatureLabReceiver.cs`. The following implementation changes the current site's title and description each time the feature is activated or deactivated.

```
using System;
using Microsoft.SharePoint;

namespace LitwareFeatureLab {
    public class LitwareFeatureLabReceiver : SPFeatureReceiver {
        public override void FeatureInstalled(SPFeatureReceiverProperties
            properties){}
        public override void FeatureUninstalling(SPFeatureReceiverProperties
            properties){}
    }
}
```

```

public override void FeatureActivated(SPFeatureReceiverProperties
properties) {
    SPWeb web = (SPWeb)properties.Feature.Parent;
    web.Title = "Litware Feature Lab";
    web.Description = "Litware Feature Lab activated at " +
        DateTime.Now.ToLongTimeString();
    web.Update();
}

public override void FeatureDeactivating(SPFeatureReceiverProperties
properties) {
    SPWeb web = (SPWeb)properties.Feature.Parent;
    web.AllowUnsafeUpdates = true; // this is required to prevent
        //errors in beta 1
    web.Title = "Plain Old Team Site";
    web.Description = "Litware Feature Lab deactivated at " +
        DateTime.Now.ToLongTimeString();
    web.Update();
}
}
}
}

```

Now we need to modify Feature.xml, using the *ReceiverAssembly* and *ReceiverClass* attributes, to indicate that there is a source file named LitwareFeatureLabReceiver.cs.

```

<?xml version="1.0" encoding="utf-8" ?>
<Feature Id="B0FFE3F6-7F28-45ed-BF94-0B597EE95E53"
    Title="Litware Feature Lab"
    Description="This feature demonstrates Hello world functionality."
    Version="1.0.0.0"
    Scope="web"
    ReceiverAssembly="LitwareFeatureLab, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=d4e5777b16a5749f"
    ReceiverClass="LitwareFeatureLab.LitwareFeatureLabReceiver"
    Hidden="FALSE"
    xmlns="http://schemas.microsoft.com/sharepoint/">
    <ElementManifests>
        <ElementManifest Location="Lightup.xml"/>
    </ElementManifests>
</Feature>

```

We can now run Install.cmd again to install the feature with the receiver functionality. We can test to make sure the functionality is working by going to the Site Features page and deactivating and reactivating the feature. If the site title and description change, as shown in Figure 5-7, we know the receiver callback functionality is working properly.

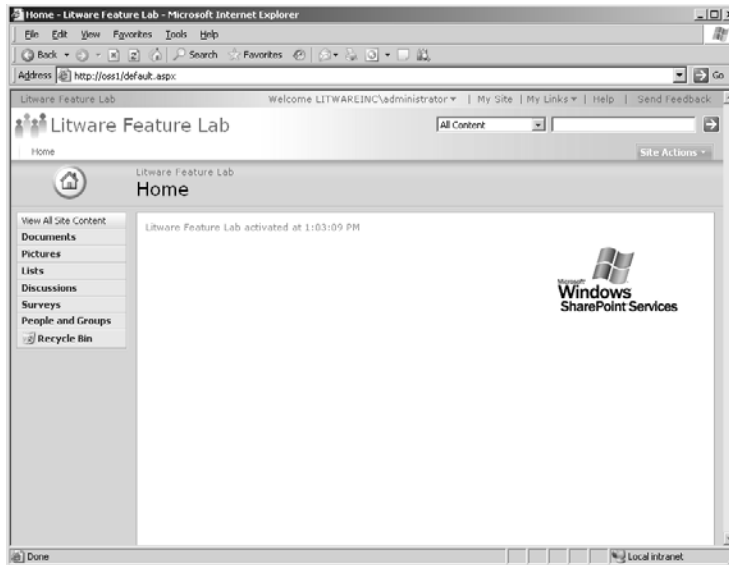


Figure 5-7 A demonstration of receiver functionality in a feature

Adding a Document Library on Activation

Let's say that we want to modify the Litware feature by changing the *FeatureActivated* event handler method so that it creates a document library named Weekly Timesheets. You could add the following code to the method as a starting point:

```
SPListTemplate template = web.ListTemplates["Document Library"];
Guid docLibID = web.Lists.Add("Weekly Timesheets", "Library for Litware Weekly
    Timesheets Documents", template);
SPList docLib = web.Lists[docLibID];
docLib.OnQuickLaunch = true;
docLib.Update();
```

You could also add complementary code to the *FeatureDeactivating* event handler to remove the document library named Weekly Timesheets if it exists:

```
try
{
    Guid docLibID = web.Lists["Weekly Timesheets"].ID;
    web.Lists.Delete(docLibID);
}
catch (Exception ex) { /* do nothing */ }
```

Adding an Event Handler to the Timesheets List

Another change we could make to this feature is to add an event handler to the Timesheets list. We'd start by creating a new class, named *TimesheetEventReceiver*, and then implement this class as an item event handler by inheriting from *SPItemEventReceiver* and overriding *ItemAdding*, *ItemAdded*, *ItemUpdating*, and *ItemUpdated*.

We could implement *ItemAdded* and *ItemUpdated* to generate a unique TimesheetID for the current item by concatenating a string that contains the values of the Consultant, Project, and Submitted On columns. After this unique string is generated, its value is assigned to the current item's TimesheetID column. The code needs to call the *Update* method on the *SPListItem* object to save the changes.

We could also implement *ItemAdding* and *ItemUpdating* to perform validation of the value entered by the user for the Submitted On date. The validation would make sure that the date entered is not a day in the future. If the user tries to enter a time sheet item for a future date, the action is cancelled and an error message is returned.

We would also modify the *FeatureActivated* event handler method to register event handlers for each of the four events in the *TimesheetEventReceiver* class. We could use code that looks like the following, which refers to *ItemUpdated*.

```
SPList list = web.Lists["Timesheets"];
list.EventReceivers.Add(
    SPEventReceiverType.ItemUpdated,
    "LitwareFeatureLabSolution, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=d4e5777b
    "LitwareFeatureLabSolution.TimesheetEventReceiverSolution");
list.Update();
```

Here's the code for the *TimesheetEventReceiver* class, showing the string concatenation and validation.

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

namespace LitwareFeatureLabSolution
{
    public class TimesheetEventReceiverSolution : SPItemEventReceiver
    {
        public override void ItemAdding(SPItemEventProperties properties)
        {
            base.ItemAdding(properties);
            ValidateChange(properties);
        }
    }
}
```

```

public override void ItemUpdating(SPItemEventProperties properties)
{
    base.ItemUpdating(properties);
    ValidateChange(properties);
}

private void ValidateChange(SPItemEventProperties properties) {
    SPWeb web = properties.Openweb();
    SPListItem timesheet =
        web.Lists[properties.ListId].GetItemById(properties.ListItemId);
    // check to make sure date is not day in future
    if (Convert.ToDateTime(timesheet["Submitted On"]).CompareTo(DateTime.Today) > 0)
    {
        properties.ErrorMessage = "You cannot enter timesheet for days in future";
        properties.Cancel = true;
        return;
    }
}

public override void ItemAdded(SPItemEventProperties properties) {
    base.ItemAdded(properties);
    GenerateID(properties);
}

public override void ItemUpdated(SPItemEventProperties properties) {
    base.ItemUpdated(properties);
    GenerateID(properties);
}

private void GenerateID(SPItemEventProperties properties) {
    SPWeb web = properties.Openweb();
    SPListItem timesheet =
        web.Lists[properties.ListId].GetItemById(properties.ListItemId);
    // generate TimesheetID by concatenating Consultant, Project
    // and Submitted On columns
    timesheet["TimesheetID"] = timesheet["Consultant"] + " " +
        timesheet["Project"] + " " +
        Convert.ToDateTime(timesheet["Submitted
        On"]).ToShortDateString(); ;
    timesheet.Update();
}
}
}
}

```

And here is the code for the updated receiver class:

```

using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.SharePoint;

```

```

namespace LitwareFeatureLabSolution
{
    public class LitwareFeatureLabReceiverSolution : SPFeatureReceiver
    {
        public override void FeatureInstalled(SPFeatureReceiverProperties
        properties)
        {

        }

        public override void FeatureUninstalling(SPFeatureReceiverProperties
        properties)
        {

        }

        public override void FeatureActivated(SPFeatureReceiverProperties
        properties)
        {
            SPWeb web = (SPWeb)properties.Feature.Parent;
            web.Title = "Litware Feature Lab Solution";
            web.Description = "Litware Feature Lab Solution activated at " +
                DateTime.Now.ToLongTimeString(); ;
            web.Update();
            SPListTemplate template = web.ListTemplates["Document Library"];
            Guid docLibID = web.Lists.Add("Weekly Timesheets", "Library for
                Litware weekly Timesheets Documents", template);
            SPList docLib = web.Lists[docLibID];
            docLib.OnQuickLaunch = true;
            docLib.Update();

            SPList list = web.Lists["Timesheets"];
            list.OnQuickLaunch = true;
            list.EventReceivers.Add(SPEventReceiverType.ItemAdding,
                "LitwareFeatureLabSolution, Version=1.0.0.0, Culture=neutral,
                PublicKeyToken=d4e5777b16a5749f",
                "LitwareFeatureLabSolution.TimesheetEventReceiverSolution");
            list.EventReceivers.Add(SPEventReceiverType.ItemAdded,
                "LitwareFeatureLabSolution, Version=1.0.0.0, Culture=neutral,
                PublicKeyToken=d4e5777b16a5749f",
                "LitwareFeatureLabSolution.TimesheetEventReceiverSolution");

            list.EventReceivers.Add(SPEventReceiverType.ItemUpdating,
                "LitwareFeatureLabSolution, Version=1.0.0.0, Culture=neutral,
                PublicKeyToken=d4e5777b16a5749f",
                "LitwareFeatureLabSolution.TimesheetEventReceiverSolution");

            list.EventReceivers.Add(SPEventReceiverType.ItemUpdated,
                "LitwareFeatureLabSolution, Version=1.0.0.0, Culture=neutral,
                PublicKeyToken=d4e5777b16a5749f",
                "LitwareFeatureLabSolution.TimesheetEventReceiverSolution");
            list.Update();
        }
    }
}

```



```
public override void FeatureDeactivating(SPFeatureReceiverProperties
    properties)
{
    SPWeb web = (SPWeb)properties.Feature.Parent;
    web.AllowUnsafeUpdates = true;
    web.Title = "Plain Old Team Site";
    web.Description = "Litware Feature Lab Solution deactivated at " +
        DateTime.Now.ToLongTimeString(); ;
    web.Update();
    try
    {
        Guid docLibID = web.Lists["Weekly Timesheets"].ID;
        web.Lists.Delete(docLibID);
    }
    catch (Exception ex) { }
    try {
        SPList list = web.Lists["Timesheets"];
        list.OnQuickLaunch = false;
        list.Update();
    }
    catch(Exception ex){}
}
}
```


Windows SharePoint Services Core Development

In this chapter:

Top-Level Classes	101
Using the Windows SharePoint Services Object Model and the Data in a List ..	107
Adding Word Documents to a Document Library	114

In this chapter, we'll review some of the primary classes in the Microsoft Windows SharePoint Services server-side object model that provide access to objects representing the various aspects of a SharePoint site. We'll then demonstrate some of these classes and their members in action, in examples that take data from a list, serialize the data to XML, add the data to a Microsoft Office Word 2007 document, and store the document in a document library.

Top-Level Classes

The *Microsoft.SharePoint* and *Microsoft.SharePoint.Administration* namespaces provide types and members for working with lists and sites and for managing a server or collection of servers that run Windows SharePoint Services. The following table lists the four major top-level classes in these namespaces. Starting with one of these classes, you can work through the object model to the class you need to use to work with lists and Web sites or to manage one or more servers.

Class	Description
<i>SPGlobalAdmin</i>	Used for central configuration settings. In particular, you can use this class to enumerate the list of virtual servers in Internet Information Services (IIS) and access <i>SPVirtualServer</i> objects.
<i>SPVirtualServer</i>	Represents a virtual server, and is used for server-wide configuration settings. Use this class to create, delete, and access sites under a specific virtual server.
<i>SPSite</i>	Represents a site collection (a top-level site and its subsites), and is used for managing existing sites and for accessing site and list data.
<i>SPWeb</i>	Represents an individual site, and is used for working with the lists, files, and security of the site, including users, site groups, cross-site groups, and permissions.

Depending on which part of a Windows SharePoint Services deployment you need to customize and the type of application you are creating, you can use different means of entry into the Windows SharePoint Services object model to obtain the higher-level object you need to start with. The *SPGlobalAdmin* class provides access to the highest-level objects in the object hierarchy. These objects are generally related to administrative operations. You can also use the *SPGlobalAdmin* class to reach objects that are lower in the hierarchy. The *SPGlobalAdmin* class includes a constructor that you can use to obtain an *SPGlobalAdmin* object to customize global administrative settings in a deployment.

To perform actions on data within a SharePoint site, you must first obtain an *SPWeb* object, which serves as the starting point for accessing lists, items, documents, users, alerts, and so on. For example, to return a collection of lists for a site within an HTTP context, you must first obtain the *SPWeb* object by passing the current *System.Web.HttpContext* object to the *GetContextWeb* method of the *SPControl* class, as shown in this example:

```
SPWeb mySite = SPControl.GetContextWeb(Context)
```

You would then use *mySite.Lists* to get the collection of lists for the site. Similarly, *mySite.Title* returns the title of the site, and *mySite.Users* returns the users on the site.

If you are creating a Web Part, Web service, or Web application to work with site collections, individual sites, or lists, use the *GetContextSite* or *GetContextWeb* method of the *SPControl* class to obtain the current site collection or site. When you create a Web application in the *_layouts* directory, the application's functionality becomes available to all sites on the Web server. Outside an HTTP context, such as in a console application, use the constructor of the *SPSite* class to obtain a specified site collection.

Here are descriptions of some of the properties and methods of these and related classes that you are likely to use when creating an application for a Windows SharePoint Services deployment:

- The *ContentDatabases* property of the *SPVirtualServer* class returns an *SPContentDatabaseCollection* object that represents the collection of content databases used for a virtual server. Each *SPContentDatabase* object provides access to properties of the content database.
- The *WebServers* property of the *SPGlobalConfig* class returns an *SPWebServerCollection* object representing the collection of front-end Web servers in the Windows SharePoint Services deployment. Each *SPWebServer* object provides access to properties of the Web server.
- The *VirtualServers* property of the *SPGlobalAdmin* class provides access to an *SPVirtualServerCollection* object representing all the virtual servers in the Windows SharePoint Services deployment. The *OpenVirtualServer* method of the *SPGlobalAdmin* class returns a specific virtual server. Each *SPVirtualServer* object has members that can be used to manage the virtual server. The *Sites* property provides access to the *SPSiteCollection*

object representing the collection of site collections on the virtual server, and the *Add* method is used to create top-level site collections.

- Each *SPSite* object represents a site collection and has members that can be used to manage the site collection. The *AllWebs* property provides access to the *SPWebCollection* object that represents the collection of sites within the site collection, including the top-level site. The *OpenWeb* method of the *SPSite* class returns a specified site.
- Each site collection includes any number of *SPWeb* objects. Each object has members that can be used to manage a site (including its template and theme) and also to access files and folders on the site. The *Webs* property returns an *SPWebCollection* object representing all the subsites of a specified site, and the *Lists* property returns an *SPListCollection* object representing all the lists in the site.
- Each *SPList* object has members for managing the list or for accessing items in the list. The *GetItems* method can be used to perform queries that return specific items. The *Fields* property returns an *SPFieldCollection* object representing the fields (or columns) in the list. The *Items* property returns an *SPListItemCollection* object representing the items (or rows) in the list.
- Each *SPField* object has members that contain settings for the field.
- Each *SPListItem* object represents a single row in the list.



Note Most classes in the *Microsoft.SharePoint* and *Microsoft.SharePoint.Administration* namespaces start with *SP*. Generally, classes in the Windows SharePoint Services assembly that don't start with this prefix represent Web form controls.

Figure 6-1 on the next page shows the relationship between the top-level classes in the Windows SharePoint Services object model and server and site architecture.

Updating Object Properties

Windows SharePoint Services does not usually save modifications of object properties to the database until you call the *Update* method on the object. The following example shows how to change the title and description for the Tasks list:

```
SPList myList = myWeb.Lists["Tasks"];
myList.Title="New_Title";
myList.Description="List_Description";
myList.Update();
```



Note An exceptional case is when a document is checked out from a document library. Metadata for a document that is checked out from a document library cannot be modified using the *Update* method of the *SPListItem* object.

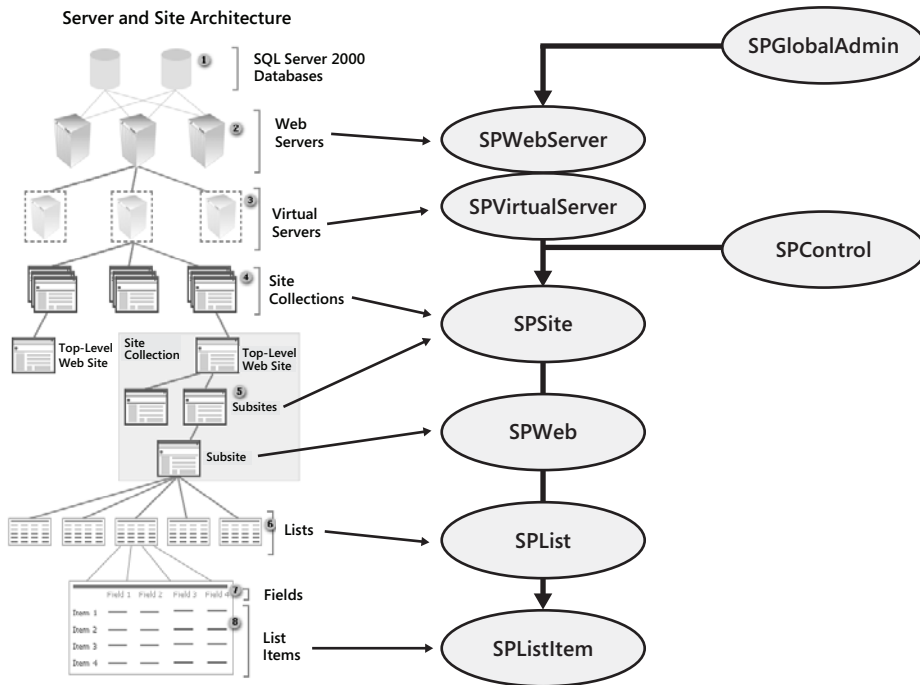


Figure 6-1 Top-level classes in the Windows SharePoint Services object model

Working with Collections

Just as lists are at the center of a SharePoint site, so are collections at the center of its object models. You can use each collection to add, delete, enumerate, and update a type of object. Collection classes generally share the following traits:

- Has a name that ends in “Collection”
- Implements the *System.Collections.ICollection* interface
- Has a *Count* property of type *Int32*
- Has an *Int32* indexer that can be used to get the *n*th item in the collection
- Has an indexer that takes an item identifier
- Has *Add* and *Delete* methods

Calling the *Add* method for a collection usually updates the content database on the server with the appropriate data. In cases in which additional information is required to update data, however, the *Add* method returns an object that you use to gather the information. For example, to add an item to a list, first use the *Add* method of the *SPListItemCollection* class to return an *SPListItem* object, assign values to appropriate properties of that object, and then call the *Update* method to effect changes within the content database.

Using Indexers

Indexers provide a useful means for accessing individual items in collections. To return an item, use square brackets ([]) in Microsoft Visual C# (or parentheses in Microsoft Visual Basic .NET) to contain either an index or a string that identifies the item within the collection.

For example, if *mySite* represents an instance of the *SPWeb* class, *SPList myList = mySite.Lists["Announcements"]* returns the Announcements list. You can then use the *Items* property for the list object to return all the items in the list (*SPListItemCollection myItems = myList.Items*). To return only a subset of items from the list, call one of the list object's *GetItems* methods and pass an *SPQuery* object to specify the subset. For example,

```
SPListItemCollection myItems = myList.GetItems(myQuery)
```

You can use an indexer to return a specific field from a list; for example, *myList.Items["Field_Name"]*. You can also specify a field name in an indexer and iterate through the collection of items in a list to return values from the field. The following example, which would require a *using* directive for the *Microsoft.SharePoint* namespace and a directive for the *Microsoft.SharePoint.Utilities* namespace, displays the Due Date, Status, and Title values for each item in a list:

```
foreach(SPListItem myItem in myItems)
{
    Response.Write(SPEncode.HtmlEncode(myItem["Due Date"].ToString()) + "<BR>");
    Response.Write(SPEncode.HtmlEncode(myItem["Status"].ToString()) + "<BR>");
    Response.Write(SPEncode.HtmlEncode(myItem["Title"].ToString()) + "<BR>");
}
```

The next example shows how to return all Title and Status values for the Tasks list on a SharePoint site. (This example would also require a *using* directive for *Microsoft.SharePoint* and *Microsoft.SharePoint.Utilities*.)

```
SPWeb mySite = SPControl.GetContextWeb(Context);

SPList myTasks = mySite.Lists["Tasks"];
SPListItemCollection myItems=myTasks.Items;

foreach(SPListItem myItem in myItems)
{
    Response.Write(SPEncode.HtmlEncode(myItem["Title"].ToString()) + " :: " +
        SPEncode.HtmlEncode(myItem["Status"].ToString()) + "<BR>");
}
```

You can also use indexers to modify values in a list. In the next example, a list item is added to a collection and the values for a URL column are assigned to the item:

```
SPListItem myNewItem = myList.Items.Add();

myNewItem["URL_Field_Name"] ="URL, Field_Description";

myNewItem.Update();
```



Note The URL field type is unique because it involves two values (separated by a comma and a space), and in the preceding example these values are both assigned to the new item through a single indexer.

The next example sets the Status and Title values for a list item.

```
SPListItem myItem = myItems[0];

myItem["Status"]="Not Started";
myItem["Title"]="Task Title";

myItem.Update();
```



Note An indexer throws an *ArgumentOutOfRangeException* exception if it does not find the specified item.

Determining Where to Build a Custom Application

The scope of an application affects the location where you build an application that runs on Windows SharePoint Services. Custom ASPX pages and Web applications that you want to be accessible from all Web sites in a Windows SharePoint Services deployment should be stored in the following directory, which supports the `_layouts` virtual directory:

```
Local_Drive:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\60\
TEMPLATE\LAYOUTS
```

Pages in this directory are accessible from all Web sites in the Windows SharePoint Services deployment through a URL in the following form:

```
http://Server_Name/[sites]/[Site_Name]/[SubSite_Name]/[...]/_layouts/File_Name.aspx
```

You may also want to create Web applications that contain custom code that works with the global settings for your deployment. For code that involves only the *Microsoft.SharePoint.Administration* namespace for working with global settings in a deployment, Microsoft recommends that the Web application be created on the administrative port. Build and store ASPX pages and Web applications for the administrative port in the following directory:

```
Local_Drive:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\60\
TEMPLATE\ADMIN\1033
```

Pages in this directory are accessible through a URL in the following form:

`http://localhost:Port_#/File_Name.aspx`

Using the Windows SharePoint Services Object Model and the Data in a List

In the following sections, we'll continue to add some functionality to a sample SharePoint site. Litware, our fictitious business, is a consultancy company that requires each of its consultants to fill in a daily time sheet that summarizes the activities the consultant performs during that day. The time sheet data is stored in a custom SharePoint list for later use. At the end of each week, managers review the time sheets to approve or disapprove the hours submitted. To facilitate the review process, we want to generate an Office Word 2007 document that summarizes the work each consultant performs during a week.

In this example, we'll perform queries against the Timesheets SharePoint list and aggregate the results into a custom XML format. The queries will be executed using the Windows SharePoint Services object model, and the results will be aggregated using some custom .NET code. We'll then use the XML Serializer to write the custom XML data to the console for verification. We'll start out by viewing the output in a console application, and then we'll focus on using the data in the Timesheets list to update an Office Word 2007 document. Finally, we'll place this document in a SharePoint document library where managers or other users can view it.

Generating Weekly Time Sheet Aggregate Views

We'll start with a Microsoft Visual Studio 2005 solution named `TimesheetAggregator`. The project contains three files: `Program.cs`, `Project.cs`, and `WeeklyAggregate.cs`.

Before we can access any data in SharePoint, we need to establish a connection to the SharePoint site. When writing code of your own, don't forget that the common terminology and the SharePoint object model don't always match up. Remember that site collections are represented by the `SPSite` object and sites are represented by the `SPWeb` object.

First we add a reference to the `Microsoft.Sharepoint` assembly. Next we create a `Microsoft.SharePoint.SPSite` object in the `Main` method to connect to the default SharePoint site. (The URL for the example is `http://oss1`.) We use the `SPSite` constructor that takes a single string parameter. We access the `SPWeb` object associated with the SharePoint site at `http://oss1` by using the `SPSite`'s `OpenWeb` method. By default this method opens the site associated with the URL provided to the `SPSite` object's constructor.

We then get a reference to the Timesheets list by using the `SPWeb` object's `Lists` property and the name of the list, `Timesheets`. Here is the code from `Program.cs`:

```
static void Main(string[] args)
{
    // get access to the SharePoint list
```

```
SPSite site = new SPSite("http://oss1");
SPWeb web = site.Openweb();
SPList list = web.Lists["Timesheets"];
```

Next we create a class to encapsulate the process of aggregating the time sheet values. The class, named *TimesheetAggregator*, contains a single method, *GenerateAggregation*. This method returns an instance of *WeeklyAggregation*, which is a class that represents the aggregations you need to generate. The *GenerateAggregation* method accepts as parameters a string representing the consultant whose data you need to aggregate and a *DateTime* value that represents a date in the week to aggregate data for. Finally, the method accepts a reference to the *SPList* object containing the data to aggregate.

```
namespace TimesheetAggregator
{
    class TimesheetAggregator
    {
        public WeeklyAggregation GenerateAggregation(string consultant,
            DateTime date, SPList timesheetList)
        {
        }
    }
}
```

Now that we've created the *TimesheetAggregator* class and the *GenerateAggregation* method, we need to write some code that will perform a query against the list to retrieve data for a specific consultant and week. First we calculate the minimum and maximum dates, which represent the Sunday and Saturday surrounding the date parameter. Keep in mind that the *DateTime.DayOfWeek* method returns an enumeration with the value 0 for Sunday, 1 for Monday, and so on. You can use the *DayOfWeek* and *AddDays* methods to help calculate the minimum and maximum dates.

Next we create an instance of an *SPQuery* object. The *SPQuery* object has a constructor that allows you to provide a view that will be used to help initialize the query. In this example, we'll use the All Timesheets view. Next we specify the *Where* section of the query by using the Collaboration Application Modeling Language (CAML) shown in the following code. Notice that this query requires that the consultant's name match one in the list and that the submitted date be between the minimum and maximum dates. *Microsoft.SharePoint.Utilities.SPUtility.CreateISO8601DateTimeFromSystemDateTime* converts a .NET *DateTime* object to a string that is acceptable to the CAML parser.

Finally, the code calls the *SPList.GetItem*s method using the query we've built. This call returns a collection of *SPListItem* objects that represent time sheet entries for the specified consultant and week.

```
public WeeklyAggregation GenerateAggregation(string consultant, DateTime date,
    SPList timesheetList)
{
```

```

// calculate the min and max dates for the week
DateTime minDate = date.AddDays(-(int)date.DayOfWeek);
DateTime maxDate = minDate.AddDays(6);

// retrieve the list items for the consultant and week
SPQuery query = new SPQuery(timesheetList.Views["All Timesheets"]);
query.Query =
    "<where>" +
    "<And>" +
    "<Eq><FieldRef Name='Consultant' /><Value Type='Text'>" +
    consultant +
    "</Value></Eq>" +
    "<And>" +
    "<Geq><FieldRef Name='Submitted_x0020_On' /><Value
    Type='DateTime'>" +
    SPUtility.CreateISO8601DateTimeFromSystemDateTime(minDate) +
    "</Value></Geq>" +
    "<Leq><FieldRef Name='Submitted_x0020_On' /><Value
    Type='DateTime'>" +
    SPUtility.CreateISO8601DateTimeFromSystemDateTime(maxDate) +
    "</Value></Leq>" +
    "</And>" +
    "</And>" +
    "</where>";
SPListItemCollection results = timesheetList.GetItems(query);

```

Now that we have the *SPListItem* objects that apply to the consultant and week in question, we need to group the hours by project. To do this, we write code in classes named *WeeklyAggregation* and *Project* that loops through each *SPListItem* in the collection returned from the query. These classes are designed to serialize into the appropriate XML when submitted to the *System.Xml.Serialization.XmlSerializer* class.

We create an instance of *System.Collections.Generic.Dictionary* by using a string as the key and *Project* as the value. This dictionary will be used to store the grouped project hours. Next we loop through each *SPListItem* in the results from the query and retrieve the values from the list item by using the indexer and by providing the name of the field. Notice in the code that spaces are encoded as `_x0020_`.

If the project hasn't already been added to the dictionary, we create a new instance of a *Project* object, initialize its *Name* and *HourlyRate* properties based on the list item, and then add it to the dictionary. Finally, for each list item, we retrieve the appropriate *Project* object from the dictionary and increment the hours based on the current list item.

```

// create the project aggregates
Dictionary<string, Project> projects = new Dictionary<string, Project>();
foreach (SPListItem listItem in results)
{
    // retrieve the values from the list item
    string project = listItem["Project"].ToString();
    int hours = Convert.ToInt32(listItem["Hours"]);
    decimal hourlyRate = Convert.ToDecimal(listItem["Hourly_x0020_Rate"]);
}

```

```

// check if we already have data for the project
if (!projects.Containskey(project))
{
    // create a new project record
    Project newProject = new Project();
    newProject.Name = project;
    newProject.HourlyRate = hourlyRate;
    newProject.Hours = 0;

    // store the project in the dictionary
    projects.Add(project, newProject);
}

// add the current item's hours to the total
projects[project].Hours += hours;
}

```

At this point, we've retrieved the data and grouped the hours by project. Next we need to build the aggregated totals and *Project* objects into the *WeeklyAggregation* object. The primary focus of this step is to take the groupings from the previous step, copy them into the *WeeklyAggregation* object, and perform some calculations on the data to come up with the totals.

First we create an instance of the *WeeklyAggregation* object and initialize the *Consultant* and *WeekStarting* properties based on the consultant parameter and the *minDate* value calculated earlier. Next we initialize the total hours and total pay to 0 so that these values have a starting point for the totals to be calculated. Because the previous grouping already created the *Project* objects that are contained within the *WeeklyAggregation* object, all we have to do is create a new array and copy the data from the dictionary into the array.

Finally, we need to loop through each item in the projects array and perform two tasks: calculate the project's total pay by multiplying the hours by the hourly rate, and increment the totals in the *WeeklyAggregation* object. Having initialized the *WeeklyAggregation* value, we return it from the *GenerateAggregation* method.

```

// initialize the weekly aggregates object
weeklyAggregation aggregation = new WeeklyAggregation();
aggregation.Consultant = consultant;
aggregation.WeekStarting = minDate;
aggregation.TotalHours = 0;
aggregation.TotalPay = 0.0M;

// copy the project objects from the dictionary and calculate the totals
aggregation.Project = new Project[projects.Count];
projects.Values.CopyTo(aggregation.Project, 0);
foreach (Project project in aggregation.Project)
{
    project.TotalPay = project.Hours * project.HourlyRate;
    aggregation.TotalHours += project.Hours;
    aggregation.TotalPay += project.TotalPay;
}

// return the results

```

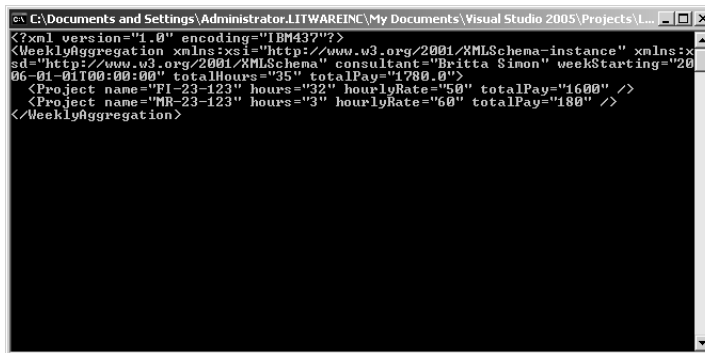
```
    return aggregation;
}
```

We can test the *GenerateAggregation* method by calling it with some hard-coded parameters. For example, we can call *GenerateAggregation* from the *Main* method in the *Program.cs* file using Britta Simon as the consultant and January 2, 2006 as the date. To see the results, we have two options. One is to look at the data in the *WeeklyAggregation* object in the debugger; the other is to serialize the *WeeklyAggregation* object to the console. Here is the serialization code.

```
// perform the aggregations
TimesheetAggregator aggregator = new TimesheetAggregator();
WeeklyAggregation aggregation = aggregator.GenerateAggregation("Britta Simon",
    new DateTime(2006, 1, 7), list);

// write the aggregation to the console
XmlSerializer serializer = new XmlSerializer(typeof(WeeklyAggregation));
serializer.Serialize(Console.out, aggregation);
```

We can test the code by running the console application. Figure 6-2 shows the console displaying an XML document that contains the aggregation of all hours submitted by Britta Simon for the week of January 2, 2006.

A screenshot of a Windows console window. The title bar shows the file path: C:\Documents and Settings\Administrator.LITWAREINC\My Documents\Visual Studio 2005\Projects\... The console displays the following XML output:

```
<?xml version="1.0" encoding="IBM437"?>
<WeeklyAggregation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs
sd="http://www.w3.org/2001/XMLSchema" consultant="Britta Simon" weekStarting="20
06-01-01T00:00:00" totalHours="95" totalPay="1780.0">
  <Project name="FI-23-123" hours="92" hourlyRate="50" totalPay="1600" />
  <Project name="MR-23-123" hours="3" hourlyRate="60" totalPay="180" />
</WeeklyAggregation>
```

Figure 6-2 Data from a SharePoint list serialized to XML

Using the Packaging API to Stuff Weekly Time Sheet Aggregates into Word Documents

In this example, we'll use the *WeeklyAggregation* object created in the previous example, serialize it to XML, and store it in an Office Word 2007 document that is designed to display the total hours for a consultant grouped by project. This summary document will be written to the file system for now, but in the next example it will be stored in a SharePoint document library, where a manager could approve or disapprove the hours submitted.

The first step in generating the time sheet summary document is to store a template document somewhere. In this case, we'll store the document template (a file named `Template.docx`) in a resource that will be compiled into the assembly.

In the project properties window, we select the Resources tab, and then click the link to create a new resource file for the project. We add the `Template.docx` file as a resource by clicking the drop-down arrow on the Add Resource button and selecting Add Existing File. The name of the resource is `Template`.

We next add a `GenerateDocument` method to the `TimesheetAggregator` class. This method should return an array of bytes and accept a `WeeklyAggregation` object as a parameter.

```
public byte[] GenerateDocument(WeeklyAggregation aggregation)
{
}
```

Now we need to read the `Template.docx` resource into a memory stream for use by the program later. To start this process, we create an instance of `System.IO.MemoryStream` to work in. This stream will be used initially to store the template and later the updated document. Next we write the byte array from the resources into the stream using the `Stream.Write` method of the `Stream` class. To access the byte array, we use the `Properties.Resources.Template` property that is generated automatically by Visual Studio 2005 based on the resource we added earlier. Finally, we need to be sure to reset the stream's location to 0 by using the `Stream.Seek` method so that future reads will start from the beginning.

```
// create a memory stream to process the file in
using (MemoryStream stream = new MemoryStream())
{
    // read the template from the library and load it into the stream
    byte[] template = Properties.Resources.Template;
    stream.Write(template, 0, template.Length);
    stream.Seek(0, SeekOrigin.Begin);
}
```

Having read the template into the stream, we need to open it by using the packaging APIs in `System.IO.Packaging` and then modify the `timesheetData.xml` part. We need to add a reference to the `WindowsBase` assembly that is part of WinFX. This assembly is located in the `C:\WINDOWS\msagent\Windows\v6.0.5070` folder.

We next create an instance of a `System.IO.Packaging.Package` object by using the static method `ZipPackage.Open` and the stream from the previous step. The file needs to be opened with read and write access. Next we create a new relative `Uri` object with a path of `/customXML/timesheetData.xml`. Finally, the code opens the file part using the `Package.GetPart` method and stores the `PackagePart` object for later.

```
// open the word document and find the timesheet chunk
Package file = ZipPackage.Open(stream, FileMode.Open, FileAccess.ReadWrite);
Uri xmlPartUri = new Uri("/customXML/timesheetData.xml", UriKind.Relative);
PackagePart xmlPart = file.GetPart(xmlPartUri);
```

The last step in modifying the template document is to write data to the XML *PackagePart* by using the *XmlSerializer* class. To do this we need to get access to the part and then write the new data to the part's stream. We first create an instance of the *System.Xml.Serialization.XmlSerializer* class for the *WeeklyAggregation* class. Next we retrieve a stream object from the XML part by using the *PackagePart.GetStream* method, making sure that the object is opened for reading and writing. We then serialize the aggregation data to the stream retrieved from the *PackagePart.GetStream* call. The length of the stream should be set to 0 before writing. If it isn't, extra data may be left at the end of the stream if the new document is smaller than the old one.

```
// serialize the weekly aggregation to the part
XmlSerializer serializer = new XmlSerializer(typeof(WeeklyAggregation));
using (Stream xmlData = xmlPart.GetStream(FileMode.Open,
    FileAccess.ReadWrite))
{
    xmlData.SetLength(0); // this makes sure there's nothing left in the
                        // stream
    serializer.Serialize(xmlData, aggregation);
}
```

Now that the updates to the package part are finished, we flush the changes to the stream and close the package by using the *Flush* and *Close* methods of the *Package* object.

```
// close the file
file.Flush();
file.Close();
```

The final step of implementing the *GenerateDocument* method is to return the byte array stored within the *MemoryStream* object. We use the *MemoryStream.ToArray* method to return the internal byte array from the *GenerateDocument* method.

```
// return the byte array containing the data in the stream
return stream.ToArray();
```

Now that we've completed the *GenerateDocument* method, we can write some code in the *Main* method to test it. We call *GenerateDocument* using the output of *GenerateAggregation* and then write the resulting byte array to a file.

```
byte[] document = aggregator.GenerateDocument(aggregation);

// write the file to the file system
using (System.IO.FileStream writer = System.IO.File.Create(@"C:\Exercise 2
    Test.docx"))
    writer.Write(document, 0, document.Length);
```

Figure 6-3 on the next page shows that this version of the application produces the same results as the console application with one exception: the data is stored in a Word file containing a formatted version of the weekly aggregations that is saved on the hard disk.

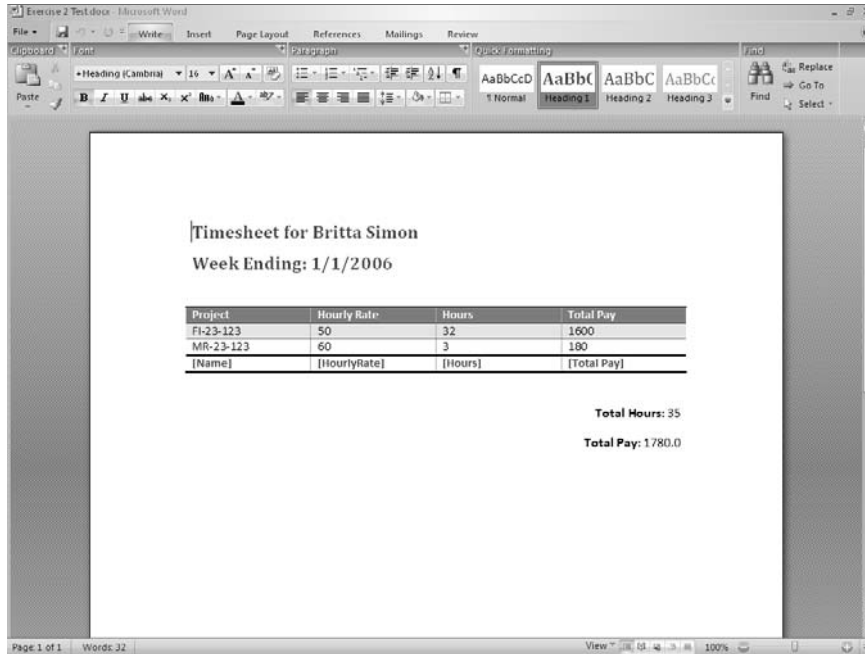


Figure 6-3 The list data is now captured in a formatted Word file.

Adding Word Documents to a Document Library

In this section, we'll take the Word document created in the previous example and store it in the Weekly Timesheets document library so that managers and users have access to it through our SharePoint site.

We need to add code to the *Main* method to access the *SPDocumentLibrary* object for the document library. Gaining access to the library is similar to gaining access to the Timesheets list, in that you need to use the *Lists* property of the *SPWeb* object. The difference is that the *SPList* object is then cast to a specialized list type of *SPDocumentLibrary*.

```
SPDocumentLibrary library = web.Lists["weekly Timesheets"] as
    SPDocumentLibrary;
```

Next we create the *StoreDocument* method for the *TimesheetAggregator* object to encapsulate the creation of the new document in the document library. This method should take three parameters: a byte array named *document*, a string called *documentName*, and a reference to an *SPDocumentLibrary* object named *library*.


```
public void StoreDocument(byte[] document, string documentName,
    SPDocumentLibrary library)
{
}
```

To add a document to the library, we'll perform three steps in implementing the *StoreDocument* method. First we determine the URL of the document library. To do this, we access the *ParentWeb* property of the *SPDocumentLibrary* object to get access to the *SPWeb* object. Then we get the *SPSite* object using the *SPWeb.Site* property. Finally, we call the *SPSite.MakeFullUrl* method to get the full URL based on the library's server-relative URL that is found using *SPDocumentLibrary.RootFolder.ServerRelativeUrl*.

In the next step, we use the URL of the document library and the *documentName* parameter to generate the document path. The third step is to add the file by using the *SPWeb.Files* collection and calling the *Add* method. The parameters needed by this method are the path to the new document, the array of bytes representing the file, and a Boolean flag specifying whether the file should be overwritten if it already exists.

```
// generate the document path
string libraryPath = library.ParentWeb.Site.MakeFullUrl
    (library.RootFolder.ServerRelativeUrl);
string documentPath = libraryPath + "/" + documentName;

// add the document to the library
library.ParentWeb.Files.Add(documentPath, document, true);
```

To use the completed *StoreDocument* method, we can call it from the *Main* method. First we generate a unique file name based on the *Consultant* and *WeekStarting* parameters from the *WeeklyAggregation* objects. Next we call *StoreDocument* using the document byte array from the *GenerateDocument* method, the document name, and the document library named *WeeklyTimesheets*.

```
string documentName = string.Format("{0} {1:MM-dd-yyyy}.docx",
    aggregation.Consultant, aggregation.WeekStarting);
aggregator.StoreDocument(document, documentName, library);
```

We can now run the application and verify that the output is in the *Weekly Timesheets* document library.

Using an Event Handler to Generate Weekly Aggregate Documents

Having written the components to aggregate time sheets, we can now automate the process. Every time an item in a list is changed, Windows SharePoint Services fires an event that you can capture and use to perform processing. In this section, we'll take the code written in the previous examples and use it to update the weekly aggregate time sheets each time a related time sheet entry is made or updated.

The first step is to create a new class, named *TimesheetEventHandler*, to contain the implementation of the SharePoint list event handler. The only change needed to the event handler class is to make it derive from *SPItemEventReceiver*.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.SharePoint;
namespace TimesheetAggregator
{
    public class TimesheetEventHandler : SPItemEventReceiver
    {
    }
}
```

Next we need to override the *ItemAdded* and *ItemUpdated* methods and use their parameters to get access to the *SPListItem* object that has changed. In both methods, we need to get access to the *SPWeb* object containing *SPListItem*. To do this, we use the *SPItemEventProperties* object and call its *OpenWeb* method. Finally, we gain access to the *SPListItem* object by using the *SPWeb.Lists* property, and then use *SPList.GetItemById* to get the *SPListItem* object that has changed. The *ListId* and the *ListItemId* properties in the *SPItemEventProperties* object contain the IDs for the list and list item that have changed.

```
public override void ItemAdded(SPItemEventProperties properties)
{
    base.ItemAdded(properties);

    // get access to the list item that has been changed
    SPWeb web = properties.OpenWeb();
    SPListItem listItem = web.Lists[properties.ListId].
        GetItemById(properties.ListItemId);
}
public override void ItemUpdated(SPItemEventProperties properties)
{
    base.ItemUpdated(properties);

    // get access to the list item that has been changed
    SPWeb web = properties.OpenWeb();
    SPListItem listItem = web.Lists[properties.ListId].GetItemById
        (properties.ListItemId);
}
```

With references to the *SPWeb* and *SPListItem* objects of the item that has changed, we can perform the operations from the previous examples on *SPListItem*. To perform these operations and share them between both event handler methods, we can put this code into a single method, named *UpdateWeeklyTimesheet*, and have it accept two parameters, the *SPWeb* object and the *SPListItem* object.

We start by retrieving the *SPList* object by using the *SPListItem.ParentList* property and the *SPDocumentLibrary* object by using the *SPWeb.Lists* property, and typecasting *SPList* to an *SPDocumentLibrary* object. Next we retrieve the consultant's name and the submitted-on date

from the list item. Finally, we process the data in the *Main* method, as we did in the previous examples but with one difference. Instead of providing hard-coded consultant and date parameters, we use the values retrieved from the *SPLListItem* object. (We also need to call *UpdateWeeklyTimesheet* from both *ItemAdded* and *ItemUpdated*.)

```
private void UpdateWeeklyTimesheet(SPWeb web, SPLListItem listItem)
{
    // get the list and document library
    SPList list = listItem.ParentList;
    SPDocumentLibrary library = web.Lists["Weekly Timesheets"] as
        SPDocumentLibrary;
    // get the list item information
    string consultant = listItem["Consultant"].ToString();
    DateTime submittedOn = Convert.ToDateTime(listItem["Submitted_x0020_On"]);

    // process the data
    TimesheetAggregator aggregator = new TimesheetAggregator();
    WeeklyAggregation aggregation = aggregator.GenerateAggregation(consultant,
        submittedOn, list);
    byte[] document = aggregator.GenerateDocument(aggregation);
    string documentName = string.Format("{0} {1:MM-dd-yyyy}.docx",
        aggregation.Consultant, aggregation.WeekStarting);
    aggregator.StoreDocument(document, documentName, library);
}
```

Adding the Assembly to the GAC

Because we turned this project into a SharePoint event handler, we need to convert it to a class library and put the assembly into the Global Assembly Cache (GAC). To do this, we open the project properties window in Visual Studio, select the Application tab, and change Output Type to Class Library. Next we sign the assembly using *TimesheetAggregator.snk*. To sign the assembly, we click the Signing tab in the properties window, select the Sign The Assembly option, and select *TimesheetAggregator.snk*. Also, on the Build Events tab of the properties window, we need to add the following command to the post-build event command line section. This line will add the assembly to the GAC after it is built.

```
"%programfiles%\Microsoft Visual Studio 8\SDK\v2.0\Bin\gacutil.exe"
-if $(TargetDir)\TimesheetAggregator.dll
```

Registering the Event Handler Assembly

We also need to register the event handler, and one way to do this is to use the SharePoint object model to write a simple application that will install the handler. We'll do this using a console application project, named *TimesheetAggregatorInstaller*, that we can add to the solution.

We need to add entries to the *SPList.EventReceiver* collection, which is where an assembly and class are associated with an event fired by the list. The first step is to open the *SPList* object in

the same manner as we did in the previous examples—open the site (<http://oss1>) and get a reference to the Timesheets list.

We next add entries to the *SPList.EventReceivers* list. One entry is for the *ItemAdded* event and the other for *ItemUpdated*. For the assembly and class names, we use the values shown in the code example that follows. We can then build the application and run it to install the event handlers.

```
// get access to the SharePoint list
SPSite site = new SPSite("http://oss1");
SPWeb web = site.OpenWeb();
SPList list = web.Lists["Timesheets"];

// add the event handler
list.EventReceivers.Add(SPEventReceiverType.ItemAdded,
    "TimesheetAggregator, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=6479e6da8b088665",
    "TimesheetAggregator.TimesheetEventHandler");
list.EventReceivers.Add(SPEventReceiverType.ItemUpdated,
    "TimesheetAggregator, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=6479e6da8b088665",
    "TimesheetAggregator.TimesheetEventHandler");
list.Update();
```

In the examples demonstrated in this chapter, the code was run under the Administrator account, so all of the object model operations were performed using an account that had full control of the <http://oss1> site. However, the event handler runs under the account of the application pool, Network Service, which means we need to give the Network Service account Contributor permissions to the <http://oss1> site. To grant permission to this account, follow these steps:

1. Open <http://oss1> (or your top-level site), open the Site Settings page, and then click the Advanced Permissions link.
2. Click Add Users, enter **Network Service**, and give that account Contributor permissions.

Finally, it's time to test the application by navigating to the Timesheets list and creating a new entry. We've used the consultant John Rodman, project FI-23-123, and the date 1/2/2006. We can verify that the event handler is working by navigating to the Weekly Timesheets document library and checking whether the new summary document was created, and Figure 6-4 shows that it was.



Note There is another way to install an event handler. The information about which event to handle and what assembly will handle it can also be installed in a feature. For more information about features, see Chapter 5, "Working with Features in Windows SharePoint Services."

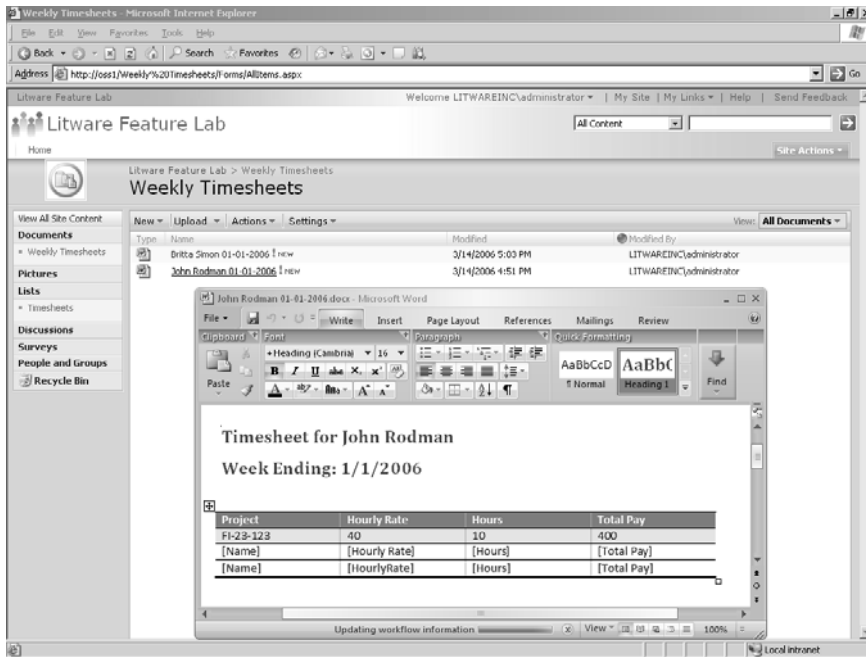


Figure 6-4 The time sheet summary document automatically generated in a document library

Creating Workflows: The Missing Piece of Office Productivity

In this chapter:

Workflows and Activities	121
Windows Workflow Foundation Run-Time Engine	123
Building Custom Workflows	123
Installation and Deployment	125
Workflow Stages	125
Workflows in Action	128

Not so long ago, the final output of a business productivity application was a paper document. As an element or outcome of a business process, that document was then mailed, faxed, or placed in an in box for activities such as review, approval, or archiving.

Information workers who created and worked with these documents were part of often loosely defined business processes. Those processes have remained elusive, and until now they have been difficult to define with any degree of precision or reproducibility. Even when e-mail became the distribution mechanism of choice, once an individual finished his or her tasks with a document, there was still the question “What happens next?”

With the 2007 Microsoft Office system, information workers and developers who build applications for them can answer that question by associating a workflow and workflow activities to a document library, list, or content type that is part of a Microsoft Windows SharePoint Services site. Workflows provide a solution to automating business processes related to document creation and management. In the future, many documents authored in Microsoft Office applications will be part of some kind of formal workflow process.

Workflows and Activities

The workflow functionality in Windows SharePoint Services is based on the Windows Workflow Foundation (WF), a set of base workflow technologies that programmers can access through WinFX. WF also provides tools for the development and execution of workflow-based applications.

There are two ways to think of workflows. Both types are supported by WF, and you can create either type for Windows SharePoint Services.

- *Sequential workflows* are modeled as flowcharts and are well suited for system-oriented workflows. A sequential workflow represents a sequence of steps that occur in order until the last step is completed. However, sequential workflows can be affected by external events and include parallel logical paths, so the precise order in which activities are executed can vary.
- *State machine workflows* are modeled as state diagrams and are best suited for human-based workflows. A state machine workflow represents a set of states, transitions, and actions. One state is denoted the start state, and based on an event, a transition is made to another state. The state machine can have a final state that determines the end of the workflow. State machine workflows are effective at capturing processes that can be changed as they run. For example, a bank manager might override and approve a loan application for a customer with a bad credit history on the basis of specific circumstances that are not modeled in the original workflow.

The two main concepts in WF are the workflow and the activity. A *workflow* represents a coordinated, event-driven set of activities and is compiled into a .NET assembly. An *activity* is a .NET class written in managed code that exposes methods and properties and fires events. A workflow runs by conditionally executing the methods in the activities.

A workflow lets you associate a business process to items on a SharePoint site. For example, you can create a simple workflow that routes a document to a series of users for approval or develop more complex workflows that involve multiple steps, including conditional branching, throughout the life cycle of an item. Workflows can be initiated by a user or designed so that Windows SharePoint Services initiates the workflow when an event occurs; for example, when an item in a list or document library is created or modified.

A workflow can also be added to a content type, and multiple workflows can be associated with a specific item. More than one workflow can run on the same item at the same time, but only one instance of a workflow can run on an item at any given time. For example, you might have workflows named Loan Approval and Document Preparation associated with a content type named Loan Application. Although both workflows can run simultaneously on an item of the Loan Application content type, you can't have two instances of the Loan Approval workflow running on the same item at the same time.

Windows SharePoint Services provides a set of basic workflows and support services that are built on WF. In turn, Microsoft Office SharePoint Server 2007 can make use of Windows SharePoint Services to implement more sophisticated workflows. Examples of workflows include moderation, approval, issue tracking, collecting feedback, and collecting signatures. Some of the built-in activities include task management activities such as creating a task, completing a task, or deleting a task; and item activities such as updating an item or changing an item's status.



Note The workflows and activities just mentioned are accurate at the time of writing but might change by the time the 2007 Microsoft Office system is released.

WF is tightly integrated with Microsoft Exchange and Microsoft Office Outlook. For example, workflows can generate tasks for individuals that will appear in the Outlook task list. Workflows can also send e-mail messages and time out if the message has not been acted on within a set time. The action can then be escalated to another user. The 2007 Microsoft Office system client applications are fully workflow aware. For example, Microsoft Office Word 2007, Microsoft Office Excel 2007, and Microsoft Office PowerPoint 2007 are enabled for workflow initiation, configuration, and completion. Microsoft Office Access 2007 can also take advantage of workflows for reporting.

Windows Workflow Foundation Run-Time Engine

The WF run-time engine manages workflow execution and enables workflows to remain active for long periods of time and survive machine reboots. The WF run-time engine provides services to workflow applications, including sequencing, state management, tracking capabilities, and transaction support. The engine serves as a state machine responsible for loading and unloading workflow templates in addition to managing the current state of any workflows that are running.

WF allows any application process or service container to run workflows by loading WF within its process. For Windows SharePoint Services workflows, SharePoint hosts WF. In place of the pluggable services that are included with WF, Windows SharePoint Services provides its own implementations of services for transactions, persistence, notifications, roles, tracking, and messaging. The functionality of the WF run-time engine, in addition to the hosting functionality that Windows SharePoint Services provides, is exposed through the Windows SharePoint Services object model.

Building Custom Workflows

The basic workflow activities provided by Windows SharePoint Services offer the means to define sophisticated workflows. Sometimes, however, you'll need to build your own activities. For example, within a health care application, a developer might want to send a Health Level Seven (HL7) message to request an X ray or a drug for a patient after it has been approved by a doctor.

Activities can be readily built with managed code in Microsoft Visual Studio 2007. Workflows can be created with Microsoft Office SharePoint Designer 2007 and with the Microsoft Visual Studio 2005 Designer for Windows Workflow Foundation. Workflows can also be customized through the user interface in Windows SharePoint Services itself.

To manage complex workflows, you'll need to create a form or a series of forms for associating and configuring the workflow, initializing the workflow, and for gathering user input required for a workflow. If you build a workflow using Office SharePoint Designer, the forms are generated for you as .aspx pages. If you use Visual Studio, you can also use Microsoft Office InfoPath 2007 forms. Office InfoPath 2007 forms will run inside the browser or natively on the client if Office InfoPath 2007 is installed.

More Info For more information about running Office InfoPath 2007 forms in the browser, see Chapter 9, "Microsoft Office InfoPath 2007 and Microsoft Office Forms Server 2007."

The Visual Studio 2005 Designer for Windows Workflow Foundation is an add-in hosted within Visual Studio 2005. Developers can use this tool to create custom workflows and workflow activities. A workflow can be created using graphical elements, XML specification, code, or a combination of these. The designer provides a graphical design surface that developers can use to easily assemble and configure predefined activities into a custom workflow. Workflow authors use and extend the workflow model in the same way they use and extend other elements of the Microsoft .NET Framework.

The Visual Studio 2005 Designer for Windows Workflow Foundation includes a number of predefined activities you can use to create Windows SharePoint Services workflows. In addition, Windows SharePoint Services provides a number of activities specifically for use in Windows SharePoint Services workflows. These activities are optimized to simplify and streamline development of workflows for use on SharePoint sites.

Because not all the activities available in the Visual Studio 2005 Designer for Windows Workflow Foundation are relevant in the context of SharePoint sites, Windows SharePoint Services supports a subset of the available activities. These include the following:

- **Code** Use this activity to add Visual Basic or C# code to your workflow.
- **ConditionedActivityGroup** Use this activity to perform a set of activities conditionally, based on criteria specific to each activity, until a condition is met for the ConditionedActivityGroup activity as a whole.
- **Scope** Use this activity to logically group activities to provide a framework for transaction and exception handling.
- **Sequence** Use this activity to execute an ordered set of child activities.
- **Replicator** Use this activity to create multiple instances of a single child activity.

Windows SharePoint Services provides specific activities that help in three main areas: creating, updating, completing, and deleting SharePoint tasks; creating, updating, and deleting SharePoint task alerts; and enabling workflow forms within specific scopes to allow users to modify workflows that are in progress.

Installation and Deployment

To install a workflow on a server farm, you need to create a workflow definition, which is an XML file that contains the information Windows SharePoint Services requires to instantiate and run the workflow. This information includes the name, GUID, and description of the workflow; the location of any custom forms used in this workflow; and the correct class within the workflow assembly to call.

Each server farm contains a workflow associations table, and each entry in this table contains data about a workflow's association with a specific content type, list, or document library. This data usually indicates whether the workflow is started automatically or by users and provides the task and history lists associated with the workflow. If a workflow has been added to multiple content types, lists, or document libraries, it will have one entry in the table for each association. Likewise, if a specific content type, list, or document library is associated with multiple workflows, the table includes an entry for each of these workflows.

Workflow Stages

Users, including administrators and end users, interact with workflows at various stages. Among these stages are association, initiation, status, and completion. The following sections describe some of these stages in more detail.

Workflow Association

Workflows are installed at the server level. The administrator of a site collection must activate the workflow to make it available to a specific site collection on that server. Site administrators can then associate a specific workflow with a list, document library, or content type. A workflow association is stored as a property of the specific list or other item. Administrators can also set general workflow parameter information such as

- A unique name for the workflow
- How the workflow is applied to a given item (either automatically when the item is created or changed or manually) and which roles (such as Administrator or Contributor) can start a workflow
- The task list for the workflow to use if it creates tasks
- The history list for the workflow to store history events, as defined by the workflow

In addition, a site administrator can specify custom parameter information for a particular workflow. To capture this information, the workflow must provide its own form that Windows SharePoint Services displays for the administrator.

Workflow Initiation

Any user who has been assigned the appropriate role can initiate a workflow that is configured to run manually. The user selects the item on a SharePoint site and then selects the workflow from the group of workflows associated with that item. The user supplies any information required for the specific workflow and then starts the workflow. Initiating a workflow creates a new workflow instance for that specific item.

Again, for a user to specify custom workflow initiation settings, the workflow must provide its own form. The user specifies the initiation settings on this form, and the settings are passed to the workflow engine, which starts the workflow instance.

Workflow Status

Users can view the progress of a workflow on a selected item. The main document library or list item page displays the status of the workflows running on an item. In addition, each item has a workflow page on which a user can see all workflows currently running on that item, all workflows that have run on the item in the past, and all the workflows available for that item.

When a user initiates a workflow on an item, Windows SharePoint Services adds a column to that item, and the column's name is initially set to the name of the workflow. Windows SharePoint Services uses this column to display the current status of the item within that workflow.

Workflow Task Completion

Workflow stages appear as tasks in a task list. When designing a workflow, its author can specify the task schema, which might include information such as the following:

- Task title
- Name of the person the task is assigned to
- Task status
- Task priority
- Task date due
- A link to the referenced item

As the workflow runs and tasks are created, the user can select the task, mark it complete, and enter any optional or required information. The workflow instance is then notified of changes to workflow tasks and can choose to respond to those changes as specified in the workflow. This response can include moving the item to another stage of the workflow.

The OnWorkflowActivated Activity

All Windows SharePoint Services workflows start with the OnWorkflowActivated activity. (Each new SharePoint workflow application project you create contains an OnWorkflowActivated activity as the first activity by default. However, you still must set the properties of that activity.) This activity initializes the correlation between the workflow instance and the correlation reference so that the WF runtime can direct incoming messages to the workflow instance.

Using the Information in the OnWorkflowActivated Activity

After a Windows SharePoint Services workflow is initiated, the *Properties* property of the OnWorkflowActivated activity returns an *SPWorkflowActivationProperties* object that represents the initial properties of the workflow as it starts, such as the user who added the workflow and the list and item to which the workflow was added. Any activity that needs access to the initialization information of the workflow, such as the workflow ID, can refer to the *Properties* object defined in the OnWorkflowActivated activity.

You can use the initiation properties from any other activity within the workflow; these are especially useful when you are using a Code activity to work with the Windows SharePoint Services item the workflow is running on. For example, you can use the following code to return the *SPListItem* object on which the workflow instance is running. In the code sample, *wfProperties* is the object variable specified for the *Properties* property of the OnWorkflowActivated activity.

```
SPWeb myWeb = new SPSite(wfProperties.SiteId).OpenWeb(wfProperties.WebId);
SPListItem myItem =
    myWeb.Lists[wfProperties.ListId].GetItemById(wfProperties.ItemId);
//Here is where you can alter the item..
myItem.Update();
```

Setting the Properties of the OnWorkflowActivated Activity

When you create a SharePoint workflow application project, an OnWorkflowActivated activity is the first activity by default. Several of the activity's properties need to be specified. Each required property that has not been specified is highlighted with an exclamation-point icon in the properties window. To ensure that the OnWorkflowActivated activity is valid, do the following:

- Set the *Activation* property to True. You must set this property to True for the workflow to initialize properly.
- Specify variables for the following properties:
 - *Properties*
 - *CorrelationReference*
 - *WorkflowId*

How Windows SharePoint Services Processes Workflows

Windows SharePoint Services runs a workflow until the workflow reaches a point at which it cannot proceed until a specific event occurs—for example, a user specifying that a task is complete. At this point, Windows SharePoint Services commits the changes made in the previous Windows SharePoint Services workflow activities. Those changes are batched into a single SQL transaction, an approach that has two major advantages:

- Batching the changes is less performance intensive than committing each change as it occurs as a separate transaction.
- If any of the changes fail when committed, Windows SharePoint Services rolls back the changes to the previous commit point, which decreases the amount of work developers have to do when creating exception handlers and compensation handlers for Windows SharePoint Services workflow activities.

When the event for which the workflow is waiting occurs, the workflow resumes running and runs until another event is reached or the workflow ends, at which point Windows SharePoint Services commits any changes made by Windows SharePoint Services workflow activities that followed the last commit point. This continues until the workflow reaches its conclusion or is stopped by the user.

This approach can affect workflows in which Code activities are interspersed with Windows SharePoint Services–specific workflow activities. Although the write operations for the Windows SharePoint Services workflow activities are batched and not committed until a commit point is reached, write operations that are included in Code activities are committed as they occur. Also, Code activities do not have the implicit rollback functionality of Windows SharePoint Services workflow activities. Exception and compensation handlers must be written for those activities.

Workflows in Action

The sample SharePoint site illustrated in the examples in previous chapters is used by consultants for Litware, a fictitious company, to fill in a time sheet each day to summarize their activities. The time sheet data is stored in a custom SharePoint list for later use. For example, at the end of every week, managers review the time sheets to approve or reject the hours submitted by each consultant. To facilitate this process, an approval process is followed for each weekly summary time sheet. In this section, we'll demonstrate a workflow that creates a task assigned to a consultant's manager and provides the manager with a form that he or she uses to accept or reject the time sheet. We'll first illustrate the built-in approval workflow so that you can see some of the steps required to associate and configure a workflow with an item on a SharePoint site. After that, we'll create our own approval workflow using Visual Studio 2005.

Associating and Activating SharePoint Workflows

As mentioned earlier, Windows SharePoint Services provides several built-in workflow types. One of these is an approval workflow. As the result of user interaction with the workflow, a task is added to a task list on a SharePoint site. The approval workflow also provides Office InfoPath 2007 forms to facilitate interaction with the task. In this section, you'll see how to attach an approval workflow to the Weekly Timesheets document library to facilitate manager approval of time sheets.

Before you can use the approval workflow, you need to enable the Routing Workflows feature on the site collection. Open the site (<http://oss1> in our example) and navigate to the Site Settings page by clicking Site Settings on the Site Actions menu. Under Site Collection Administration, click Site Collection Features. Figure 7-1 shows the Site Collection Features page, which lists built-in workflows among the features. In the features list, find the Routing Workflows feature and be sure that it is activated.

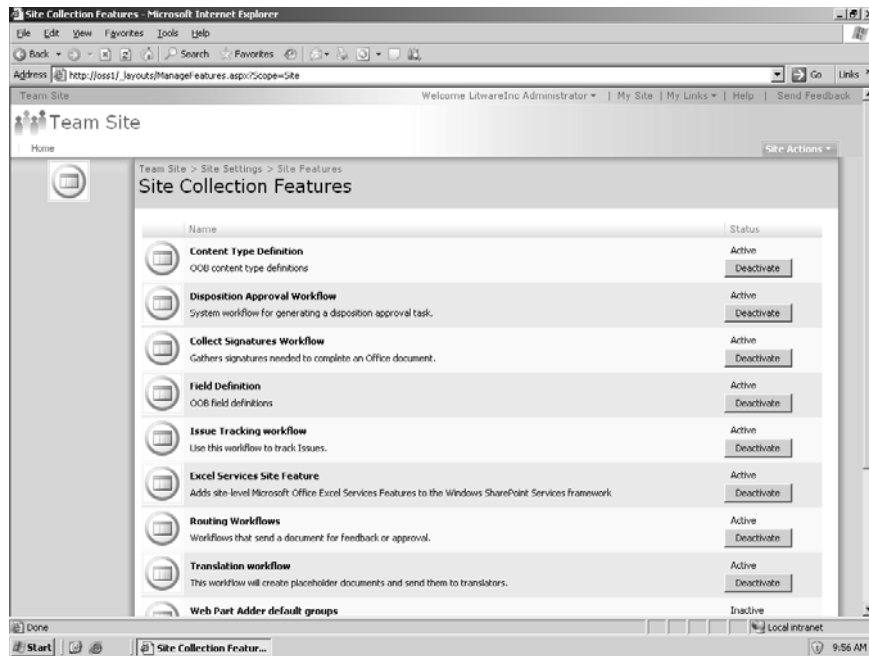


Figure 7-1 To use a built-in workflow, you must activate the workflow on the site collection.

For this example, we also need to create a task list to store workflow-related tasks. To do this, we open the site's complete content list using the View All Site Content link and create the list, choosing Tasks as the list type.

Now that the preliminary work is done, we can associate the approval workflow with the Weekly Timesheet document library. After opening the document library, we use the Document Library Settings page to open the page shown in Figure 7-2, which lets us associate the workflow with the library. On the page, we select the Approval workflow template, name the workflow TimeSheetApproval, and select the task list we created to store the workflow tasks. In this example we also indicate under History List to create a new workflow history list to store the history of the workflow.

The screenshot shows the 'Add a Workflow: Weekly Timesheets' page in a Microsoft Internet Explorer browser window. The page is titled 'Add a Workflow: Weekly Timesheets' and contains several sections for configuring a workflow. The 'Workflow' section has a dropdown menu with 'Approval' selected. The 'Name' section has a text box containing 'TimeSheetApproval'. The 'Task List' section has a dropdown menu with 'Tasks' selected. The 'History List' section has a dropdown menu with 'TimeSheetApproval History (new)' selected. The 'Start Options' section has a checked checkbox for 'Allow this workflow to be manually started by anyone with a permission level of: Contribute'.

Figure 7-2 This page is used to define attributes when adding a workflow to a list or library.

After clicking Next on the page shown in Figure 7-2, we see the page shown in Figure 7-3. On this page, we can configure settings related to the approval process. For our purposes, we're interested only in the Approvers text box. In it we enter **litwareinc\administrator** so that the administrator must approve each time sheet document. We then click OK to complete the steps for setting up the workflow. Now we can test the approval process.

In the Weekly Timesheets document library, we select the Workflows menu item from one of the time sheet documents, and then start the workflow by clicking the TimeSheetApproval link under Start A New Workflow, shown in Figure 7-4.

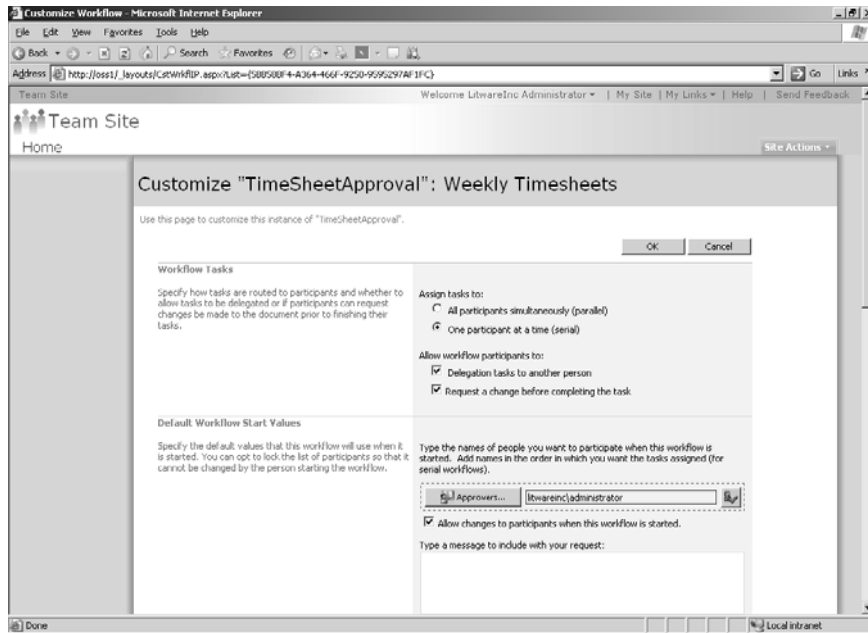


Figure 7-3 This page is used to define aspects of the approval process managed by the workflow.

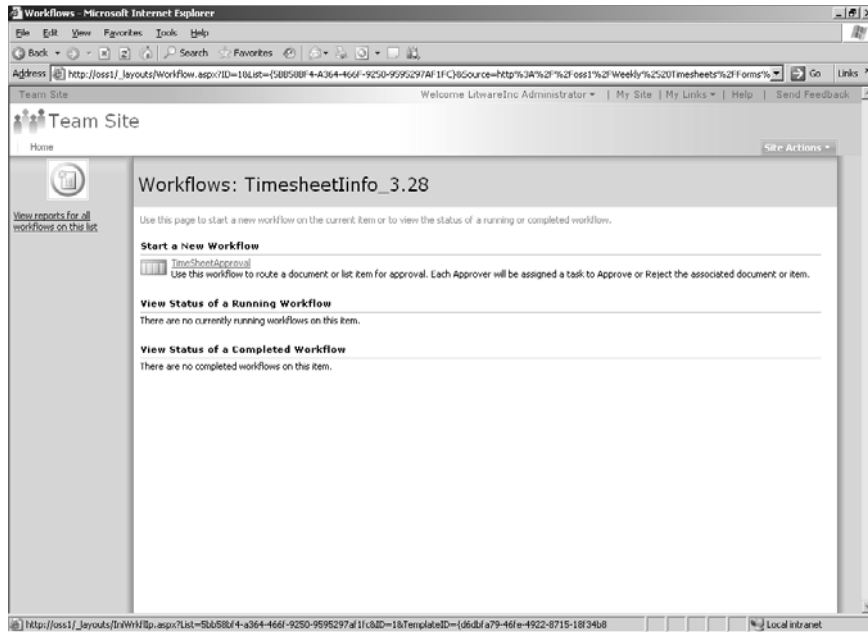


Figure 7-4 Users initiate a workflow using a menu command that opens this page.

The initialization page shown in Figure 7-5 allows us to modify the initialization parameters set when we created the workflow. In this case, we won't override any of these settings, so we simply click Start to initiate the workflow.

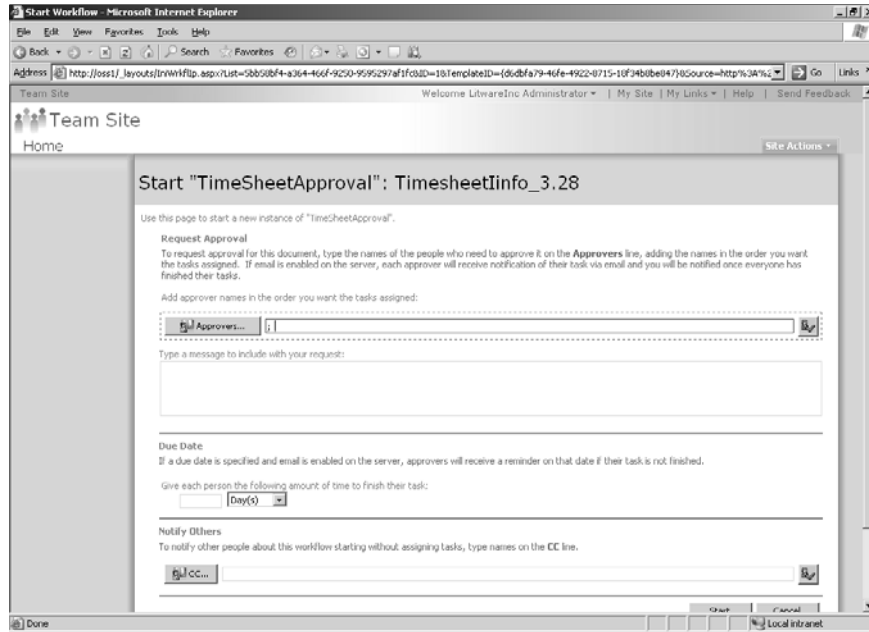


Figure 7-5 The page that starts the workflow

As you can see in Figure 7-6, the workflow has created a task for the LitwareInc Administrator to resolve.

The next step in the workflow is to complete the task so that the workflow can continue. We do that in the Tasks list by editing the properties of the task. On the approval form, shown in Figure 7-7, an approver can enter comments and approve or reject the time sheet.

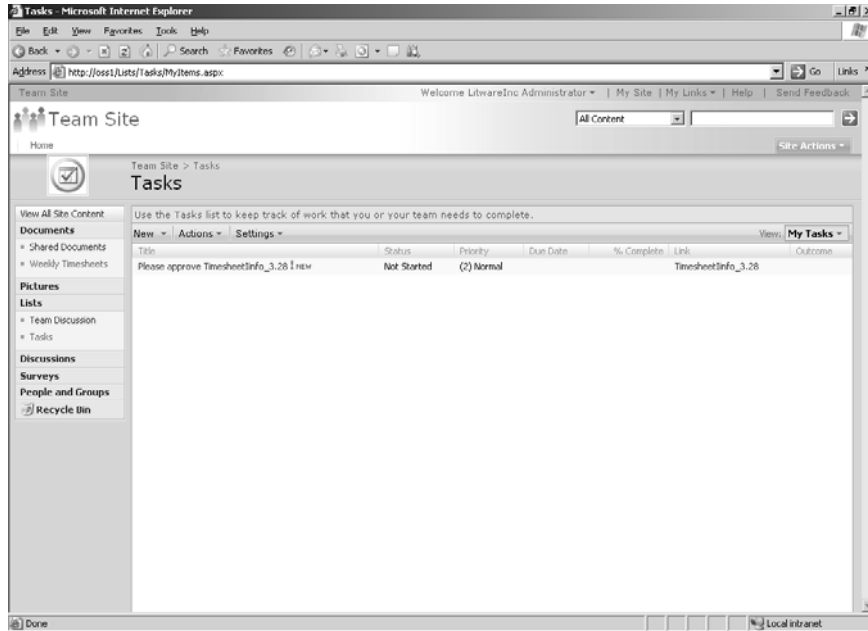


Figure 7-6 A task is created once the workflow is run.

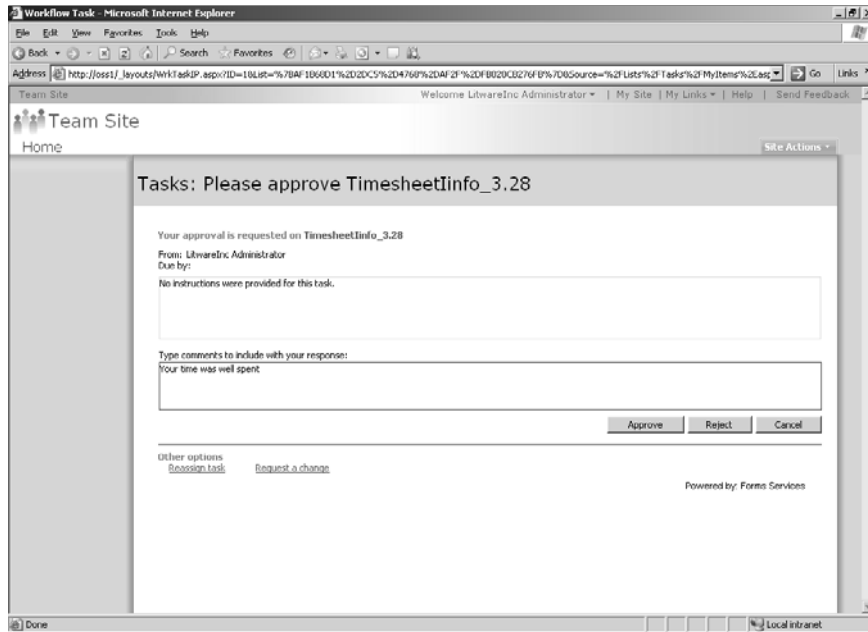


Figure 7-7 A manager uses this form to approve or reject the time sheet.

After approving the time sheet, the last step is to verify that the workflow was successfully completed and that the time sheet has been approved. We can see in Figure 7-8, which shows the Weekly Timesheets document library, that the TimeSheetApproval column for the document we approved is set to Completed.

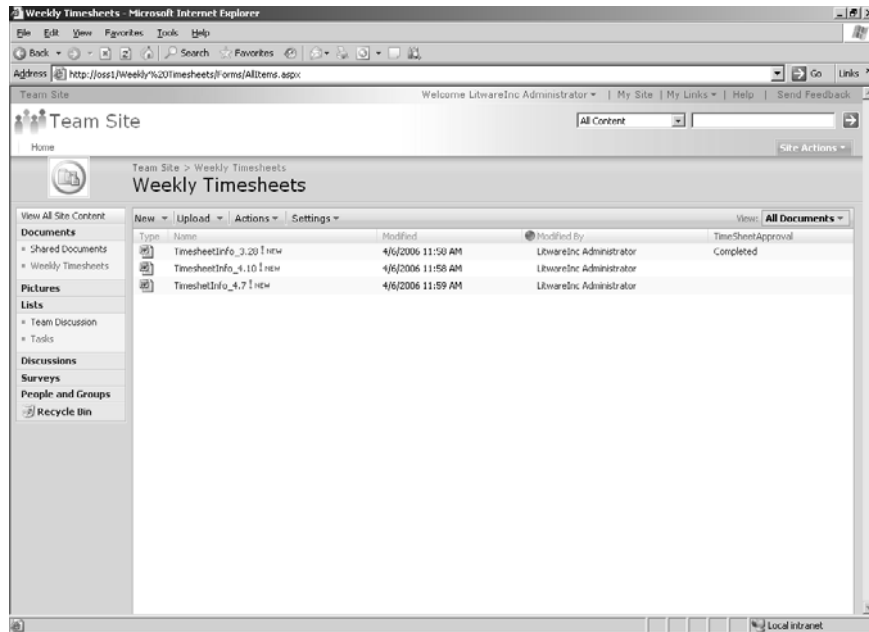


Figure 7-8 The status of a workflow is displayed for the item in a document library.

Creating a Human Workflow in Visual Studio 2005

Even though Windows SharePoint Services provides several workflow solutions, from time to time you'll need more functionality than is provided by the built-in routines. In these cases, developers can create custom workflow solutions as well as custom Office InfoPath 2007 forms for managing workflow initialization and task completion. In this section, you'll see how to create a custom approval workflow and interactive forms using Visual Studio 2005 and Office InfoPath 2007. We'll start by creating a project (named CustomApproval) in Visual Studio 2005, basing the project on the SharePoint Sequential Workflow Library template. Visual Studio fills out the project with initial references, an XML file, and a preliminary class named Workflow1.cs. Figure 7-9 shows the Visual Studio window with Workflow1.cs open in design view and the default onWorkflowActivated1 activity already in place.

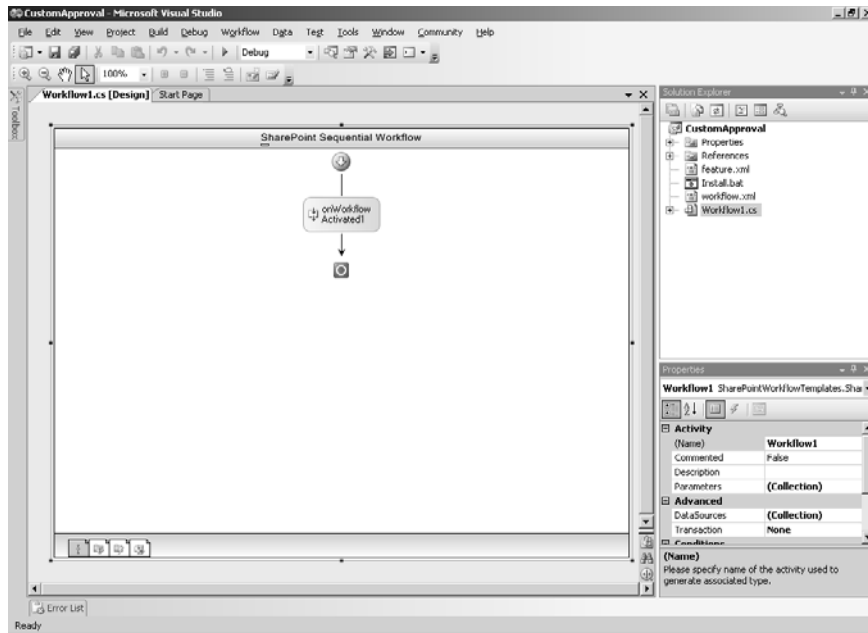


Figure 7-9 The initial project window for a SharePoint Sequential Workflow Library template project

The first step in implementing the workflow is to retrieve information such as the name of the manager who needs to approve the document and any comments associated with the document approval process. To gather this information we add three member fields to the `Workflow1.cs` class. (To view the code for `Workflow1.cs`, you can right-click `Workflow1.cs` and select `View Code`.) The member fields we add, all strings, are named `user`, `comments`, and `taskStatus`.

Next we switch `Workflow1.cs` back to design mode and create an event handler for this initial activity by right-clicking the representation of `onWorkflowActivated1` and selecting `Generate Handlers`. We implement the handler by accessing the `workflowProperties.CreationData` collection to access the `User` and `Comments` values and store them in the member fields. Here's the code for the initial fields and the handler:

```
public string user;
public string comments;
public string taskStatus;
private void onWorkflowActivated1_Invoked(object sender, EventArgs e)
{
    // retrieve the creation data
    user = workflowProperties.CreationData["User"].ToString();
    comments = workflowProperties.CreationData["Comments"].ToString();
}
```

Next we'll create a step in the workflow by using the CreateTask activity. We start by creating several member fields in the Workflow1.cs class to store information about the task's properties before and after editing. These fields are used to retrieve the results of the task's completion.

```
public Guid taskId = default(System.Guid);
public Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties taskProperties =
    new Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties();
public Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties
    taskBeforeProperties =
    new Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties();
public Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties
    taskAfterProperties =
    new Microsoft.SharePoint.Workflow.SPWorkflowTaskProperties();
public System.Workflow.Runtime.Messaging.CorrelationReference
    taskReferenceObject =
    new System.Workflow.Runtime.Messaging.CorrelationReference();
```

We switch Workflow1.cs to design mode again and drag a CreateTask activity onto the canvas after the activation activity, as shown in Figure 7-10.

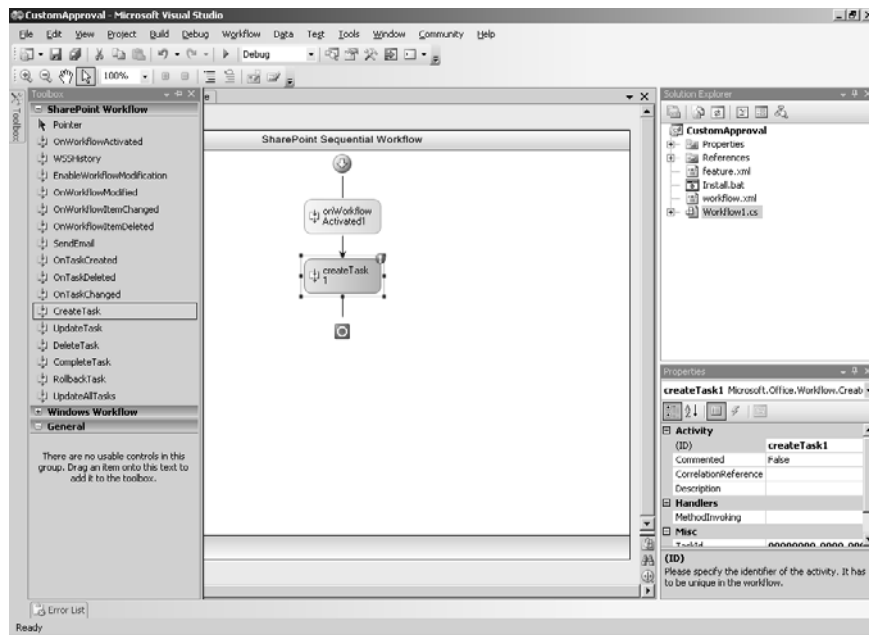


Figure 7-10 You can build the stages of a workflow by dragging activities from the toolbox.

Now we need to use the properties pane and set several properties for the CreateTask activity. We set its *CorrelationReference* property to `/Workflow1.taskReferenceObject`, its *TaskId* property to `/Workflow1.taskId`, and its *TaskProperties* property to `/Workflow1.taskProperties`. These values are used by the activity to manage the lifetime of the workflow and to provide parameters for the task's creation.

Finally we need to populate the properties of the task by using the *taskProperties* variable. To access this code, we right-click createTask1 and click Generate Handlers. The only property that is required is *taskID*, which was set in the designer. The *ExtendedProperties* property is used to send information to and from the Office InfoPath 2007 forms.

```
private void createTask1_MethodInvoking(object sender, EventArgs e)
{
// initialize the task ID
taskID = Guid.NewGuid();

// populate the properties of the task
taskProperties.AssignedTo = user;
taskProperties.Description = "Approve the document.";
taskProperties.Title = "Timesheet Approval";
// populate the type of the form and extended properties
taskProperties.TaskType = 0; // This is used to map tasks to InfoPath forms
taskProperties.ExtendedProperties["Comments"] = comments;
}
```

The previous step created a new task. The next step is to wait for the task to change and store the results of the change. We open Workflow1.cs in design mode and drag an OnTaskChanged activity to the workflow canvas, placing it after the createTask1 activity. We set its *CorrelationReference* property to /Workflow1.taskReferenceObject, its *TaskId* property to /Workflow1.taskId, its *BeforeProperties* property to /Workflow1.taskBeforeProperties, and its *AfterProperties* property to /Workflow1.taskAfterProperties.

Now we need to write the code associated with the *OnTaskChanged* handler that will retrieve the status value from *AfterProperties* and store it in the member variable *taskStatus*.

```
private void onTaskChanged1_Invoked(object sender, EventArgs e)
{
// retrieve the TaskStatus property from the infopath form
taskStatus = this.taskAfterProperties.ExtendedProperties["TaskStatus"].ToString();
}
```

The last step in defining the workflow is to complete the task and set its status to the *TaskStatus* value retrieved in the previous step. Again we open Workflow1.cs in design mode. This time we drag a CompleteTask activity onto the canvas and place it after the onTaskChanged1 activity. For this activity, we set the *CorrelationReference* property to /Workflow1.taskReferenceObject, the *TaskId* property to /Workflow1.taskId, and the *TaskOutcome* property to /Workflow1.taskStatus. The *TaskOutcome* property is the text value that is used when setting the status value of the completed task.

Now that the design and coding of the workflow is complete, we need to sign and build the workflow assembly. To sign the assembly, we navigate to the Signing tab in the project properties window, select the Sign The Assembly check box, and browse to the .snk file to use—CustomApproval.snk in this example.

Next we need to create the Office InfoPath 2007 forms that will be used by the workflow. The first form will be used to retrieve the user name of the manager who is assigned the approval task. The form has two fields, User and Comments, that are tied to appropriate data sources for receiving data. (For the purposes of the example, we used XML files.) The form also includes a text box and a button that is used to submit the form to Windows SharePoint Services and then close the form. Figure 7-11 shows the initialization form, named CustomApprovalAssoc.

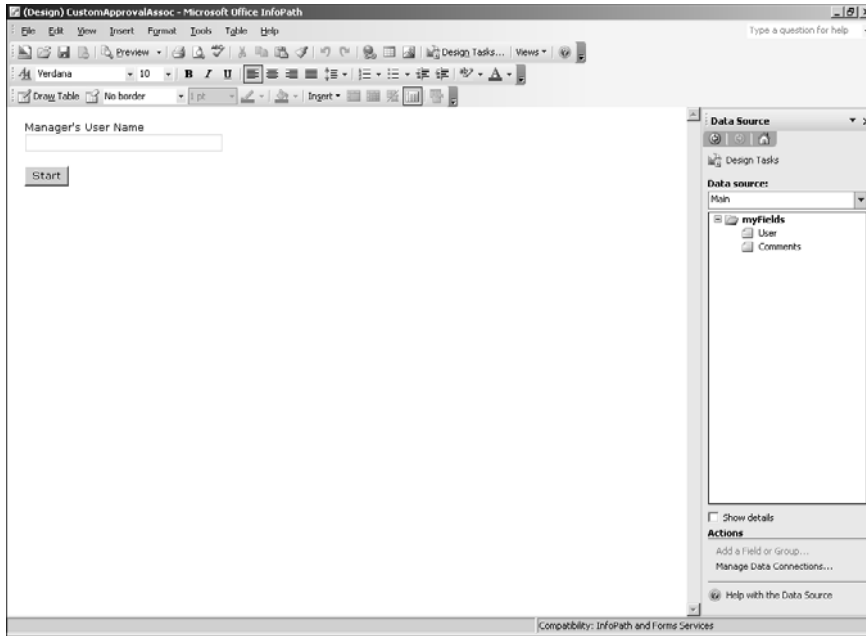


Figure 7-11 The first of the three Office InfoPath 2007 forms created for the custom workflow

Next we'll define the form that is displayed when a new instance of the workflow is started. We can base this form on the previous form so that the primary data source will have the same namespace in both forms. We modified this form by changing the text box label from Manager's User Name to Comments and changing the binding of the text box to the Comments data source field. This form is saved as CustomApprovalInit.

The last form we create is used when acting on the approval task. In this form, the manager is presented with any comments entered in the CustomApprovalInit form and two buttons—one to click to approve the document and another for cases when the document is rejected. The design of this form includes a field named *TaskStatus* and two XML data connections for receiving data. (The Comments text box at the top of the form is bound to a field named *ows_metaInfo_Comments* in one of the data sources.) The Accept and Reject buttons set the value of the *TaskStatus* field, submit the data to Windows SharePoint Services, and then close the form. Figure 7-12 shows the approval form.

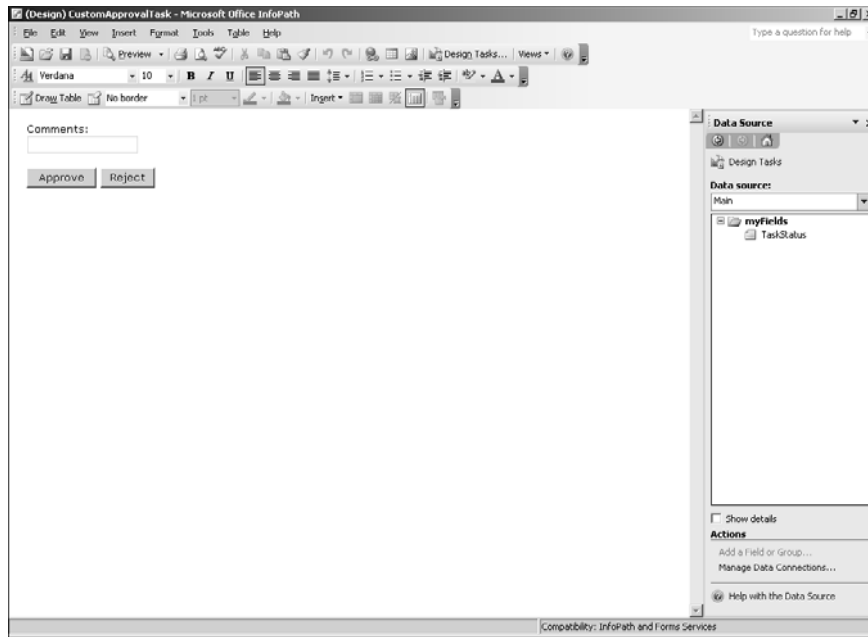


Figure 7-12 The task approval form

The next step is to define a feature that will install the workflow. The Visual Studio project provides a basic `Feature.xml` file that you can use for this purpose. The information you need to add to the file includes a GUID for the workflow, which you can generate by clicking the `Create Guid` command on the `Tools` menu in Visual Studio. For the `RegisterForms` value for this example, we use `Forms/*.xsn`. Here is the XML from the file:

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- _lcid="1033" _version="12.0.3111" _dal="1" -->
<!-- _LocalBinding -->

<!-- Customize the text in square brackets.
Remove brackets when filling in, e.g.
Id="[UNIQUE GUID]" ==> Id="328B9FAF-0E29-46b1-92B3-71F7D1D67761" -->

<Feature Id="FD130413-76CE-475e-9160-CB5C75116165"
  Title="Custom Timesheet Approval"
  Description="Custom Timesheet Approval workflow"
  Version="12.0.0.0"
  Scope="Site"
  ReceiverAssembly="Microsoft.Office.Workflow.Feature, Version=12.0.0.0,
    Culture=neutral, PublicKeyToken=71e9bce111e9429c"
  ReceiverClass="Microsoft.Office.Workflow.Feature.workflowFeatureReceiver"
  xmlns="http://schemas.microsoft.com/sharepoint/">
  <ElementManifests>
    <ElementManifest Location="workflow.xml" />
  </ElementManifests>
  <Properties>
    <Property Key="GloballyAvailable" Value="true" />
  </Properties>
</Feature>
```

```

    <!-- Value for RegisterForms key indicates the path to the forms relative to
         feature file location -->
    <!-- if you don't have forms, use *.xsn -->
    <Property Key="RegisterForms" Value="Forms/*.xsn" />
  </Properties>
</Feature>

```

Next we open the Workflow.xml file, which is also provided by Visual Studio, and fill in the first portion of its template. (You'll need to check the public key token of the *Microsoft.Office.Workflow.Feature* assembly deployed in the Global Assembly Cache (GAC) when working with the file yourself.) We populate the *MetaData* section with URNs for the three Office InfoPath 2007 forms we created earlier. To find the URN, open the form in Office InfoPath 2007 in design mode and choose Properties from the File menu.

```

<?xml version="1.0" encoding="utf-8" ?>
<!-- _lcid="1033" _version="12.0.3015" _dal="1" -->
<!-- _LocalBinding -->

<!-- Customize the text in square brackets.
Remove brackets when filling in, e.g.
Id="[UNIQUE GUID]" ==> Id="328B9FAF-0E29-46b1-92B3-71F7D1D67761" -->

<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <workflow
    Name="Custom Timesheet Approval"
    Description="Custom Timesheet Approval workflow"
    Id="E6D98FE0-E332-4471-AFDE-20881AFCAAD5"
    CodeBesideClass="CustomApproval.Workflow1"
    CodeBesideAssembly="CustomApproval, version=3.0.0.0, Culture=neutral,
      PublicKeyToken=6479e6da8b088665"
    AssociationUrl="_layouts/CstWrkflIP.aspx">

    <Categories/>
    <MetaData>
      <Association_FormURN>urn:schemas-microsoft-
        com:office:infopath:CustomApprovalAssoc:-myXSD-2006-01-02T01-28-
        35</Association_FormURN>
      <Instantiation_FormURN>urn:schemas-microsoft-
        com:office:infopath:CustomApprovalInit:-myXSD-2006-01-02T01-28-
        35</Instantiation_FormURN>
      <Task0_FormURN>urn:schemas-microsoft-
        com:office:infopath:CustomApprovalTask:-myXSD-2006-01-02T02-08-
        14</Task0_FormURN>
      <StatusPageUrl>_layouts/wrkStat.aspx</StatusPageUrl>
    </MetaData>
  </workflow>
</Elements>

```

The last step in creating the feature is to update the Install.bat file that Visual Studio 2005 provides when you create the workflow project. You can use this file to install and activate the feature. Here is the file as Visual Studio provides it.

```
@echo off
if %1.==.. (
goto usage
)

echo Copying the feature...
echo.
rd /s /q "%CommonProgramFiles%\Microsoft Shared\web server extensions\12\TEMPLATE\FEATURES\
[FEATURE NAME]"
mkdir "%CommonProgramFiles%\Microsoft Shared\web server extensions\12\TEMPLATE\FEATURES\
[FEATURE NAME]"

copy /Y [FEATURE XML FILE NAME] "%CommonProgramFiles%\Microsoft Shared\web
server extensions\12\TEMPLATE\FEATURES\[FEATURE NAME]"
copy /Y [WORKFLOW XML FILE NAME] "%CommonProgramFiles%\Microsoft Shared\web
server extensions\12\TEMPLATE\FEATURES\[FEATURE NAME]"
xcopy /s /Y [FORMS, e.g. *.xsn or Forms\*.xsn] "%programfiles%\Common
Files\Microsoft Shared\web server extensions\12\TEMPLATE\FEATURES\[FEATURE
NAME]\[FORMS DIRECTORY if any; should match the location specified in your
feature.xml under RegisterForms]"

echo.
echo Adding assemblies to the GAC...
echo.
"%programfiles%\Microsoft Visual Studio 8\SDK\v2.0\Bin\gacutil.exe" -uf [DLL
NAME (NO EXTENSION, e.g. HelloWorldFlow)]
"%programfiles%\Microsoft Visual Studio 8\SDK\v2.0\Bin\gacutil.exe" -if [PATH
TO DLL RELATIVE TO THIS FILE, e.g. bin\Debug\HelloWorldFlow.dll]

echo.
echo Activating the feature...
echo.
pushd %programfiles%\common files\microsoft shared\web server extensions\12\bin
stsadm -o deactivatefeature -filename [FEATURE NAME\FEATURE XML FILE NAME,
e.g. HelloWorldFlow\feature.xml] -url %1
stsadm -o uninstallfeature -filename [FEATURE NAME\FEATURE XML FILE NAME, e.g.
HelloWorldFlow\feature.xml]

stsadm -o installfeature -filename [FEATURE NAME\FEATURE XML FILE NAME, e.g.
HelloWorldFlow\feature.xml]
stsadm -o activatefeature -filename [FEATURE NAME\FEATURE XML FILE NAME, e.g.
HelloWorldFlow\feature.xml] -url %1

echo Doing an iisreset...
echo.
popd
iisreset

goto end
```

```
:usage  
echo.  
echo Usage: Install ^<url of Server^>  
echo i.e. Install http://myserver  
echo.  
:end
```

You execute Install.bat with the command line parameter with the URL to your site, which will register the assembly in the GAC, uninstall, reinstall, and finally activate your workflow feature. You could now follow the steps shown earlier in the chapter to associate the custom workflow with the Weekly Timesheet document library, configure the workflow so that the LitwareInc Administrator must approve the document, and then run and test the workflow.



Note You can implement conditional branching in a workflow. For example, if a time sheet is approved, the manager needs to be sure the consultant is paid. You could extend the approval workflow by using the Windows Workflow If-Else activity to create a new task for the manager only if *TaskStatus* is set to Accepted.

Introducing Excel Services

In this chapter:

Key Scenarios for Excel Services	144
Excel Services Architecture Overview	148
Controlling Visible Information and Interacting with Workbooks	149
Building Applications with Excel Web Services	151
Controlling and Protecting Workbooks	154
Data Connection Libraries	156
Unsupported Features in Excel Services	159
Excel Services and Reporting in a Portal	162
Coding with Excel Web Services	172
Excel Services User-Defined Functions	178

Microsoft Office SharePoint Server 2007 Excel Services, new server technology that ships with the 2007 Microsoft Office system, supports loading, calculating, and rendering Excel workbooks on servers. Excel Services has two primary interfaces: a Web-based user interface that enables users to view workbooks in a browser, and a Web services interface for programmatic access to the data and logic in workbooks. Excel Services, which requires Microsoft Office SharePoint Server 2007, provides the following capabilities:

- Allows browser-based access to workbooks
- Incorporates workbooks in portals and on dashboards
- Controls access to workbooks for either regulatory and audit concerns or to protect intellectual property in the workbooks
- Eliminates “multiple versions of the truth”; in other words, Excel Services makes it unnecessary for users to maintain separate copies of the same workbook, which are often out of sync
- Uses servers to offload long-running calculations from desktop computers
- Enables logic and business models built in Excel to be used in other applications without having to develop the logic or business models again in a programming language

Excel Services is built on Office SharePoint Server 2007, which means, for example, that when a user saves a workbook to Excel Services, he or she saves it to a SharePoint document library and can then make use of SharePoint document library features. A whole range of additional

features in SharePoint and Excel Services together provide tools to help organizations and users share, manage, and control workbooks. These features include workflows, versioning, auditing, and taking snapshots of workbooks at specific points in time.

This chapter introduces some of the key concepts and practices of Excel Services. We'll look at the scenarios for which Excel Services can be used, its architecture, mechanisms for controlling who can access a workbook, and how users can interact with workbooks in the browser. We'll also review methods and other elements of the Web services interface. Then we'll demonstrate how to work with Excel Services and a SharePoint site, how to incorporate Excel formulas in an application, and how to create a user-defined function (UDF) that you can use with Excel Services.

Key Scenarios for Excel Services

Microsoft targeted three scenarios for the initial release of Excel Services:

- Sharing workbooks through the browser
- Building business intelligence (BI) dashboards
- Reusing the logic encapsulated in Excel workbooks in custom applications

In the following sections, we'll look at each of these scenarios in more detail, starting with sharing workbooks and issues related to controlling and managing workbooks.

Sharing Workbooks Through a Browser

Certain types of workbooks—those that contain important business information or logic, require intensive calculation, or are shared with customers or partners—don't easily lend themselves to current methods of sharing workbooks, such as e-mail or file shares. For example, sharing through e-mail tends to multiply the number of copies of a workbook. It's easy for users who receive the file to modify their personal copies, which results in more than one copy of the data. Which version is accurate?

It is also difficult for workbook authors to protect parts of their work—for example, key logic, business modeling, or data that is considered confidential or proprietary intellectual property—while sharing other parts of the workbook, such as the completed analysis they want a client to see. Finally, large workbooks that require intensive calculation are not well suited for distribution by e-mail. Every user who opens the file has to wait for the workbook to download and the calculations to run before they can continue with their work.

Excel Services provides users with means to handle these and other issues related to sharing workbooks. Using Excel 2007, workbook authors can save their files to a SharePoint document library and give other users browser-based access to the server-calculated version of that workbook. When a user accesses the workbook, Excel Services loads the workbook, refreshes external data when necessary, runs necessary calculations, and sends the resulting output to

the browser. The workbook is rendered using only DHTML. No ActiveX control is required, and no client installation of Office Excel 2007 is necessary. Users view the latest version of the workbook, and they can interact with its data in the browser. Moreover, a workbook author can restrict access to specific sheets or cell ranges in the workbook, thus hiding intellectual property from people who need to see only the results. A user accessing the file in the browser sees only cell values and not the underlying formulas. If multiple users are viewing the same workbook, it is loaded and calculated only once by the server, saving the network bandwidth and lessening the wait time for long-running calculations. Finally, workbook access can be logged and audited. Overall, these capabilities provide a powerful means for sharing workbook via the browser in a controlled and secure way.



Note Excel Services does load balance separate workbook calculations across multiple servers and calculates more than one workbook on a server (each request runs on a different thread). Excel Services does not, however, break out a single workbook across multiple servers. This design is optimized for scaling to large numbers of workbooks and requests; for example, many users viewing a dashboard with data from a number of workbooks on it, or programmatically running a large parametric sweep of a workbook or group of workbooks.

The upcoming Microsoft Windows Compute Cluster Server 2003 complements Excel Services' interactive end-user scenarios by providing scalable, reliable, batch workbook processing. Compute Cluster enables computation-intensive applications to scale their performance and achieve high availability using farms of 64-bit servers. Compute Cluster capabilities can be easily integrated into an application using a .NET or DCOM object model. For example, a Web Part can be programmatically connected to a Compute Cluster job scheduler that distributes a processing load over a number of compute nodes, each running a separate instance of the Excel Services engine. For more information about Microsoft Windows Compute Cluster Server 2003, see www.microsoft.com/windowsserver2003/ccs/default.mspx.

Building Business Intelligence Dashboards

Business intelligence is one of the key areas of focus for Office Excel 2007 and Excel Services. The browser-based sharing of workbooks described in the previous section is part of that effort. Business people such as financial analysts, business planners, and engineers who work with data in Excel workbooks can't easily reuse and share that data in a portal or on a dashboard. The integration of Excel Services with Windows SharePoint Services and Office SharePoint Server 2007, however, allows all sorts of users to build Web solutions that contain live and interactive data without using any code, as shown in Figure 8-1 on the next page.

Let's look at the pieces that could be included on a dashboard and how it would be created.

- Office SharePoint Server 2007** The dashboard is built using Windows SharePoint Services, and its format is consistent with the rest of the SharePoint site it belongs to. Because the concept of a dashboard (a page containing a set of SharePoint Web Parts) is native to the SharePoint platform, it is simple for users to create dashboards that look like the rest of their intranet site.

- **Excel Services** Two of the Web Parts on the dashboard are provided by Excel Services; in this example, a table and chart are contained within a workbook, but these could also be formulas, PivotTables, and so on. In the context of a dashboard, however, users can configure Excel Services to show only a portion of the workbook (a “summary” view). The author can also configure the level of interactivity available for a dashboard. Finally, because Excel Services is just showing a workbook, the creator of the dashboard can make use of his or her knowledge of Excel to create that workbook and format it without writing any custom code to access external data, perform calculations, format the results, or provide interactivity.
- **Key Performance Indicators** One of the Web Parts on the dashboard is a Key Performance Indicator (KPI) List Web Part. This Web Part is part of SharePoint, not Excel Services. A KPI Web Part provides a highly visual display of data that allows users to get a sense of the status of important factors in their work. The data can come from sources such as a manually maintained Excel worksheet or an enterprise database.
- **SharePoint Filters** The final set of Web Parts on this dashboard are Filter parts, which are also SharePoint Web Parts, not from Excel Services. These Web Parts can be hooked up without code to other parts on the page, enabling users to filter everything they see on the page. For example, if a user named Beth wanted to see only her support calls for ProductA during FY04, she would enter those criteria and click Apply Filters. All the content on the dashboard (chart, table, KPIs) would update accordingly. Filters can be set up to allow users to pick from a predetermined set of values; enter a date, number, or string; filter on the user viewing the dashboard; filter on values in an external database; and the like.



Figure 8-1 A business intelligence dashboard built without code using Excel Services

How is this dashboard created? A user with the appropriate SharePoint permissions can create a dashboard with just a few clicks. They can choose which Web Parts (there are many more than the few just mentioned) they want to display on the dashboard, as well as where they want those parts displayed. After choosing the Web Parts to include on the dashboard, the user can configure the dashboard using a Web browser, setting up which workbook data is shown (and how), what KPIs are shown in the KPI list, and what filters should be available, for example.

Reusing the Logic Encapsulated in Excel Workbooks

In addition to a browser-based interface to the server, Excel Services provides a Web service interface. The same workbook that is published by its author can be accessed programmatically by any application that can communicate through Web services. The calling application can change values, calculate formulas in the workbook, and retrieve some or all of the updated workbook using the Web service interface (subject, of course, to security permissions). This capability opens a range of interesting possibilities and solutions for using Excel and Excel Services.

Today, developers or other users of spreadsheets need to rewrite the logic represented in them in code to support running in a server-grade environment. Excel Services lets the logic be retained and updated when necessary in a workbook and used to its full effect in the context of an application. This combination of effort provides users with the best of both worlds—easier and more effective modeling in Excel, as compared to custom code, together with application integration, scalability, and manageability.

Multi-User Editing of Workbooks

Excel Services does not attempt to address the scenario of multiple users who want to jointly author a workbook in real time and see each other's edits. Excel Services allows more than one user to open and interact with a workbook at the same time, but each user has his or her own session. If a user interacts with the workbook (for example, by setting a filter on a PivotTable), other users do not see those changes. The changes a user makes are not saved to the original file.

The server storing the workbook opens the file as read-only, and each user has his or her own session state in the server's memory. A workbook is loaded only once in server memory no matter how many users are accessing it. Each user's session maintains the specific interactions (for example, filters) that the user has performed and how the workbook results should be calculated and returned to that user. This behavior is the same whether calling the workbook through Web services or accessing it through the Web-based user interface.

Excel Services Architecture Overview

As mentioned previously, Excel Services is part of Office SharePoint Server 2007. Excel Services has three components: Excel Web Access, Excel Web Services, and Excel Calculation Services.

- Excel Web Access is a SharePoint Web Part that performs the rendering (in this case, creating the HTML) of Excel workbooks on a Web page. You can use this Web Part as you would any other Web Part in SharePoint to create a wide range of pages.
- Excel Web Services provides programmatic access to the logic and data in a workbook. This component is a Web service hosted in SharePoint. You can use methods in this Web service to develop applications that incorporate calculations performed by Excel Services and to automate the updating of Excel workbooks.
- Excel Calculation Services is the component that loads the workbooks, calculates them, refreshes external data, and maintains session state for interactivity. This component is at the heart of Excel Services.

Additionally, Excel Services makes use of a proxy internally to handle the communication between the components on the Web front end and the application server in multiple-server configurations. This proxy also handles load balancing in case there are multiple application servers in your installation.

In the simplest configuration, the three components that make up Excel Services would run on the same computer; however, in most production environments with a significant number of users, the components on the Web front end and the application server would be on different computers.

Security

Excel Services uses the security infrastructure provided by Windows SharePoint Services. Excel Services uses Windows SharePoint Services for authentication (who can log onto the server) as well as authorization (who has access to which workbook and the type of access: read, write, view only, and so on). The reliance on the Windows SharePoint Services security infrastructure provides a strong security environment for protecting workbooks.

Performance and Scalability

Excel Services is optimized for scenarios in which multiple users access the same worksheets. This optimization is achieved by caching at multiple levels so that collective performance for a group of users is improved by caching the workbooks as well as any external data queried by the workbooks. These operations are transparent to an end user except for the response time. (Cached results are shared only between users who have the same rights.)

Controlling Visible Information and Interacting with Workbooks

When a workbook is published to Excel Services, the entire workbook is saved to the server so that data can be refreshed or calculations run. The workbook author can, however, control which parts of the workbook are visible when a user views the file in the browser or it is accessed through the Excel Services Web services API. The three choices for controlling how a workbook is viewed on the server are

- The entire workbook (the default setting).
- A subset of sheets (as many or as few as you want). This approach is useful when you have workbooks that contain lots of behind-the-scenes sheets that hold intermediate calculations, source data, and so on, but only a few sheets that you want users to see. If an author needs to make changes to the workbook over time, however, he or she still can use all the features in Office Excel 2007 and can see the entire workbook without having to unhide sheets.
- A set of named items, such as named ranges, charts, tables, PivotTables, and PivotCharts. This mode is called Named Object view. In this view, users can select the items they want to see from a drop-down list in the browser. Items are displayed one at a time, which simplifies setting up a dashboard. Even though only a single item is shown at a time, the entire workbook is loaded in Excel Services, which means that the visible objects can be refreshed and are interactive.

Defining Parameters

Users are able to interact with a workbook in the browser, but they will not be able to directly edit cells in the grid. Instead, workbook authors can specify parameters as part of the process of publishing a workbook to Excel Services. Parameters let an author expose specific cells whose values a user can then change when working with the data. Excel Services provides a built-in task pane for changing the values of parameters. After the value of a parameter has been changed, the workbook calculates new values and the user sees the results in the browser.

Note that not all cells can be exposed as parameters. Here are the restrictions:

- The parameter must be a single cell and cannot be a range.
- The cell must not contain a formula.
- The cell must be a regular workbook cell, not a cell in a PivotTable, table, chart, and so on.
- The cell must have a defined name.

Interacting with Workbooks in the Browser

Users can interact with workbooks in the browser in a number of ways:

- View workbooks in a browser
- Navigate within workbooks
- Sort, filter, and perform other operations on the data in workbooks
- Change parameters to facilitate what-if analysis in the workbooks

Features related to interactivity are generally grouped into three areas: worksheets, tables, and PivotTables. Although the level of interactivity does not include full authoring of workbooks in the browser, users with appropriate permission can open a workbook in Office Excel 2007 and have the complete range of Excel's features at their behest. In situations in which interactivity needs to be controlled—for example, if you want to display some data using a PivotTable but don't want users to be able to change the view—interactivity can be turned off. You can, for instance, allow users to sort but not filter, to interact with tables but not with PivotTables, and so on.

Interactivity with Worksheets

Excel Services supports the same layout and formatting capabilities as does Office Excel 2007. To users, a worksheet viewed in Office Excel 2007 will look the same as the worksheet viewed in a browser (within the constraints of HTML). The look of a worksheet includes familiar formatting as well as features new to Office Excel 2007, such as data bars, color scales, table styles, and others.

Only one section of a worksheet is served up by Excel Services at one time. This limitation is a performance optimization, as less HTML needs to be served up to the client. The number of rows and columns can be configured. The default values are 75 rows and 20 columns. Paging controls allow a user to move between sections.

Users will also be able to page between worksheets (using tabs as in Excel), expand and collapse outlining, set parameters, refresh external data and calculate the worksheet, and find values within the worksheet. (The Find feature will search for values in the entire worksheet, even the parts that are outside the current section.)

Interactivity with Tables and AutoFilters

If a workbook contains a table or autofilters, users will be able to set and update sorts and filters. Specifically, users will be able to sort data in ascending or descending order, use a multi-select or Top 10 filter, set quick filters ("Above Average," "Below Average," "Contains," "Last Month," and so forth), or set custom filters ("Less Than," "Contains," and so on).

Interactivity with PivotTables

If a workbook contains a PivotTable, users can interact with the PivotTable within the browser, including expanding and collapsing levels, sorting in ascending or descending order, applying a multiselect or Top 10 filter, setting quick filters, or setting custom filters.

Building Applications with Excel Web Services

One of the main purposes of Excel Services is to enable the logic encapsulated in Excel workbooks to be used in custom applications. Having this capability means having access to workbooks and their contents on the server through Web services in a manner that's scalable and manageable.

As mentioned earlier in the section about Excel Services architecture, Excel Calculation Services is the engine of Excel Services; it is the component that loads and calculates workbooks. In situations in which users interact with a workbook in the browser, Excel Calculation Services loads and calculates a workbook and then hands it off to the Excel Web Access component, which produces the HTML that is rendered in the browser. Developers can also use Excel Calculation Services without interacting with the Excel Web Access component. Specifically, a Web service API sits on top of Excel Calculation Services so that developers can call server-side workbooks directly from their own applications. For example, developers can write code that opens a workbook on a server, sets cells and ranges to specific values, controls an external data refresh and workbook calculation, and then retrieves values from the calculated workbook or retrieves the workbook in its entirety.

Excel Web Services enables many possibilities. In one scenario, a business expert who authors a workbook can now maintain the model in Excel; an administrator can protect that model on the server using the appropriate set of users, roles, permissions, and a firewall; and a developer can call Excel Web Services to incorporate the logic of the model into the operations of a custom solution. A variation on this would be to provide a custom user interface to Excel-based server applications that use Excel Web Services to interact with a server-side calculation session.

Another scenario in which Excel Web Services will play a role is automating workbook updates on servers. This capability works especially well in combination with the new Open XML file format, which greatly simplifies the task of programmatically creating an Excel file from scratch or by using a template. After a file has been created, it often needs to be calculated; for example, external data feeds need to be updated. It is straightforward for developers to write code that uses Excel Web Services to perform these operations and then retrieves an up-to-date copy of the calculated file and saves it back to the server or delivers it to any other destination.

In brief, you can write code that can start a session with Excel Calculation Services, set values in cells and ranges, process the workbook, and get calculated values or the entire workbook

back to use within your application. The following list describes some methods and properties of Excel Web Services. You'll see some sample code later in the chapter.

- *GetApiVersion* gets a version string of the installed Web service API build.
- *sessionId = OpenWorkbook* opens a server-side calculation session. The method takes a workbook file path and a few other arguments and returns a *sessionId*.
- *GetSessionInformation* gets a few properties of the server session, primarily the language context of the session.
- *SetCell* sets a value in a cell on one of the workbook's sheets. Two versions of this method exist: one takes a cell address (for example, B52) or a named range (for example, Interest), and the other accepts integer coordinates for cases in which using the coordinates in your code is more convenient, usually when you have indexes in the code and want to use them to index the sheet.
- *SetRange* is the same as *SetCell* but is used for setting values for an entire contiguous range. The same two versions exist as in *SetCell*.
- *Refresh* reads data from an external data connection (or all of a workbook's connections) and refreshes the values in the relevant cells; for example, in PivotTable cells or in the results of cube formulas.
- *Calculate* recalculates the formulas in a specific range or in the entire workbook. This method is useful when the workbook author has turned off automatic calculation. Two versions exist: you can use either a string or integer coordinates to refer to a range, much like in the *Set* methods.
- *CalculateWorkbook* calculates the entire workbook, using one of two calculation methods:
 - *Recalculate* calculates only formulas that have dependencies that changed (aka "dirty" formulas).
 - *CalculateFull* calculates all formulas, regardless of dependency changes.
- *GetCell* gets a value from a cell. The two regular addressing mechanisms exist here as well. You can get either formatted string values or the raw binary values.
- *GetRange* gets a set of values from a contiguous range of cells; it has the same addressing flavors as *GetCell*.
- *GetWorkbook* gets the entire calculated workbook into your application memory as a byte array. You can get either the live result or a snapshot. A snapshot is essentially a workbook with the layout of the original workbook, with all the original formatting and with up-to-date values, but with the formulas and external connections stripped out and without the portions of the workbook that were marked during the publishing process not to be viewed.

- *CancelRequest*: If your application runs the Excel Web Services session in a separate thread and wants to cancel a long-running server request (like a long calculation that a user got tired of waiting for), it can do so by calling this method.
- *CloseWorkbook* tells the server to close the workbook that it opened for this session, thereby also allowing the server to release all the resources it maintained for the context of your session.

Error Handling

Errors are exposed to an Excel Web Services application in three ways:

- Excel calculation errors show up just like they do in Excel—as cell error values (#VALUE!). When you call *GetCell* or *GetRange* and ask for formatted values, you'll get the #-style error string. When you ask for unformatted values, you'll get an enumerated error code.
- An error in processing one of the Web service methods (which prevents the method from completing successfully) is exposed as a SOAP exception that your code can catch.
- Less critical errors that do not prevent the method from returning normal results are returned as part of the method's arguments, specifically as an output argument. The reason for this approach is that an exception would divert the code from its normal execution path, which is not desirable with noncritical errors. Checking for these errors is optional.

Sessions

One concept that developers need to be aware of is the way Excel Calculation Services maintains sessions for performance reasons. A good way to understand the benefit of server state is to think about a user who interacts with an Excel workbook in a Web browser. Each time the user takes an interactive step—such as revealing details in a PivotTable, changing an input parameter, refreshing data connections, and so on—we want the server to compute only the difference between what the user saw on the screen before taking that step and what they should see as a result of the step. For performance reasons, we do not want the server to read the workbook file from disk again or to recalculate formulas that do not need to be recalculated.

This behavior is also desirable in situations in which an Excel server-side calculation is performed by the server for a custom solution that uses Excel Web Services. For example, if the solution's code sets a cell to a new value, we want the server to calculate only formulas that depend on that cell's value and nothing else. So, the server keeps the state, or context, of a custom application's calculation in server memory. This context is called a *session*. A session ID lets the application tell the server which session it is working with. The server returns this ID to the application when it starts a new session (by opening a workbook), and then the application code passes this session ID to subsequent Web service calls.

Controlling and Protecting Workbooks

In this section, we'll review how Excel Services enables you to lock down and protect workbooks. These capabilities help address an organization's need to limit access to workbooks either for regulatory and audit concerns or to protect proprietary information. To further address this requirement, in addition to allowing users to execute and view workbooks on the server, the Windows SharePoint Services architecture has been extended with a new right, called the View Item right.

After describing the View Item right, we'll show how to ensure that the workbooks users view are the "right" workbooks—in other words, how to control which users can author workbooks that will be run on the server, and which versions of those workbooks will be available for users to view.

The View Item Right

To help you understand the View Item right, think of how SharePoint is used for a document store on the server. Users can save and version files, administrators can control access permissions, and so on, all through any browser. (SharePoint does a lot more than serve as a document store, of course.)

Currently, SharePoint administrators can give users Reader rights, which lets them look at content; Contributor rights, which provides permission to look at, change, and add to content; or Administrator rights, which provides full control. With the View Item right, you can lock down workbooks that have been published to SharePoint so that users can open the workbooks using Excel Services, interact with them, and see the execution results, but can't download a copy of the workbook or access any areas that were not published as viewable on the server. This right helps hide any proprietary information contained within the workbook, such as specific formulas, a proprietary model, external data connections, or hidden elements of the workbook.



Note The View Item right is specific to SharePoint document libraries and does not work with workbooks that are stored in UNC shares or generic HTTP locations.

As an example of how the View Item right can be used in an Excel Services solution, imagine a workbook that takes several inputs and then calculates discount rates for a large retailer. The discount rate for any specific distributor is dependant on many factors, including the quantity of product that is purchased, the time of year, and the number of previous transactions for a given distributor. Of course, the formula for the discount rate is carefully guarded by the retailer because it determines the profit made on each transaction. With Excel Services, this retailer can now assign a distributor View Item rights to the workbook containing this sensitive model without having to worry that the distributor will be able to download or see the model.

The View Item right affects how both Excel Web Access and Excel Web Services allow access to a workbook. The specific elements that are affected include the following:

- Which portions of a workbook a user can access. When a user has only the View Item right, he or she can see only the portions of the workbook that have been marked as viewable on the server during the publishing process.
- Which portions of the workbook can be opened in Excel. Although users with the Reader right can open the original workbook in Excel if they want to see the model, formulas, data connections, and so on, users with the View Item right can only open a snapshot of the original workbook in Excel. In a snapshot, a user can see the numbers but not the proprietary information behind those numbers (formulas, connections, and so on). That information, in fact, is not contained in the snapshot. (And, of course, users can see the numbers only for the portions of the workbook that were marked as visible on the server.)

These examples focus on viewing the workbook through the browser using Excel Web Access. Similarly, if an application accesses the workbook through Excel Web Services, the View Item right is enforced. For example, issuing a call such as *GetRangeA1* to a range that has not been marked as viewable will result in an exception, as would *GetWorkbook*.

Controlling Who Can Publish Workbooks to Excel Services

The first step in controlling who can save workbooks to the server is controlling the locations from which the server will load workbooks. An administrator controls these locations by maintaining a list of directory paths as trusted locations. Excel Services checks this list before opening any workbook and will not load and execute a workbook unless it comes from a trusted location. Using SharePoint rights (for workbooks stored in SharePoint document libraries) or simple file system rights (for arbitrary UNC paths), an administrator can control who can save workbooks to these locations. Effectively, this list of trusted locations allows an administrator to control which users have access rights to save workbooks that will be executed by Excel Services.

For example, on a company intranet all employees could have the rights to save workbooks (and other files) to various SharePoint sites within a portal. However, an administrator could designate one trusted location within the portal where only a select few users could save workbooks that would be loaded and executed by Excel Services. In turn, the users browsing those workbooks are guaranteed that they are viewing sanctioned copies of the workbooks.

Controlling the Publishing Process for Workbooks in Excel Services

When workbooks are stored in SharePoint document libraries, many more features for controlling the process of authoring and publishing workbooks are provided, and these features guarantee not only that the right version of a workbook is made available to users but that each workbook is subject to proper review and approval cycles. Additionally, an audit log is

available that tracks who accessed which workbook and when. This log is useful in the context of compliance processes, for example.

The check-in, check-out, and versioning mechanisms in Windows SharePoint Services 3.0 allow for major and minor version numbering as well as security specifically for old versions of workbooks (and other documents). Additionally, SharePoint has built-in functionality for policies around retention and expiration of documents, which means that outdated versions of a workbook are automatically retained and then destroyed in order to meet compliance requirements.

Document approval within SharePoint allows an administrator to set up a document library so that when a workbook author saves a new version of a workbook to the library, the workbook is not available to other users to view immediately. Instead, the workbook needs to be reviewed by an administrator or an appointee, such as a financial analyst in charge of the library, and be approved or rejected. Only after approval does the workbook become available for other users who have the rights to view it. An approval process can be as simple as the administrator monitoring the library and changing a flag on the workbook in the document library, or it can be a custom workflow that sends e-mail messages to a group of approvers to ensure that the workbook meets any number of internal requirements prior to its approval.

More Info For more information about workflows, see Chapter 7, "Creating Workflows: The Missing Piece of Office Productivity."

Finally, Windows SharePoint Services 3.0 allows administrators to audit key events within document libraries. Although auditing within workbooks themselves has not been implemented, events such as opening, creating, modifying, and deleting workbooks are all recorded in a centralized log. Several built-in reports are provided to analyze that log, in addition to mechanisms to generate custom Excel reports.

Data Connection Libraries

In the past few sections, you've learned about managing, sharing, and securing Excel workbooks using Windows SharePoint Services and Excel Services. In this section, we'll cover a new feature that provides management, sharing, and security of data connections: the Data Connection Library.

What Is a Data Connection Library?

A Data Connection Library (DCL) is a new type of SharePoint library (much like a document library) that provides a location in which to store, share, and manage connection files. Connection files are Office Data Connection (.odc) files that contain the information and parameters needed to form a data connection, such as the server name, the OLAP cube or table name,

and a query. Because a DCL is a library in SharePoint, it provides SharePoint features such as workflow support, file approval, library level/item level security, and sorting and filtering based on metadata. You create a DCL the same way you create any library, and DCLs can be created on a portal or on a team site.

The way Excel interacts with a DCL makes the DCL more valuable than just a document library full of connection files. In the following sections, we'll describe a few of the problems that a DCL and Office Excel 2007 help solve.

Connecting to Databases Made Easy

Setting up a connection to a database in Excel is a task that many users struggle with. For example, if they want to connect to an ODBC data source or to a SQL Server Analysis Services cube, users must know details such as server names, cube names, table names, what type of connection to create, user credentials, and so on. Lots of steps and knowledge are required. Office Excel 2007 and DCLs make connecting to databases a much simpler activity. Users need to know what data they want to work with, and that's pretty much it. Here's an example involving a PivotTable.

To connect to a database (or to another external data source such as a Web query), a user clicks the Data tab. Figure 8-2 on the next page shows the Data tab (in the beta 1 build of Office 2007; this is not the final user interface). The user clicks the Existing Connections button, and the Existing Connections dialog box will list the connections that are stored in a DCL. These connections have friendly names and nontechnical descriptions, so it is easy for users to choose the connection they need. The names and descriptions are provided by the person or persons who set up the DCL and populate it with .odc files. Users simply highlight the connection they want to use. At this point, they'll see another dialog box that allows them to indicate what Office Excel 2007 should do with the data. Based on the connection the user selected, Office Excel 2007 offers only options that are possible for a particular database. For example, the data source in this example is a SQL Server Analysis Server database, which cannot be represented in Office Excel 2007 as a table, so that option is unavailable. At this point, all that is left is to click OK, and the user has connected a PivotTable to a data source. The whole operation is a total of three or four steps.

But how do the connections end up in the DCL in the first place? In general, either connection-savvy power users or members of the IT staff will create data connection files and put them in the DCLs where the connections can be used throughout an organization.

Excel knows about the existence of DCLs through a SharePoint feature that allows an administrator to "advertise" the location of the DCL to Office 2007 clients, which in turn allows connections from a DCL to show up in Office Excel 2007. Of course, the DCL shows up only if the user has permissions to access the connection files the library contains.

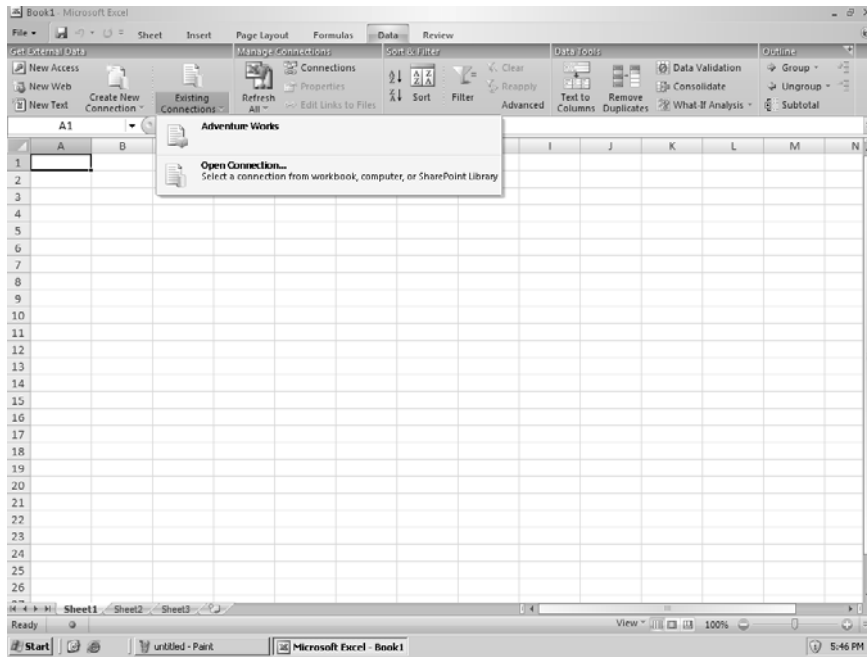


Figure 8-2 The Data tab

Solving Connection Management Problems

In addition to improving the discoverability and ease of using connections, DCLs help users manage connections. For example, information about data sources, such as the server name, OLAP cube name, table name, and the like, can change. A typical case is a database that is moved from a test server to a production server. For organizations that have many users creating workbooks, it can be difficult to communicate changes such as these to all the people who are affected. A worse situation is one in which hundreds of existing workbooks need to have their data connections updated.

A DCL helps solve problems such as these because only a single connection file in the DCL needs to be updated with the new information. After the connection file is updated, workbook authors will have the correct connection information the next time they use that connection file, and any existing workbooks that were created using the connection file will have their connections updated automatically the next time that workbook's data is refreshed.

How exactly does this work? By default, workbooks will refresh their connection information from a DCL only when they fail to connect to the data source. (You might think of this as a failover mechanism.) But you can also force workbooks to always get the latest connection information before the connection is attempted. You might make use of this capability when you want workbook authors to start using a new database for business reports but you want to keep the old database up and running for auditing or test reasons. Connections to the old database still work, but you want current and future workbooks to start using the new

database. The “always use this file to refresh data” setting is a property in the .odc file itself and can be set when the file is created.

Making Data Connectivity More Secure

A DCL can also be used to make connecting to data more secure. One common security concern is knowing which data connections are safe to run. For example, data connections can contain malicious queries, or they can contain connection parameters that can slow down an application or compromise the integrity of the data. By creating a DCL, and by allowing only the most knowledgeable and trusted users to save connections to the DCL, you add a layer of security that helps ensure that connections coming from a DCL are safe to run.

Much like trusted locations, Excel Services has trusted connection libraries for data connections. Excel Services has a mode in which it will process data connections only from DCLs that an administrator has explicitly marked as “trusted” by the server. As mentioned previously, data connections have many security threats associated with them; in many ways processing a data connection can be like running code. By providing trusted connection libraries, Excel Services gives an administrator the ability to allow only specific data connections to be run on the server.

Unsupported Features in Excel Services

Excel Services will not load every Excel file. Workbooks that contain one or more of the following features will not load in Excel Services.

- Workbooks with code. This includes workbooks with Microsoft Visual Basic for Applications (VBA) macros, form controls, and toolbox controls, Microsoft Excel 5.0 dialog sheets, and XLM macro sheets.
- IRM-protected workbooks
- ActiveX controls
- Embedded Smart Tags
- PivotTables based on “multiple consolidation” ranges
- External references (links to other workbooks)
- Workbooks saved in formula view
- XML expansion packs
- XML maps
- Data validation
- Query tables, SharePoint lists, Web queries, and text queries
- Workbooks that reference add-ins
- Workbooks that use the *RTD()* function

- Workbooks that use workbook and sheet protection
- Embedded pictures or clip art
- Cell and sheet background pictures
- AutoShapes and WordArt
- Ink annotations
- Organization charts and diagrams.
- DDE links



Note As noted in the preceding list, Excel Services does not support or load workbooks that contain VBA code. Excel Services will support user-defined functions that are written in managed code. Native code add-ins will not be directly supported, although native code functions called by wrapper functions can be used. For more information about user-defined functions, see “Excel Services User-Defined Functions” on page 178.

When a user publishes a workbook to a document library, the user has the option of viewing it in the browser after the publish process is completed. If the workbook includes a feature or features that are not supported in Excel Services, the user will see a message that indicates this.

In some cases, Excel Services will not display an object in a workbook with full fidelity as compared to Office Excel 2007. For example, charts displayed through Excel Services will not support all the visual effects included in Office Excel 2007.

Workbooks that contain one or more of the following features will load in Excel Services but the features will not be displayed. (The features won't be removed from the file, however, so the next time you open the file in Office Excel 2007, they will be displayed.)

- Split and frozen panes
- Headers and footers
- Page Layout view
- Cell patterns
- Zoom
- Analysis Services' member properties in tooltips
- Some cell formatting, such as diagonal borders and border types not supported by HTML
- Interacting with files. While some features support interactivity in the browser, not all do. For example, users cannot add or rearrange fields in PivotTables when working with workbooks in a browser.

- PivotCharts will not be interactive when viewed in a browser (in other words, you won't be able to filter a PivotChart directly), but if you interact with the PivotTable supplying data for the PivotChart, the PivotChart will update accordingly.
- Users will be able to sort, filter, expand, and collapse data in PivotTables in a browser, but they will not be able to reveal all levels of detail, use SQL Server Analysis Services actions, add or remove fields, or rearrange (pivot) fields.
- In a browser, users can use either Named Object view or the provided navigation controls to move around a workbook, but there is no equivalent of the GoTo command.
- Zoom; Minimize and Maximize. These capabilities do not map well to browser-based viewing of workbooks.
- Switching to Page Layout view. A new view was designed to facilitate printing (an activity that is best performed in the Office Excel 2007 client).
- Goal Seek and Scenario Manager
- Formula auditing (trace precedents, trace dependents, show formulas, and so on)
- Altering a workbook's calculation mode
- Watch window

As described earlier in this chapter, the full authoring of workbooks is not a scenario that was targeted in the initial release of Excel Services. For example, you cannot insert a chart or change a formula after the workbook is opened in Excel Services. The following operations are not supported in this release of Excel Services:

- Inserting a new worksheet
- Inserting a chart
- Creating a table
- Inserting a PivotTable
- Inserting a PivotChart
- Editing formulas
- Entering data into arbitrary cells
- Defining names
- Changing cell formatting
- Altering conditional formatting rules
- Grouping and ungrouping (you can interact with groups once they are defined in Office Excel 2007)
- Creating outlines (again, you can interact with outlines once they are defined in Office Excel 2007)

- Defining consolidated ranges
- Converting text to columns

Excel Services and Reporting in a Portal

In this section, we'll explore Excel Services some more and show how users can interact with workbooks that are stored in SharePoint libraries. We'll also cover some of the steps required to configure Excel Services and Windows SharePoint Services to work together.

Adding a Trusted Location and a Trusted Data Connection Library to an Excel Services Configuration

After installing Office SharePoint Server, you need to configure trusted locations for Excel Services before you can start rendering Excel documents using Excel Web Access. This section outlines the steps for adding a trusted file location and a trusted data connection to an Excel Services configuration.

1. Open SharePoint 3.0 Central Administration, and then click Application Management.
2. On the Application Management page, click Create Or Configure This Farm's Core Services.
3. Click the link for the default Shared Services Provider (SSP). Figure 8-3 shows the Core Services page, which includes a set of links for managing Excel Services.

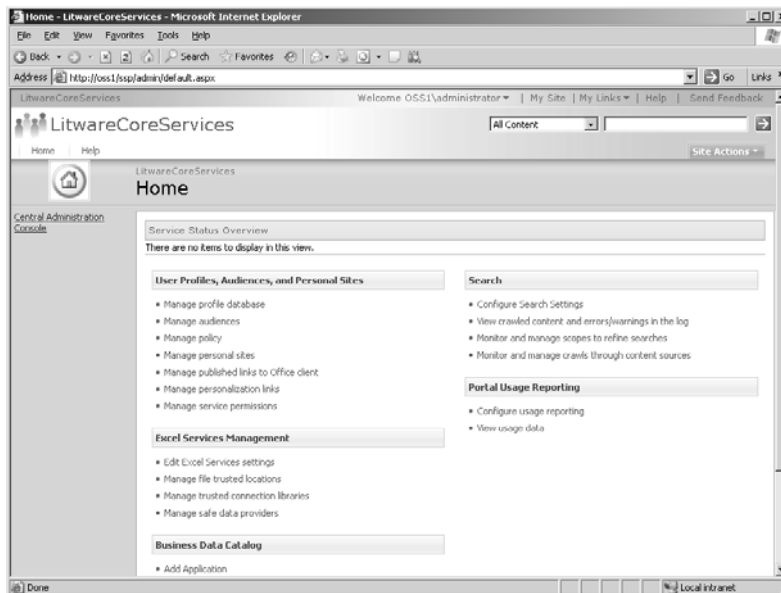


Figure 8-3 The Core Services home page

4. Under Excel Services Management, click Manage File Trusted Locations, and then click Add File Trusted Location. (The list of trusted locations will be empty immediately after the installation.)
5. On the Add File Trusted Location page, shown in Figure 8-4, enter the URL of your server (for example, `http://oss1`). You might also want to select the Children Trusted check box if you want to render documents from all document libraries on this server.

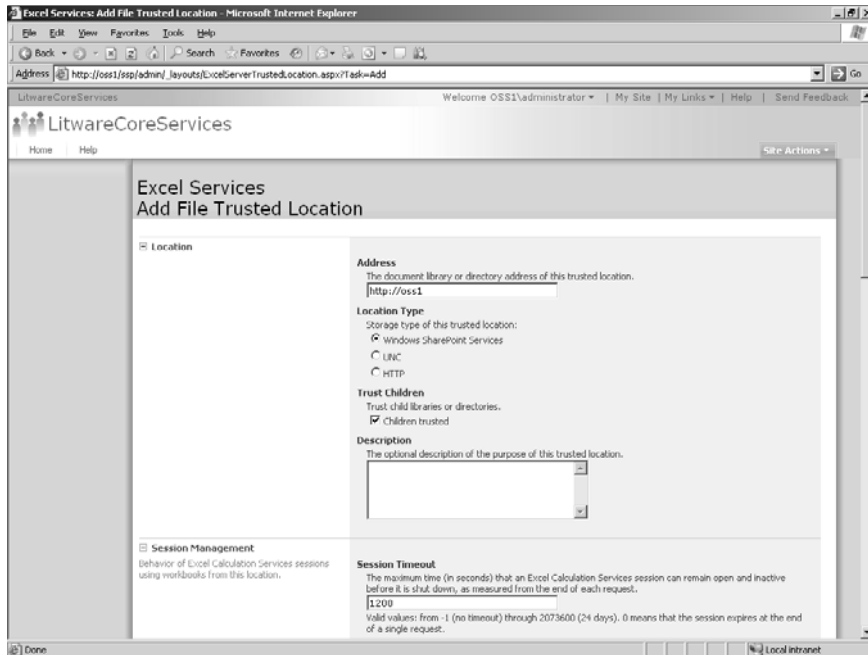


Figure 8-4 The Add File Trusted Location page

6. Scroll down to the External Data section of this page. Select the option under Allow External Data that matches how you plan to use external data with Excel documents. The choices are None, DCL, and DCL And Embedded.
7. After you click OK to add this location, you should see it in the list of trusted locations, as shown in Figure 8-5 on the next page. You can open this page when necessary to edit the settings for this trusted location or add more trusted locations.

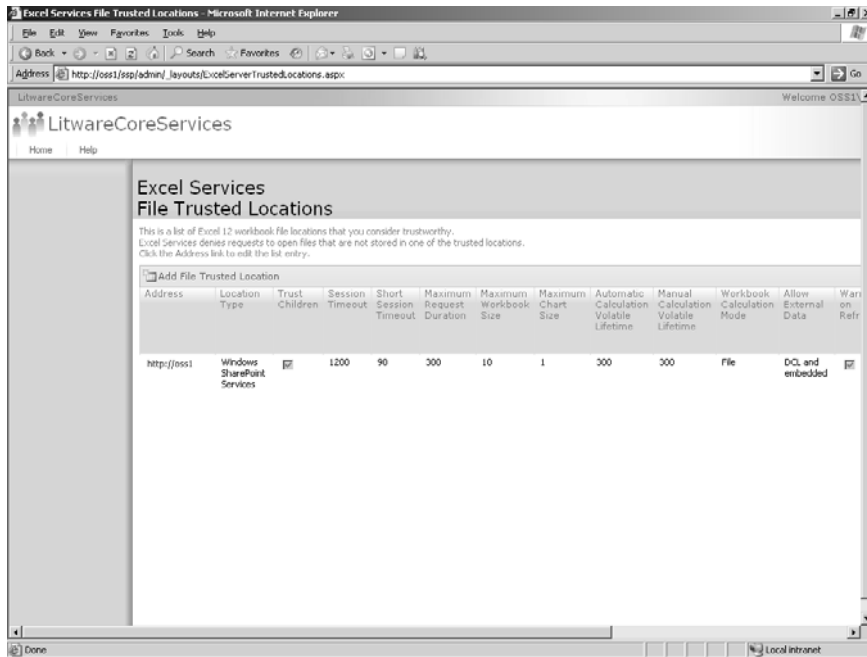


Figure 8-5 The trusted location list with the new item added

Workbooks that are loaded into Excel Services might try to use connection files from a DCL to refresh external data. Excel Services will allow these requests to be executed only if the DCL is explicitly marked as trusted by the Excel Services administrator. The following steps detail how to set up a trusted DCL:

1. On the home page for the site's core services (shown earlier in Figure 8-3), under Excel Services Management, click Manage Trusted Connection Libraries, and then click Add Trusted Connection Library.
2. On the Add Trusted Connection Library page, type the HTTP address of the data connection library (for example, <http://oss1/DCL>) that you want to add. (If this DCL is not available, you will have to create the library first.) Optionally, add a description. Figure 8-6 shows the completed page. After you complete this step, Excel Services is able to load .odc data connection files from the DCL that you configured.

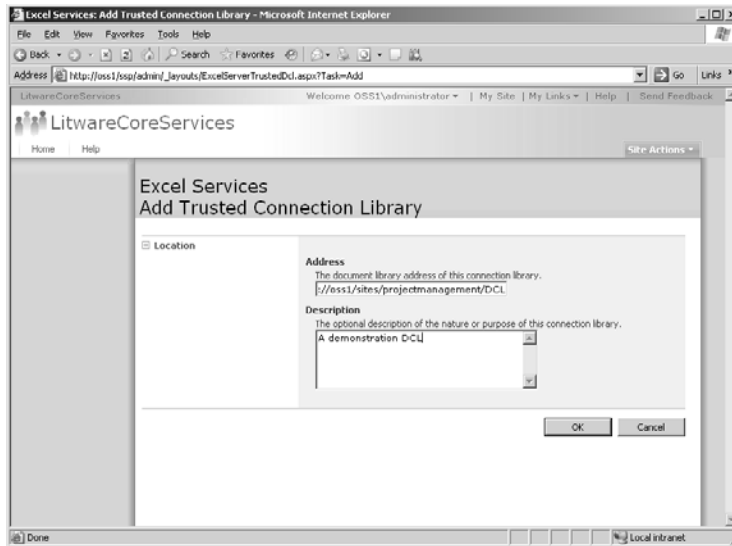


Figure 8-6 Adding a trusted data connection library

Building a Report for Excel Web Access

The scenario in this section shows the processes of building a report based on data stored in a SQL Server Analysis Services Unified Dimensional Model (UDM) using Office Excel 2007 cube functions, and then publishing the report to Office SharePoint Server for viewing within Excel Web Access.

To start, we need to create the data connection to be used in the report. The first step is to open a new workbook in Office Excel 2007 and then add external data to the workbook. You do this by clicking Data, Create New Connection, New Analysis Services Connection, as shown in Figure 8-7 on the next page.

To create the connection, you step through screens in the Data Connection Wizard. You enter the name of the Analysis Services server that contains the cube you want to use, and select the database and the cube. This example uses Adventure Works DW as the database and the Adventure Works cube. After you click Finish in the wizard, Excel prompts you for details about how you want to import the data. For this example, we need to select Only Create Connection, as shown in Figure 8-8 on the next page. You can use the Properties button to name the connection and to set other properties. For example, on the Definition tab of the Properties dialog box, you can click the Export Connection File button to save the connection file to a data connection library or to another location.

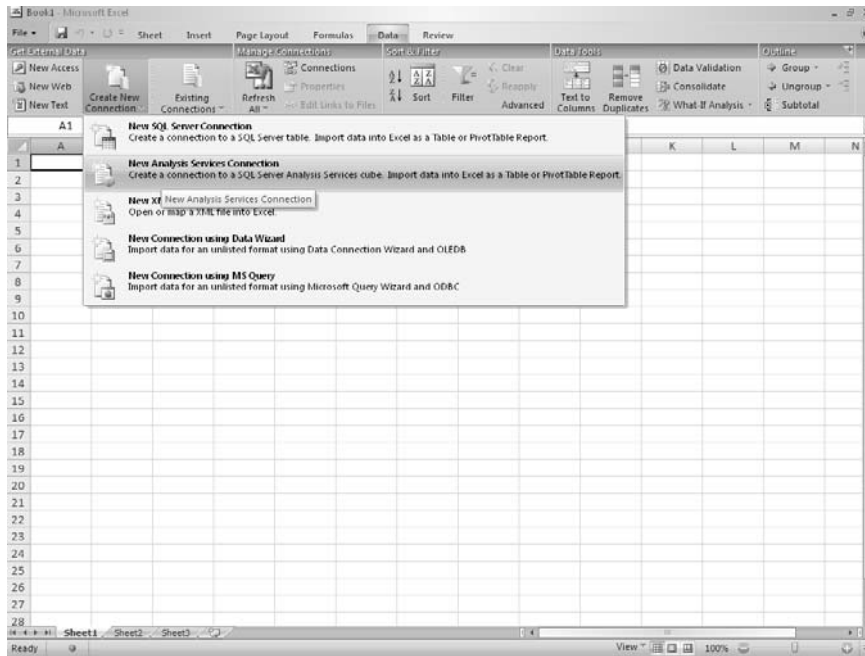


Figure 8-7 Creating a new connection to external data in Office Excel 2007

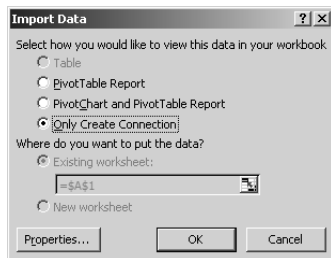


Figure 8-8 Use this dialog box to choose an option for importing external data.

To place the data connection file in a SharePoint data connection library, you can also open the library and upload the file. After checking in the file to the DCL, you need to approve the check-in to make the connection available to users. In the Data Connection Library list view, open the drop-down menu for the data connection, and then click Approve/Reject. Figure 8-9 shows the DCL we created earlier with the Adventure Works data connection file checked in and approved.

Next we'll create a workbook that uses the Adventure Works data connection. In Excel, click Data, Existing Connections, Open Connection. In the Existing Connections dialog box, choose the Adventure Works connection. When prompted for the import option, select Only Create Connection.

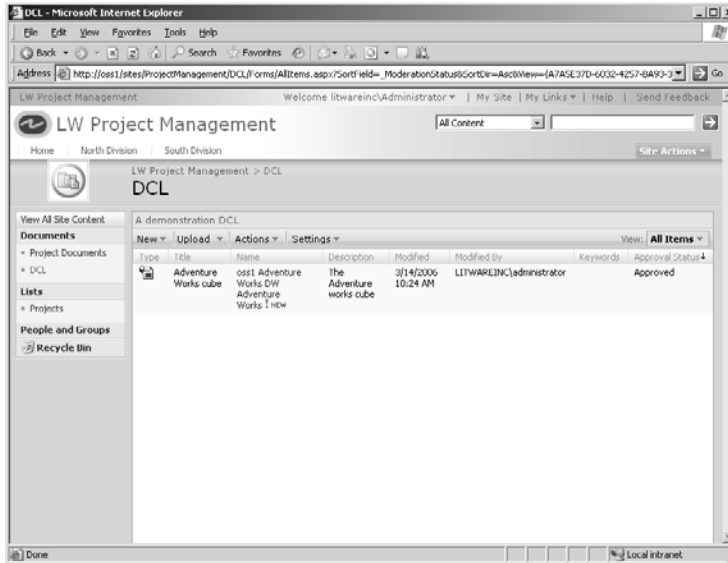


Figure 8-9 The approved data connection file in the trusted data connection library in SharePoint

The next step is to start filling out the body of the report, including a title and header rows. Then we'll add the rows of the table and actual references to the external data. Figure 8-10 shows the data added to Sheet 1 (renamed Reports), including the cells that we'll use later as parameters for the report (defined in D4 and D5) and their initial values (E4 and E5). Setting up these parameters gives users the ability to change the values in the report to reflect a specific year and location. When we publish this workbook to the server, we'll declare these cells as parameters.

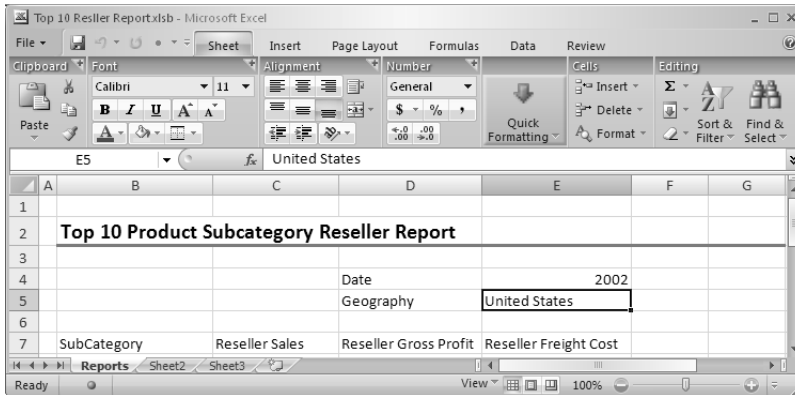


Figure 8-10 The report title, column headings, and parameters set up for the report

Figure 8-11 on the next page shows how the external data connection is used in conjunction with a new Office Excel 2007 feature called cube formulas to return data from our data

source. We entered these formulas on Sheet2 of the workbook. Column A includes descriptive labels of the data the formulas retrieve.

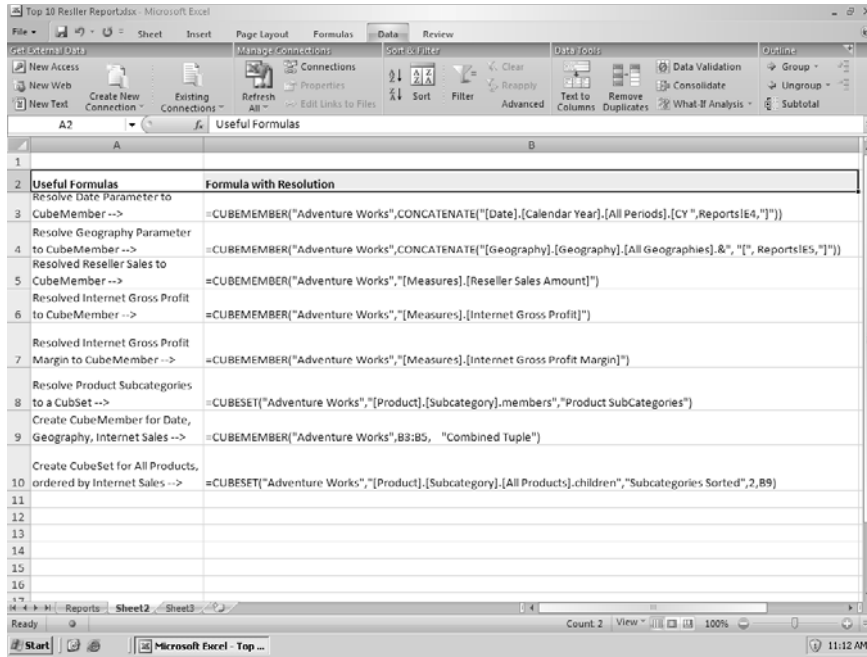


Figure 8-11 Cube formulas used in Excel to retrieve external data from an Analysis Services cube

When you enter cube formulas, Excel provides a list of OLAP-based connections that have been defined in the workbook. When a cube formula expects a *set_expression* argument, as you begin to enter the argument Excel fetches available dimensions, hierarchies, measures, and member names directly from the cube so that you don't need to enter the entire member name by hand.

One final item to note here is that you can provide Excel with any valid multidimensional expression (MDX) fragment that resolves to a member or to a set for *member_expression* and *set_expression* arguments. For example, in cell B10, the set expression *[Product].[Subcategory].[All Products].children* is a valid MDX expression that returns all the product subcategories.

The previous step got the data needed to fill in the body of the report, which now contains the top 10 product subcategories sorted by reseller sales based on the year and geographical location specified by the user. The final formula, found in cell B10, provides the sorted set that will be used to construct each individual row of the report. In the next step, we return to the Reports worksheet and enter the cube formulas that are used to create each row in the report. Here are the cube formulas used for the report's columns:

- B8, SubCategory

=CUBERANKEDMEMBER("Adventure Works",Sheet2!\$B\$10,ROW(B1))

- C8, Reseller Sales

=CUBEVALUE("Adventure Works", \$B8, Sheet2!\$B\$9)

- D8, Reseller Gross Profit

=CUBEVALUE("Adventure Works", \$B8, Sheet2!\$B\$3, Sheet2!\$B\$4, "[Measures].[Reseller Gross Profit]")

- E8, Reseller Freight Cost

=CUBEVALUE("Adventure Works", \$B8, Sheet2!\$B\$3, Sheet2!\$B\$4, "[Measures].[Reseller Freight Cost]")

In cell B8, the formula returns the first item from the *CUBESET* function on Sheet2!B10 by using the formula *ROW(B1)*, which evaluates to 1. The *ROW* function is a great helper function when you're building sheet data reports and you need to retrieve individual items from a set.

To provide the rest of the report's data, you can simply fill down the formulas in the rows. The report body now appears as in Figure 8-12.

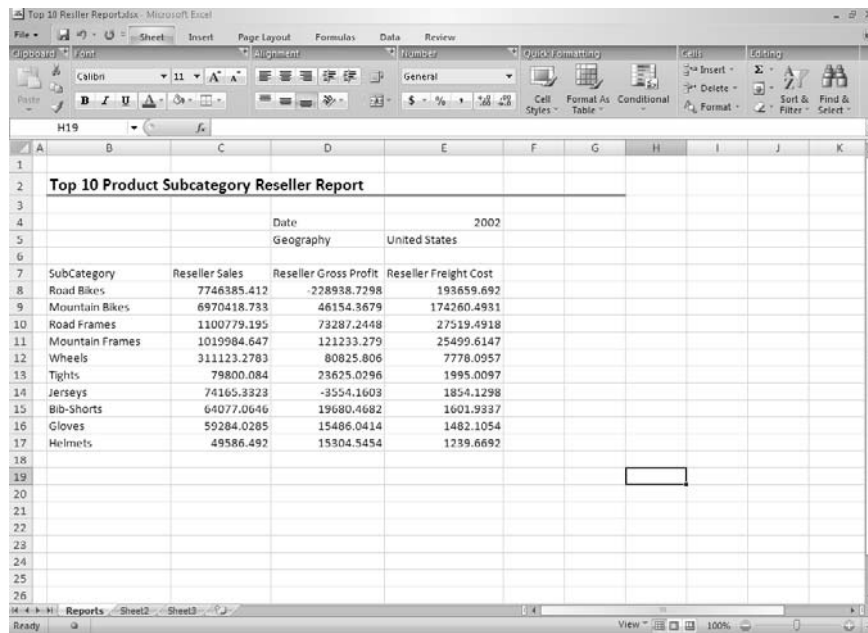


Figure 8-12 The report with data drawn from the external data source

Our goal for this report is to produce a pie chart, and we want the chart to contain a slice that represents the subcategories that didn't make the top 10 list. To accomplish this, we can enter the label All Others in cell B18 and the following cube formula in cell C18:

```
CUBEVALUE("Adventure Works", "[Product].[Subcategory].[All Products]", Sheet2!B3, Sheet2!B4, Sheet2!B5) - SUM(C8:C17)
```



Note You could apply other formatting features to a worksheet such as this. For example, you could format it as a table (using the Insert ribbon) and apply a table style. You could make the number formatting consistent. Finally, you could finish formatting the table by using Office Excel 2007's data bar functionality, which adds a colored background to a cell in proportion to a cell's relative value in a column.

To create a pie chart based on the data in the SubCategory and Reseller Sales columns, we first select these columns and then use the Insert ribbon to insert a chart. After clicking OK, the chart is added to the workbook, as you can see in Figure 8-13. With the chart in place, you can add a chart title or do other chart formatting as you like.

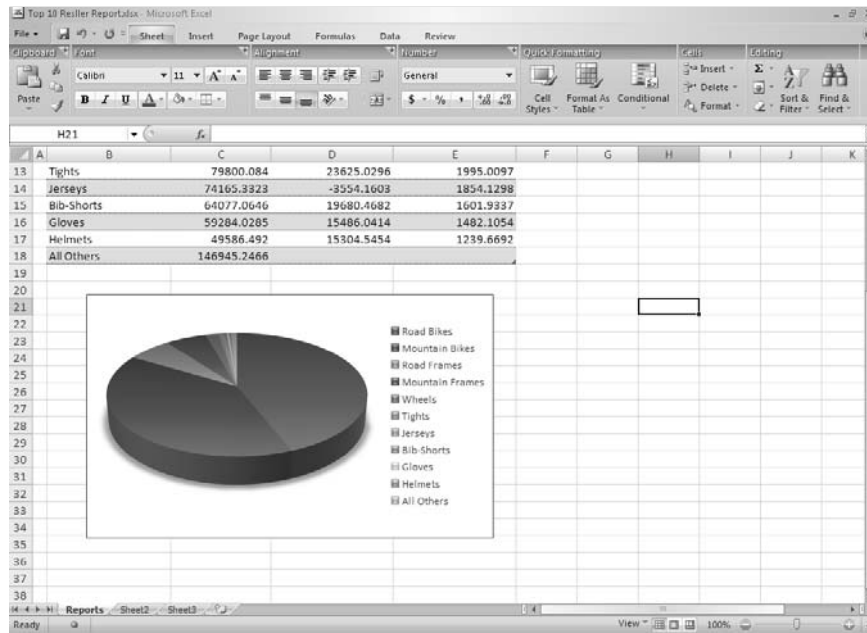


Figure 8-13 The report data represented in a pie chart

The report is complete from a presentation perspective. Now we need to publish this workbook to Office Server so that it can be shared with the rest of the organization. In this scenario, we will share the chart and the Reports sheet but not the entire workbook. As mentioned earlier, we could make the entire workbook available to users, but we'll use the Named Object view here to enable portions of a worksheet to be published. In addition, we'll let users change

the report's data by entering a different country and year through workbook parameters. To add the parameters, we create a couple of named ranges in Excel that represent the cells in which the year and country values are entered, E4 and E5 in the Reports worksheet. (You can name cells by entering the name in the Name box, just to the left of the formula bar.)

To start the process of publishing the workbook to Office Server, we choose File, Send, Publish To Office Server. In the Excel Services Options dialog box, we set up the parameters by clicking the Parameters tab, shown in Figure 8-14, and then adding the two named cells as parameters for the workbook. This list of parameters is based on those cells in the workbook that are defined as named ranges. Named ranges that are defined as a single cell and that only contain values are eligible for being workbook parameters.

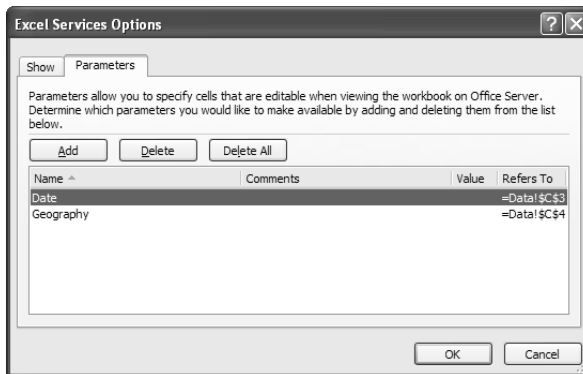


Figure 8-14 Defining parameters that users can interact with when viewing the workbook's data in the browser

On the Show tab, a drop-down list provides three publishing options: Entire Workbook, Sheets, and Items In The Workbook (such as the chart or table in the example we've been showing). We selected the chart and the Reports sheet to be viewable on the server.



Note Remember that regardless of which option you select here for the items that can be viewed on the server, a copy of the entire workbook is saved to the location where you publish the workbook so that cell references and data models are available. For example, in this case the chart relies on data that is in the table (not marked for display) and the table relies on data from Sheet2 (also not marked for display).

Figure 8-15 on the next page shows what the workbook looks like in Excel Web Access, after making the chart and the Reports worksheet available and publishing the workbook. When a user changes the values of the parameters, the chart and table will update to reflect the change in the parameter values. The complete set of data is available because the entire workbook is loaded on the server, but the user has the ability to see only the elements you published.



Figure 8-15 The chart opened in Excel Web Access.

Coding with Excel Web Services

In this section, we'll describe two examples that use managed code to incorporate logic in an Excel workbook into an application. The integration is achieved by calling Excel Web Services to run the Excel portion of the logic on the server. The first example involves a Monte Carlo simulation model for a stock and option portfolio. The user of this application runs a command line tool and entered some arguments for the command. The code passes the arguments to the server through Web service method calls and receives back results. The results are then sent to the console. The second example shows similar code used in part of a simple Web-based mortgage calculator that could be run from a SharePoint Web Part.

Monte Carlo Simulation

We used a console application in this example because it enables us to show a complete application while focusing on the Excel Web Services aspects rather than on user interface considerations. Typically, code that uses this Web service would be a server-side, middle-tier application such as an .aspx page or a Web application.

Figure 8-16 shows two worksheets from the Monte Carlo workbook that we will make available to Excel Services and access through Excel Web Services. The worksheet named Simulation holds the proprietary calculation model that we want to protect on the server and use in an application. For this sample, the model is just a few randomly generated numbers, but the workbook interface to the application is demonstrated just as if this were an elaborate and truly valuable calculation model.

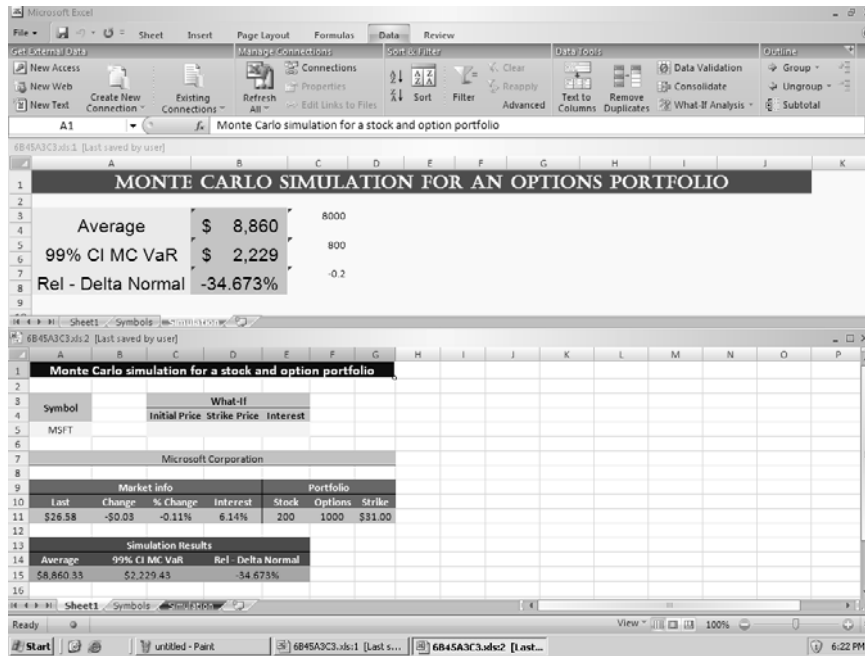


Figure 8-16 The Monte Carlo workbook. We want to use the logic on the Simulation sheet in an application.

To publish the workbook to Office SharePoint Server, we follow the steps described in the previous section, making sure that we don't publish the Simulation sheet. We can then write some code to interact with the Excel workbook using the Web service.

In Microsoft Visual Studio, we create a C# console application project and name it MonteCarlo. We first need to add a Web reference to Excel Web Services, which is http://oss1:500/_vti_bin/excelservice.asmx, and add a *using* statement for the new namespace:

```
using MonteCarlo.myserver;
```

Next we define the interface between our application and the Excel workbook. The interface is a set of named ranges that are associated with input and output cells.

```
class Program
{
    // input & output cell names
    private static string[] inputCells = {"Symbol", "InitialPrice",
        "StrikePrice", "Interest"}
    private static string[] outputCells = {"Average", "VaR",
        "RelDeltaNormal"};
```



Note Both direct range references (for example, B52) and named ranges can be used. Using named ranges isolates the parameter location in the workbook from the application, making the model easier to maintain.

Then we add code to display a usage line when the command is run with an insufficient number of arguments:

```
static void Main(string[] args)
{
    if (args.Length < 1 + inputCells.Length)
    {
        Console.Error.WriteLine("Usage: MonteCarlo <workbookPath>");
        foreach (string argName in inputCells)
            Console.Error.WriteLine(" <{0}>", argName);
        Console.Error.WriteLine("");
        return;
    }
}
```

The usage line will help at run time because it refers to all the required arguments: a path to the server-side Excel workbook and the input cell names that you defined in the previous step.

Next we add code to instantiate and initialize the proxy object:

```
// Excel web services stuff:
// Instantiate and initialize the service proxy object
ExcelService xlsrv = new ExcelService();
xlsrv.Credentials = System.Net.CredentialCache.DefaultCredentials;
```

Setting credentials to *DefaultCredentials* means that your process security context is passed to the Web service for authentication.

The next step is to open the workbook and start a session with Excel Services.

```
// Open the workbook
Status[] status;
string sessionId = xlsrv.OpenWorkbook(args[0], string.Empty, string.Empty, out status)
```

The *OpenWorkbook* method returns a *sessionId*, which is used in subsequent calls as a user's session context. The two empty strings designate default cultures (languages) to be used. The *status* output argument returns an array of noncritical errors (usually empty). Critical errors are returned as SOAP exceptions that you can catch.

We next set the values that are passed on the command line to cells in the workbook, using the array of input named ranges that were defined earlier. After setting these values, we call a method to have the server calculate the new state of the workbook.

```
// Set parameter values into cells
for (int i = 0; i < inputCells.Length; i++)
```

```

status = x1Srv.SetCellA1(sessionId, "Sheet1", inputCells[i], args[1+i]);

// Calculate the workbook.
status = x1Srv.CalculateWorkbook(sessionId, CalculateType.Recalculate);

```



Note When a model is complex and time consuming to calculate, it is often a good idea to turn off automatic recalculation when authoring the workbook in Excel. When automatic recalculation is turned off, the server will not perform recalculations after each parameter value you set, and you need to explicitly instruct the server to perform the new calculations after all parameters are set.

After the workbook is calculated, you can get output values from the server session, using the array of output named ranges that you just defined, and write them out.

```

// Get results and write them out.
foreach (string cellName in outputCells)
{
    object result = x1Srv.GetCellA1(sessionId, "Sheet1", cellName, true, out
        status);
    Console.WriteLine("{0}:\t{1}", cellName, result);
}

```

As a last step, we close the workbook and your server session. This step is important for optimization. If you do not close the session when you are finished with it, the server will still time out the session after an amount of time (controlled by an administrator) passes. This time-out period is usually set according to the needs of graphical applications to allow users to pause between interactions. Unnecessary sessions that remain open consume server resources and can cause decreased overall performance.

```

// Close the session.
status = x1Srv.CloseWorkbook(sessionId);

```

After building the application in Visual Studio, we can run it without arguments to get the usage line.

```

... \Debug>MonteCarlo.exe
Usage: MonteCarlo <workbookPath> <Symbol> <InitialPrice> <StrikePrice> <InterestRate>

```

Now we pass it the path to the Excel workbook and values to the other arguments. Here are the result values that are written out.

```

... \Debug>MonteCarlo.exe "http://oss1:500/report/reportlibrary/MonteCarlo.xlsx" MSFT 26
Average: $80,950.37
VaR: $1,165.32
RelDeltaNormal: -53.026%

```

A Simple Mortgage Calculator

The mortgage calculator example, much like something you would find on a bank's Web site, is similar to the Monte Carlo scenario. Although it's another simple application, it demonstrates key Excel Services concepts such as the following:

- Using a server-side Excel workbook calculation as part of another application.
- Writing a custom, interactive user interface around Excel Calculation Services. (The user interface for this application could be a SharePoint Web Part or any other format that can communicate with Web services.)
- Protecting and maintaining proprietary business models while still providing users with the ability to enter their own inputs and receive answers. There's no need to recode an Excel model just to protect it.

The example here is a simplified version of such a tool, with one twist: the entire calculation is performed on the server by an Excel workbook and not by a function written in a programming language. As far as the users are concerned, the tool does not appear to be related to Excel; it is just an ordinary interactive Web form. A user can type in the mortgage amount, mortgage period length, and interest rate; click Calculate; and see their prospective monthly payment on the form. Behind the scenes, the Web page uses an Excel Web Services session to load the mortgage calculator workbook, set its parameter cells to the values provided by the user, and retrieve the calculated result from the appropriate formula cell.

The entire calculation is performed in a single cell by using the Excel *PMT* function. We name this cell Payment. Similarly, we can define three input parameter cells that are also named, although they don't have to be named to be exposed by Excel Web Services. Cell references can be used as well. However, using named cells better isolates the workbook and the code around it; the code refers to the sheet only through named ranges so that the model can be edited later and laid out in any way that the author sees fit, as long as the parameter and result names remain intact. The workbook is published to a server document library.

The following sample code, which relates to running Excel Web Services sessions, is contained within a single method that is called from a button control's *Click* event. This application would use standard Web Part code for the user interface and other operations.

```
private void CalculateUsingWebService()
{
    Status[] status;
    string sessionId = null;

    // Step 1: Instantiate the web service
    XlMortgageCalcWebPart.Es.ExcelService es = new
        XlMortgageCalcWebPart.Es.ExcelService();

    // Step 2: Set web service link
    es.Url = this.ExcelWebServiceUrl;
}
```

```
// Step 3: Set credentials
es.Credentials = System.Net.CredentialCache.DefaultCredentials;

// Step 4: Start the session
try
{
    sessionId = es.OpenWorkbook(this.MortgageCalculatorWorkbookUrl,
        String.Empty, String.Empty, out status);
}
catch
{
    sessionId = null;
}
if (sessionId == null)
{
    _lblError.Text = "Error opening workbook. Please make sure that the
        correct MortgageCalculatorWorkbookUrl and ExcelWebServiceUrl are
        specified in the Web Part Properties.";
    this.Controls.Clear();
    this.Controls.Add(_lblError);
    return;
}

// Step 5: Set parameters
es.SetCellA1(sessionId, "SimpleCalculator", "MortgageAmount",
    _txtMortgageAmount.Text.Trim());
es.SetCellA1(sessionId, "SimpleCalculator", "MortgageLength",
    _txtMortgageLength.Text.Trim());
es.SetCellA1(sessionId, "SimpleCalculator", "InterestRate",
    _txtInterestRate.Text.Trim());

// Step 6: Get result
object o = es.GetCellA1(sessionId, "SimpleCalculator", "Payment", true,
    out status);
if (o != null)
{
    _lblTotal.Text = Convert.ToString(o);
}
else
{
    _lblError.Text = "Error getting total value from workbook.";
    this.Controls.Clear();
    this.Controls.Add(_lblError);
}
return;
}

// Step 7: End the session
status = es.CloseWorkbook(sessionId);
}
```

To instantiate the Web service, the code creates an instance of the *ExcelService* object, which is generated by Visual Studio when a developer adds a Web reference to the Web service's .asmx file. As you can see, the object is generated in the application's namespace.

The code sets the URL to the Web service and then sets credentials. In this case as well, we set the credentials that are passed to the Web service for authentication to *DefaultCredentials*, meaning that the application's own credentials are used.

To start the session, the code calls *OpenWorkbook*, passing the path to the mortgage calculator Excel workbook (a Web Part property in this case) and receiving a *sessionId*. This *sessionId* is used later in other Web service calls to identify our session.

The code next calls *SetCellA1* to set the three parameter cells to the values that a user of the application has entered in the Web Part form. You can see how named ranges are used, as opposed to direct cell references, to make the code insensitive to layout changes in the workbook and thus more robust.

The call to *GetCellA1* retrieves the calculation's result, the value in the Payment named range. The sample workbook was set to be automatically recalculated, so as soon as the parameters are set, users can expect the result to be available. In some cases, automatic recalculation is turned off while authoring the Excel workbook; a call to *Calculate* is then necessary at the point in the code at which you want to tell Excel Calculation Services to explicitly calculate formulas.

To end the session, the code calls *CloseWorkbook*. This call tells Excel Calculation Services that we are finished with this session and that all resources that were associated with the session can be released.

That's it. In seven lines of code and some exception handling, we have integrated an Excel workbook calculation on the server with our mortgage calculator application. The application provides a custom user interface, Excel Services processes the Excel workbook model, and Excel Web Services ties this functionality together.

While this is a simple example, you can envision more complex calculations represented in workbooks that you can call from an application. The calculations can be provided as a service to applications, while the calculation model is safely hidden and secured.

Excel Services User-Defined Functions

We'll conclude this chapter with some information about user-defined functions (UDFs), which are a mechanism by which developers can extend Excel Calculation Services.

To create a UDF, you must use Visual Studio 2005 because your project needs a reference to the Excel Services UDF dynamic link library, *Microsoft.Office.Excel.Server.Udf.dll*. This assembly has been compiled using .NET Framework 2.0. It is located on the computer where you have installed Office SharePoint Server in the following path:

[drive:]\Program Files\Common Files\Microsoft Shared\web server extensions\12\ISAPI



Note If you use Visual Studio 2003 to create a managed-code UDF, you will not be able to reference *Microsoft.Office.Excel.Server.Udf.dll*. It is not possible for an assembly created using an older version of the .NET Framework to reference an assembly created using .NET Framework 2.0.

UDF assemblies can reside either in a local directory or in the Global Assembly Cache (GAC). In a farm scenario, the local directory path must be identical across the farm. A UDF assembly also has to be present on a local disk on the same computer that runs Excel Calculation Services.

By default, UDF assemblies are disabled. The Excel Services administrative pages in Windows SharePoint Services Central Administration maintain a User-Defined Functions list for each Shared Services Provider (SSP). Each trusted location has an *AllowUdfs* flag as well. The default *AllowUdfs* value is *False*, which means the UDF assemblies in that particular trusted location are disabled. If the *AllowUdfs* value is *False* when a session is started in a workbook that has UDF calls, the UDF calls will fail. A change in this flag takes effect on the next session. Administrators have to register all UDF assemblies and enable them by setting the *AllowUdfs* flag to *True*.

By default, UDF assemblies run with full trust. If you don't want a particular UDF assembly to run with full trust, you must explicitly restrict code access security (CAS) permission for that UDF assembly. You can do this by using the CAS *Exclusive* attribute. Developers can also use the *RequestMinimum* and *RequestOptional* methods in their code to manage permissions.

A UDF class in a UDF assembly can be public or sealed. It cannot be abstract, internal, or private. It must have a parameterless, public constructor. When you create a UDF assembly, you need to mark your UDF class using the *Microsoft.Office.Excel.Server.Udf.UdfClass* attribute.

You must mark UDF methods using the *Microsoft.Office.Excel.Server.Udf.UdfMethod* attribute. Methods that are not marked with the *Microsoft.Office.Excel.Server.Udf.UdfMethod* attribute in the UDF assembly are ignored because they are not considered UDF methods. The *Microsoft.Office.Excel.Server.Udf.UdfMethod* attribute has an *IsVolatile* property. You use this property to specify whether a UDF method is volatile or not. The default value of *IsVolatile* is *False*, which means that particular UDF method is nonvolatile.

A UDF method in a UDF assembly must be public and it must be thread-safe. A UDF method cannot have *ref* or *out* parameters, use the *retval* attribute, use optional arguments, or use unsupported data types. The UDF method supports numeric types (Double, Single, Int32, UInt32, Int16, UInt16, Int64, UInt64, Byte, Sbyte), strings, Boolean, object arrays (one- or two-dimensional arrays, that is, object [] and object [,]), and DateTime data types. Supported return value types are as follows:

- Numeric types: Double, Single, Int32, UInt32, Int16, UInt16, Int64, UInt64, Byte, Sbyte

- String
- Boolean
- Object arrays: The arrays can be one or two dimensions
- DateTime
- Object
- Object (Null)

Here's an example that shows work with a UDF project in Visual Studio 2005. The project is a C# Class Library project named SampleUdf. Before writing the class, we need to add a reference to *Microsoft.Office.Excel.Server.Udf.dll*. To get to this assembly, we click the Browse button in the Add Reference dialog box, and then go to

```
[drive:]\Program Files\Common Files\Microsoft Shared\web server extensions\12\ISAPI
```

If you are working on a computer that does not have Office SharePoint Server 2007 installed, you need to copy this assembly from a computer that includes Office SharePoint Server 2007.

After we add a reference to *Microsoft.Office.Excel.Server.Udf.dll*, the next step is to create some custom functions and mark them with the user-defined function attributes. As mentioned earlier, you must mark a UDF class with the *Microsoft.Office.Excel.Server.Udf.UdfClass* attribute and UDF methods with the *Microsoft.Office.Excel.Server.Udf.UdfMethod* attribute. Here's the completed code for the class showing the attributes.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Office.Excel.Server.Udf;

namespace SampleUdf
{
    [UdfClass]
    public class Class1
    {
        [UdfMethod]
        public double MyDouble(double d)
        {
            return d * 9;
        }

        [UdfMethod(IsVolatile = true)]
        public DateTime ReturnDateTimeToday()
        {
            return (DateTime.Today);
        }
    }
}
```

The first UDF function takes a number (of type *double*) and multiplies the number by 9. The function is a UDF method that is not volatile. (Because the default value for the *IsVolatile* property is *False*—which means that particular UDF method is not volatile—it is sufficient to mark a nonvolatile UDF method as *[UdfMethod]*. You don't need to mark it as *[UdfMethod(IsVolatile = false)]*.) The second function returns the current date using the *System.DateTime.Today* property. This function is marked volatile.

Remember that you need to deploy a UDF assembly to a location that has been designated a trusted location in Excel Services and also enable the UDF. You enable UDFs for a location using links for Excel Services Management on the Core Services home page, which you've seen earlier in Figure 8-3.

Two links are used to enable UDFs: Manage Trusted File Location and User-Defined Functions. You use the first link to edit the properties of a trusted location, enabling an option to allow user-defined functions. You use the User-Defined Functions link to specify a name of a particular UDF assembly that you want to trust.

Microsoft Office InfoPath 2007 and Microsoft Office Forms Server 2007

In this chapter:

Key Scenarios	184
Components and Architecture of Office Forms Server 2007	185
Data Connections and Office Forms Server 2007	186
Designing Form Templates	187
Deployment	189
Security	190
Custom ASP.NET Pages with the InfoPath Form Services Control	190
Automating Office Forms Server 2007 Administration Tasks	191
Working with InfoPath Forms and Forms Server	193
Embedding an InfoPath Form in an Application	201

Microsoft Office Forms Server 2007, like Excel Services, builds on Microsoft Windows SharePoint Services 3.0 and Microsoft Office SharePoint Portal Server 2007. Office Forms Server 2007 can be used to render editable Microsoft Office InfoPath 2007 form templates in a browser in HTML, making the forms available to users who don't have Office InfoPath 2007 installed on their client computers.

In this chapter, we'll provide an overview of Office Forms Server 2007, including a description of classes that can be used to automate administrative tasks. We'll describe some of the situations in which Office Forms Server 2007 supports the use of InfoPath forms, its architecture, and some aspects of designing form templates that will be enabled for viewing in a browser. We will also illustrate some of the steps involved in deploying a form template to Office Forms Server 2007 and how you can use an InfoPath form on a Web page or in a Windows application.

Key Scenarios

The introduction of Office Forms Server 2007 greatly increases the number of scenarios in which InfoPath form templates can be deployed. (Each scenario, of course, would entail specific considerations for deployment topologies, scaling requirements, workflows, and other information.) Some of these scenarios include the following:

- An international importing company could use Office Forms Server 2007 to query suppliers about inventory.
- Local government agencies could use Office Forms Server 2007 to manage processes over the Internet, such as contractor permit applications and approvals.
- A department within a large organization could deploy Office Forms Server 2007 to collect information from planning meetings with customers, using forms that are connected to a database. This configuration would enable employees to retrieve and analyze the data they collect from many different meetings at a later time.
- An organization such as an insurance company could use Office Forms Server 2007 to communicate with agents in addition to policyholders and business partners. All of the processing for documents such as insurance claims could be managed online.

The main aim of Office Forms Server 2007 is to enable users who do not have Office InfoPath 2007 installed to view and fill out forms while working in a Web browser. Form templates (.xsn files) designed in Office InfoPath 2007 are translated into browser-editable forms that run on Office Forms Server 2007. Form templates with custom business logic are deployed by an administrator to the Windows SharePoint Services content database and are safe-listed in the configuration database. Form templates are managed through a global list of form templates on the SharePoint Central Administration form template management page, which is shown in Figure 9-1.

The same form template can be designed to be deployed to Office Forms Server 2007 (making it available through a Web browser when Office InfoPath 2007 is not available) and to run in Office InfoPath 2007. A form that is designed for both Web browsing and for Office InfoPath 2007 is generally restricted to using only features that are common to Office InfoPath 2007 and the Office Forms Server 2007 browser-based run times. For example, business logic built into form templates that are compatible with Office InfoPath 2007 and Office Forms Server 2007 can be developed only in C# or Microsoft Visual Basic .NET. These development languages can only call members of the new InfoPath managed-code object model provided by the *Microsoft.Office.InfoPath* assembly.

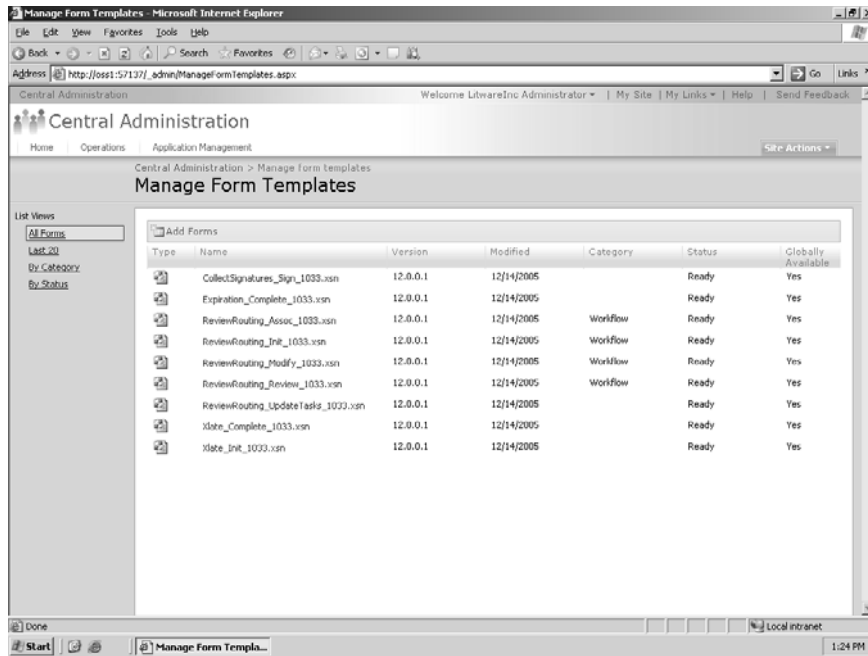


Figure 9-1 Form templates are managed in SharePoint Central Administration.

Components and Architecture of Office Forms Server 2007

The rendering component of Office Forms Server 2007 is the FormServer.aspx page, which resides in the _layouts folder of the server. This page can accept multiple parameters, including the following:

- *XMLLocation* populates the Web-based form template with data that overrides the default form template data.
- *XSNLocation* is a pointer to the form template solution file (.xsn).
- *OpenIn* specifies whether the form should be opened in Office InfoPath 2007 or in the browser.
- *SaveLocation* overrides *XMLLocation* as the default location where a submitted form is saved. This parameter is optional.
- *SourceLocation* is the location to which the browser navigates after a form is submitted.
- *DisableSave* is a property used to hide the Save button in a Web-based form.

For URLs with an XML file or an XSN, logic in FormServer.aspx sends the form template to InfoPath if it is present on the client computer; otherwise, it renders the template in the browser. FormServer.aspx uses a special header value in the page and a registered MIME type to launch Office InfoPath 2007. An option on the Advanced Settings page for a form library (named Opening Documents) specifies whether to open a form in the InfoPath client if the client is available.

Client Architecture and Postback Optimization

When a Web-based form template is rendered, an XML payload and HTML are sent to the browser along with supporting JavaScript files, which are cached in the temporary browser cache. Subsequent postbacks from the browser optimize the changes in the XML payload sent from the server, and local scripts handle the HTML update of the page. On custom pages, if a full reload of the page occurs, operations are similar to loading a page for the first time in that the full XML payload and the HTML are sent to the browser.

The Form Template Converter

When a form is converted, the XSN file (which corresponds to a form template) is converted to the files needed by Office Forms Server 2007 to run the same form in the browser. Form conversion will log an entry if certain XSL constructs are encountered, such as *apply-templates* with no matching template, *apply-templates* with multiple matching templates, *xsl:imports*, and others.

Form conversion will stop if the operation encounters either unsupported controls or unsupported Office InfoPath 2007 features for which Office Forms Server 2007 cannot provide an appropriate behavior.

Data Connections and Office Forms Server 2007

Some differences exist with respect to how data connections behave for a form template opened in Office InfoPath 2007 and a Web-enabled form template that is opened from Office Forms Server 2007. In general, Web-enabled form templates should use Web services for retrieving and submitting data. If using a Web service is not possible, developers must address some limitations about data connections when designing a template for the server. For example, the Human Workflow Services adapter set is not supported on the server. In addition, the database adapter becomes read-only (submit is disabled). When using a Web service with *DataSet* objects, changes in the data are not tracked.

The following types of data connections are available through Office Forms Server 2007:

- Database
- E-mail

- HTTP post
- SharePoint (the native API is used for same-domain data connections; Distributed Authoring and Versioning [DAV] extensions are used for cross-domain connections)
- SharePoint list
- Web services
- XML file

Designing Form Templates

The following sections outline which form template controls are supported in Web-enabled form templates, levels of support for different browsers, and considerations for designing form templates in Office InfoPath 2007.

Controls Supported on Office Forms Server 2007

The following standard controls are supported on Office Forms Server 2007:

- Text box
- Rich-text box
- Drop-down list box
- List box
- Date picker
- Check box
- Option button
- Button

Office Forms Server 2007 also supports the following container controls:

- Repeating section
- Repeating table
- Section
- Optional

In addition, hyperlink and expression box controls (two of the advanced controls) are supported.

Controls Not Supported on Office Forms Server 2007

The following controls are not supported on Office Forms Server 2007:

- Combo box
- Multiple-selection list box
- Master/Detail
- Bulleted, numbered, and plain list
- Picture
- Ink picture
- File attachment
- Vertical label
- Scrolling and horizontal region
- Horizontal repeating table
- Choice group
- Repeating choice group
- Choice section
- Repeating recursive section
- Custom controls

Browser Support

Office Forms Server 2007 supports a number of different browsers on various operating systems. The extent of the support is based on four levels.

- Level 1 offers full fidelity and provides the best experience with a Web-enabled form.
- Level 2 provides a fully functional experience, but certain features (such as a date picker control) might not be available on some of the browsers.
- Level 3 does not provide full fidelity. Certain features might not work and rendering will differ significantly between browsers.
- Level 4 offers no support and possible blocking.

The following table lists browsers, operating systems, and corresponding levels of support.



Note The information in the following table is subject to change.

Browser Support Matrix	Operating System	Browser
Level 1 Windows SharePoint Services Forms Administration	Windows 98, Windows Me, Windows 2000, Windows XP, Windows Server 2003	Internet Explorer 6.x (plus the 64-bit version)
Level 2 Internet, Windows SharePoint Services Site Administration, Form Filling for a majority of users	Windows 98, Windows Me, Windows 2000, Windows XP, Windows Server 2003	Internet Explorer, Firefox, Netscape 7.2 (moving to 8.x)
	UNIX/Linux	
	Mac OS X	Firefox, Safari 1.2
Level 3 Limited support	UNIX, Linux, Windows (versions other than above)	
Level 4 No support	Everything else	Everything else

Controlling Postback Behavior When Control Values Change

The properties dialog boxes for controls in Web-enabled form templates provide settings to manage when and whether the browser will post back data and refresh the page when the control's value changes. These postback settings can be found on the Browser Forms tab of a control's properties dialog box.

Deployment

A codeless form template can be deployed to Office Forms Server 2007 by using the Office InfoPath 2007 user interface. A form template containing code that encapsulates custom business logic must be activated on the forms server by an administrator and requires Administrator privileges.

Custom business logic assemblies should be strong-named to support side-by-side versioning of form templates when business logic changes. If you upgrade a form template with changes to its business logic, the identity of the assembly should also be changed because the assemblies run in the same Windows SharePoint Services application domain and cannot be dynamically unloaded.

If a form template needs to be accessible from inside a firewall and externally (as in an extranet scenario), Office Forms Server 2007 works with Windows SharePoint Services 3.0 Alternate Access Mapping (AAM). AAM works for all connections in the form template, in addition to multiple URLs to the same form template, but it will not resolve linked image controls, hyperlinks in user data, and database and e-mail data connections.

Deploying a form template to Office Forms Server 2007 is preceded by a verification step that involves passing the form through the converter to ensure that the form is server compliant. A verification page will return a list of any errors and warnings for the form template.

Deployment can be accomplished either directly or as a combination of uploading and publishing. Uploading is the process of putting the form template on the server, and publishing is the process of placing the form templates into sites on the content database so that users have access.

Security

As mentioned earlier, only server administrators can deploy form template solutions that contain code. Users who have either the Contributor or Web Designer role can create a form library based on form templates that do not contain code, as they could in previous versions of Windows SharePoint Services. Users with these roles can also create a form library with domain-based form templates that contain code or fully trusted form templates that contain code, but these form templates will work only in Office InfoPath 2007.

Here are some other considerations about security when using Office Forms Server 2007:

- Office Forms Server 2007 relies on the Windows SharePoint Services 3.0 AppDomain to load and run business logic code.
- Office Forms Server 2007 supports only domain-based and fully trusted solutions.
- Office InfoPath 2007 form templates will get the ASP.NET/Medium permission set as defined in the WSS_MediumTrust policy file. Office Forms Server 2007 uses PermissionDeny to remove the permissions that are defined in the Domain_new security profile.
- Cross-domain access is not allowed on Office Forms Server 2007. The form template receives a security exception when it crosses domain boundaries. In contrast, on the Office InfoPath 2007 client a user is prompted, which is the same behavior as in InfoPath 2003. This difference creates scenarios in which the form template does not work on the server but does run on the client.

Custom ASP.NET Pages with the InfoPath Form Services Control

Office Forms Server 2007 includes a control, named *XmlFormView*, in the *Microsoft.Office.InfoPath.Server.Controls* namespace. (You'll see an example that uses this control later in the chapter.) Developers can use the *XmlFormView* control to host Office InfoPath 2007 form templates in a custom ASP.NET page. Hosting the control allows you to manage the appearance of the Web page and provides Web-based form templates for users to fill out. The *XmlFormView* control includes various properties, methods, and events that allow programmatic access to operations such as initializing and closing the form template and a "submit to host" mechanism that can be used to send an event to the parent process, such as the .aspx page.

Some of the limitations of working with the *XmlFormView* control include the following:

- Only one *XmlFormView* control can be used per page.
- Development in Visual Studio must occur on the same machine on which Office Forms Server 2007 is installed.
- The client-detection logic available on standard Web-enabled form templates will not work on custom pages hosting the control.

Automating Office Forms Server 2007 Administration Tasks

The Office Forms Server 2007 object model allows an administrator to automate tasks through the use of the .NET Framework. Many of these tasks can be managed through command-line switches run with *Stsadmin.exe* as well.

The main objects in the *Microsoft.Office.InfoPath.Server.Administration* namespace include the following:

FormsService class The highest level of the forms server topology represents Office Forms Server 2007 Server Administration on the Web farm. The *FormsService* class extends *Microsoft.SharePoint.Administration.SPService*.



Note Windows SharePoint Services defines *SPService* as a farm-wide application or the base object for any root “controller” object for a server application.

Here is an example of retrieving and using the *FormsService* class:

```
SPFarm farm = SPFarm.Local;
FormsService ser = farm.Services.GetValue<FormsService>("");
Console.WriteLine(ser.FormTemplates.Count);
```

The following table lists the properties of the *FormsService* class:

Property	Description
<i>Instances</i>	An <i>SPServiceInstanceCollection</i> inherited from <i>SPService</i> . A collection of <i>FormsServiceInstances</i> , where there is one <i>FormsServiceInstance</i> per <i>SPServer</i> on the <i>SPFarm</i> .
<i>FormTemplates</i>	A <i>FormTemplateCollection</i> , an enumeration containing all of the <i>FormTemplate</i> objects registered on the farm.
<i>DataDefaultConnectionTimeout</i>	Returns an <i>int</i> representing the default data connection timeout in milliseconds.
<i>DataMaxConnectionTimeout</i>	Returns an <i>int</i> representing the maximum data connection timeout in milliseconds.

FormTemplate class The *FormTemplate* class represents an Office InfoPath 2007 form template or an .xsn file that has been deployed to the forms server. All *FormTemplate* classes on the server are contained in *FormTemplateCollection* on the *FormsServerService* class. The following tables list the methods and properties of the *FormTemplate* class.

Method	Description
<i>Activate(SPSite site)</i>	Activates or reactivates a previously uploaded form template to the site specified by <i>site</i> . Fails if the form template is deployed by using <i>RegisterFormTemplate</i> .
<i>Deactivate(SPSite site)</i>	Inverse operation of <i>Activate</i> . Fails if a form template is deployed by using <i>RegisterFormTemplate</i> .
<i>GetAbsoluteUri(SPSite site)</i>	Returns the absolute URL of the form template at the given site in the current user context.
<i>Quiesce(TimeSpan maxDuration)</i>	Starts quiescing the form template for the time specified by <i>maxDuration</i> .
<i>QuiesceEndTime()</i>	Returns the time the form template will stop quiescing as a <i>System.DateTime</i> .
<i>Unquiesce()</i>	Stops quiescing the form template.

Property	Description
<i>Category</i>	Gets or sets the category to which the <i>FormTemplate</i> is assigned.
<i>CreatedTimeUtc</i>	Gets the creation time as universal time (not local time).
<i>DataConnectionFileReferences</i>	Gets the collection of data connection file references.
<i>Description</i>	Gets the description attribute from the form template .xsn file.
<i>FeatureId</i>	Gets the ID (GUID) of the feature (<i>SPFeatureDefinition</i>) that contains this <i>FormTemplate</i> .
<i>FormId</i>	Gets URN of the <i>FormTemplate</i> .
<i>Id</i>	Gets or sets the ID (GUID) of the <i>FormTemplate</i> in the <i>FormTemplateCollection</i> object. Inherited from <i>SPPersistedObject</i> .
<i>IsFullTrust</i>	Gets a Yes/No string that specifies if the form template is full trust.
<i>IsSigned</i>	Gets a Yes/No string that specifies whether the form template is signed.
<i>Locale</i>	Gets a string that specifies the form template locale (for example, en-US).
<i>ModifiedTimeUtc</i>	Gets the last modified time as universal time (not local time).
<i>QuiesceState</i>	Gets the current quiesce state as a <i>QuiesceMode</i> enumeration: Normal, Quiesced, or Quiescing.
<i>SolutionId</i>	Gets the ID (GUID) of the solution that contains the XSN.
<i>ViewStateEnabled</i>	Gets or sets whether the form template is currently using view state. This property can only be set when the form template is quiesced.

Working with InfoPath Forms and Forms Server

In the sections that follow, you'll see examples of an InfoPath template that is used by our fictitious Litware consultants to fill in daily time sheets. The template will be deployed so that consultants can use either the InfoPath client or a browser to enter time sheet data. We'll also demonstrate the *XmlFormView* control and some other new ASP.NET and Windows controls that developers can use to embed an InfoPath form within an application.

Creating an InfoPath Form to Capture Time Sheet Data

To start this example, we will create two InfoPath template parts, which are form elements that can be saved and used in template after template. The first template part is for the Litware banner, and the second is a form footer that contains text that Litware uses on its internal documents.

In InfoPath, in the Fill Out A Form dialog box, click Design A Form. In the Design A Form dialog box, select the Template Part option and base the template part on a blank template. Also select the Web Browser Enabled option because this form template will be deployed to an Office Forms Server 2007 site later. Figure 9-2 shows the Design A Form dialog box with the options selected.

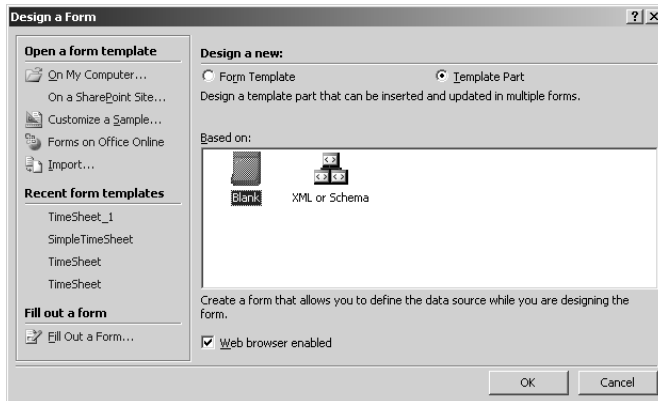


Figure 9-2 The Design A Form dialog box. Note the Web Browser Enabled option.

After the InfoPath design window opens, in the Design task pane, click Layout and then double-click the Two-Column Table option. In the table's first cell, we enter the text **Litware Inc**, and in the second cell, the text **Our code is so tasty, it will melt in your mouth**. We can then apply some borders and shading to the table and formatting to the font. Figure 9-3 on the next page shows the template part up to this point.

We add a second row to the table and merge this table's cells. In the merged cell, we add the text **Add here the title of the template**, and then save the template part as Banner.xtp.

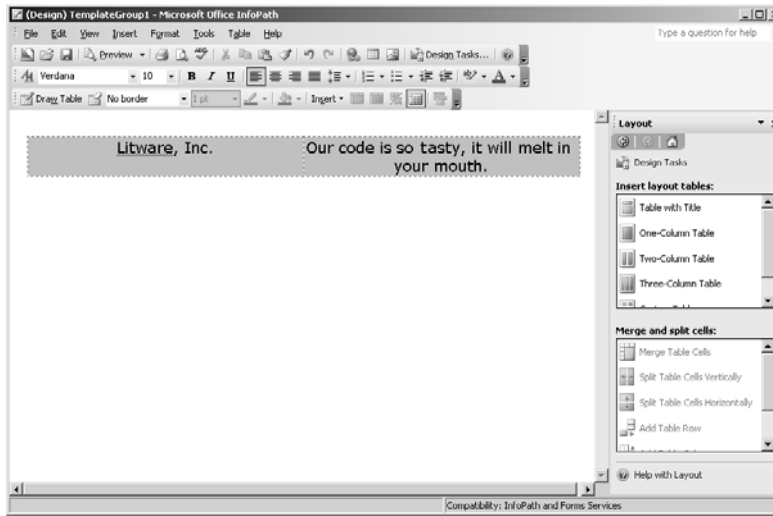


Figure 9-3 The initial appearance of the banner template part

For the second template part, we follow the same initial steps, choose the One-Column Table option for the layout, and then add the text **Litware Inc. Internal Document - All Rights Reserved**. This template part is saved as Footer.xtp.

We now have two reusable InfoPath blocks and can design an InfoPath template that will incorporate these template parts and be used to capture time sheet information. To begin the template, we need to take care of the layout and the controls, and then we can turn our attention to linking the template to some data sources and adding its functionality. Figure 9-4 shows the completed template in design view. Like the template parts, the form template was designed so that it is enabled for Web browsing.

As mentioned earlier, InfoPath does not support all types of controls on a form that is enabled for Web browsing. When you click Controls in the Design task pane, InfoPath displays the controls that can be used, and a message box at the bottom of the task pane indicates that some controls are not compatible with server forms and have been hidden. Clicking the message in the task pane displays a message box, shown in Figure 9-5, that lists controls that aren't compatible with server forms, including a combo box, file attachments, a multiple-selection list box, and others.



Note During development of a form template, you can run the Design Checker to view and correct any compatibility errors with the selected run-time environments. Once you choose a form mode, you can switch the compatibility by using the Change Compatibility Settings link in the Design Checker task pane.

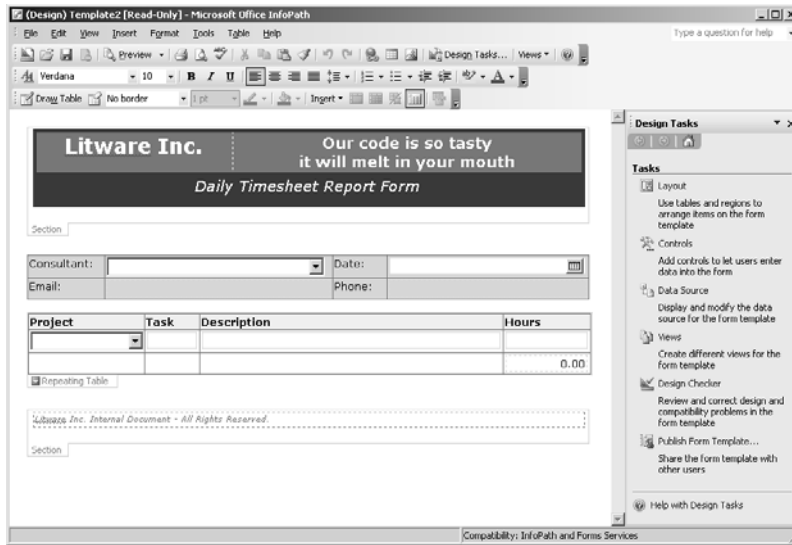


Figure 9-4 The time sheet form template in design view



Figure 9-5 InfoPath displays the controls that are not compatible with a form that is enabled for Web browsing.

To add the banner and the footer template parts to this form, first click the Add Or Remove Custom Controls link at the bottom of the pane. Add the template parts, and then you can drag the banner and the footer onto the blank form view. For this template, the text “Daily Timesheet Report Form” is added to the placeholder in the second row of the banner.

Here’s a summary of the rest of the work required to lay out the form as it’s shown in Figure 9-4. Some additional borders and shading were applied to format the table.

- Using the Layouts task pane, we added a custom table with four columns and two rows. We added Consultant as the label in the first cell of row 1; Email in the first cell of row 2; Date in the third cell of row 1; and Phone in the third cell of row 2
- Using the Controls task pane, we added the following controls to the view:
 - A drop-down list box in the second cell of row 1, for consultants

- A date picker in the fourth cell of row 1
- An expression box in the second cell of row 2
- An expression box in the fourth cell of row 2
- We added a repeating table to the form, with four columns. The column headers are Project, Task, Description, and Hours. The text box under Project was changed to a drop-down list box using the Change To command on the shortcut menu.

We've built this form largely from scratch, but as we add controls to the form, InfoPath generates an XSD schema. You can see the default names InfoPath assigns to the elements by clicking on Data Source in the Design task pane. You can modify the schemas by changing the default names and types, for example. Using the Data Source task pane, we renamed the default elements as follows:

- myFields to timesheet
- myFields_1 to banner
- myFields_2 to footer
- field1 to consultant
- field2 to date
- group1 to items
- group2 to item
- field3 to project
- field4 to taskID. We also changed the type to Whole Number (integer)
- field5 to description
- field6 to hours. We also changed the type to Decimal (double)

We continued building the form by adding a group under the timesheet element, called header, using the Move Up command to position the new group. We moved the consultant and date fields to the new group, and moved the banner and the footer elements to the bottom. Figure 9-6 shows the Data Source task pane after these changes have been made.

We completed the form's layout by right-clicking the table to open the Repeating Table Properties dialog box. On the Display tab of the dialog box, we selected the Include Footer check box. Finally, we added an expression box control to the Hours column in the footer row, used the formula editor to insert a Sum function for the Hours field, and changed the type of the expression box content to Decimal.

With the form template set up with controls and formatting, we can begin to take care of populating the drop-down list boxes with data. The first data connection we'll create retrieves consultant data from a database (a simple Microsoft Access database in this example). The second data connection retrieves project data from a SharePoint list.

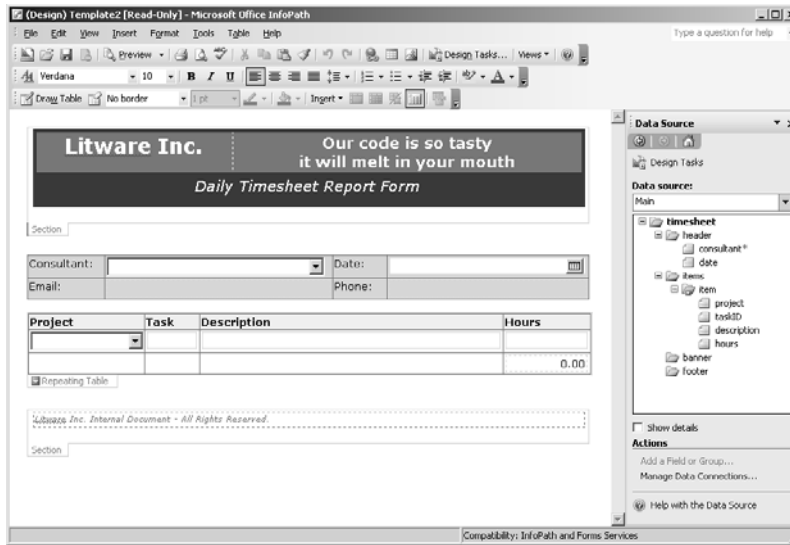


Figure 9-6 The Data Source task pane shows the names for the elements in the template.

In InfoPath, we start by selecting Tools, Data Connections. Stepping through the series of screens, we specify first that this connection is for receiving data. Next we select Database, and then browse to the database file and select the Consultants table. We also choose to store the consultant's data in the template to support offline scenarios.

For the second data connection, we specify the URL to the SharePoint site where our Projects list is stored and then select the list by name. We included the fields named Code, Project Title, and Manager, and here as well, we select the option to store the project data in the template to support offline work.

We associate the Consultant drop-down list box with the Consultants data connection by double-clicking the control, selecting Look-Up Values From An External Data Source, and specifying Consultants as the data connection. We also specify the Name field as the field to be displayed and stored in the XML that is generated from the data entered in the form. Likewise, the Projects drop-down list is linked to the Projects data connection, with Code as the field that is displayed and stored.

The next phase in creating the template is to add some validation and rules to the controls, which are set up in the properties dialog box for the control. Here are the rules and data validation conditions that were added to the form:

- The Consultant field cannot be blank.
- Today's date is set as the default date.
- Users are restricted from entering a date that is in the future.

- The e-mail and phone fields are filled in automatically when a user selects a consultant. These lookups are performed by using XPath expressions such as the following:

```
xdxDocument:GetDOM("Consultants")/dfs:myFields/dfs:dataFields/d:Consultants/
@Telephone[../@Name = xdxDocument:get-DOM()/my:timesheet/my:header/my:consultant]
```

- Change the background color of the expression box that calculates the total of hours to red if the amount is higher than 8.

As a last step, we add a button control to the form template that will be used to submit data to a Web service. Here's the code for the *ProcessSheet* class that defines the *SubmitTimesheet* Web method that's invoked through the Web service.

```
using System;
using System.Xml;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
using System.Data.OleDb;
using System.IO;

/// <summary>
/// Summary description for ProcessSheet
/// </summary>
[WebService(Namespace = "http://www.litwareinc.com/")]
[WebServiceBinding(ConformsTo = wsProfiles.BasicProfile1_1)]
public class ProcessSheet : System.Web.Services.WebService {

    [WebMethod]
    public Boolean SubmitTimesheet(string timesheet)
    {
        //-- just for debugging
        StreamWriter sw = new StreamWriter(@"C:\data\temp.xml");
        sw.Write(timesheet);
        sw.Close();

        bool returnValue = true;

        try
        {
            StoreInDatabase(timesheet);
        }
        catch
        {
            returnValue = false;
        }

        return returnValue;
    }
}
```

```

private void StoreInDatabase(string timesheet)
{
    //-- load the xml
    XmlDocument doc = new XmlDocument();
    doc.LoadXml(timesheet);

    //-- drop the timesheet items in the Access database
    string consultant = doc.SelectSingleNode("//consultant").InnerText;
    DateTime date = Convert.ToDateTime(doc.SelectSingleNode("//date").InnerText);

    //-- grab the current data in the database
    OleDbConnection conn = new
        OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Data\
            Timesheet.mdb");
    conn.Open();
    OleDbCommand cmd = new OleDbCommand();
    cmd.Connection = conn;

    //-- add the new stuff to it
    XmlNodeList items = doc.SelectNodes("//item");
    foreach (XmlNode item in items)
    {
        cmd.CommandText = string.Format("INSERT INTO TimeSheetData
            (Consultant,Project,ItemDate,TaskID,Description,Hours) " +
            " VALUES('{0}','{1}','{2}',{3}','{4}','{5})", consultant,
            item.FirstChild.InnerText, date, item.ChildNodes[1].InnerText,
            item.ChildNodes[2].InnerText,item.ChildNodes[3].InnerText);
        cmd.ExecuteNonQuery();
    }

    //-- clean up
    conn.Close();
}
}

```

We need to define a data connection for this Web service as well, although this connection is to submit data. After selecting Web Service as the type of connection, enter the URL to the Web service (<http://localhost:6154/TimeSheetServices/ProcessSheet.asmx>, for example—the port number will vary). Select the *SubmitTimesheet* method, and then select the timesheet field as the value for the input argument. Select XML Subtree, including the selected element, as the value for the Include option.

With the data connection set up, we can create a rule that defines the button's behavior. Double-clicking the button opens the Button Properties dialog box. The rule we define has no conditions but performs the following three actions:

- Submits the data using the Web service data connection
- Shows a dialog box with a thank-you message
- Closes the form

The template is now ready to use. We can test the template by opening the form, entering some data, and then clicking the Submit button. If we get the message, the information was processed successfully by the Web service.

Publishing a Form to a Windows SharePoint Services Forms Library

In this section, we take on the role of an administrator responsible for deploying an InfoPath template. We'll show how to deploy a template to a SharePoint forms library and make it available through Office Forms Server 2007 for consultants who do not have InfoPath installed on their computers.

Before deploying a template, you have to verify the security level. For the deployment to behave properly, follow these steps to request full trust, using settings on the Security And Trust page of the Form Options dialog box:

- Clear the check box for Automatically Determine Security Level Based On Form's Design.
- Select the Full Trust option.
- Select Sign This Form, and then click the Create Certificate button.
- Select the Administrator certificate.

In the Design task pane, click Design Checker to ensure that the form is compatible with being enabled for Web browsing. You should not see any red icons indicating a compatibility error. You might see one or more blue icons indicating a warning.

Now click Publish Form Template in the Design task pane. A wizard guides you through the steps to publish the template to a SharePoint site or an Office Forms Server 2007 site. You need to specify the URL to the SharePoint server, and then you will create a document library for the form template.

As an administrator of the forms library, you can specify that the InfoPath form always has to be filled in using the browser, even if a user has InfoPath installed on his or her computer, by setting the Opening Documents option mentioned earlier.

Figure 9-7 shows the time sheet form open in the browser after being successfully published to Office Forms Server 2007.

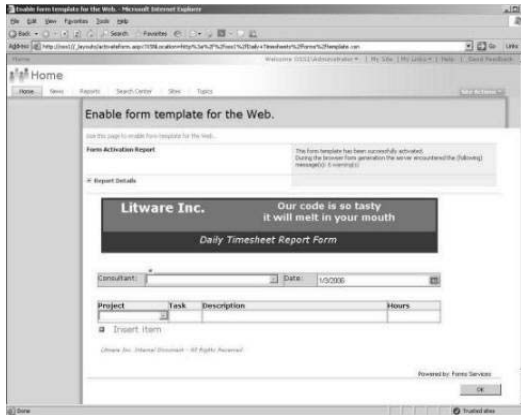


Figure 9-7 The time sheet form rendered in the browser

Embedding an InfoPath Form in an Application

Developers now have the option to embed InfoPath forms within an application. You can work with two types of controls, one that you can use within ASP.NET applications, and the other with Windows applications. In this section, we'll take a look at how to work with an embedded InfoPath control. In the first example, we'll show a Web page that displays a list of submitted time sheets. When a user selects a time sheet in the list, a control provides a read-only preview of the time sheet on the page itself.

To host the Web application we'll work with, we created a virtual directory under the default Web site. The settings for virtual directory permissions included Run Scripts as well as the default options. The other setting we made on the Virtual Directories tab was to select Scripts Only for the Execute Permissions option.

We also turned on session state for Office Forms Server 2007 by opening the Web.config file found at C:\inetpub\wwwroot and removing the comment markup from the *SessionState* entry in the *HttpModule* element, which is shown here:

```
<add name="Session" type="System.Web.SessionState.SessionStateModule"/>
```

Then we set *enableSessionState* to True in the *pages* element in the Web.config file:

```
<pages enableSessionState="true" enableViewState="true"
  enableViewStateMac="true" validateRequest="false" pageParserFilterType=
  "Microsoft.SharePoint.ApplicationRuntime.SPPageParserFilter,
  Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
  PublicKeyToken=71e9bce111e9429c" asyncTimeout="7">
```

We next created a new Web site application in Visual Studio 2005, setting the type to HTTP and choosing C# as the language.

In the designer, we add **Litware Timesheet Viewer** as the title of Default.aspx, and then add a table with two rows and two columns. In the first cell of the first row, we enter **Timesheets**, and then add a list box control to the second row. We named this control *listTimesheets* and set the *AutoPostBack* property to True.

The next step is to right-click in the toolbox, and select Choose Items. Once the Choose Toolbox Items dialog box appears, use the Browse button to load Microsoft.Office.InfoPath.Server.dll from the C:\Program Files\Microsoft Office Server\12.0\Bin folder. Three controls are added to the list, including the *XmlFormView* control.

We can now drop the *XmlFormView* control onto the second cell of the second row of the table. In the properties window, we change the name of the control to *xmlFormViewTimesheets* and set the *Visible* property to False. Figure 9-8 shows a preview of the form in the browser.

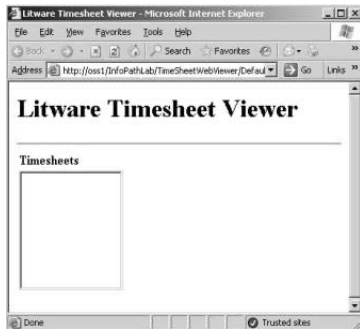


Figure 9-8 An example of the *XmlFormView* control

We next add the code to display all the submitted time sheets in the [http://oss1/Daily Timesheets](http://oss1/DailyTimesheets) forms library. In the code-behind file for Default.aspx, we need to add a reference to Microsoft.SharePoint.dll and then declare the namespace:

```
using Microsoft.SharePoint;
```

Next we create a private procedure that will encapsulate the code that populates the list box.

```
private void PopulateListBox() {}
```

In this procedure, the code we write first establishes a connection to the forms library:

```
SPSite site = new SPSite("http://oss1");
SPWeb web = site.RootWeb;
SPList list = web.Lists["Daily Timesheets"];
```

The next step is to get the time sheet items as a *DataTable* type so that we can bind the data to the *listTimesheets* control.


```

DataTable timesheets = list.Items.GetDataTable();
listTimesheets.DataTextField = "Title";
listTimesheets.DataValueField = "ID";
listTimesheets.DataSource = timesheets;
listTimesheets.DataBind();

```

Now we call this procedure in the *Page_Load* event handler:

```

protected void Page_Load(object sender, EventArgs e)
{
    if(!Page.IsPostBack)
        PopulateListBox();
}

```

Figure 9-9 shows the populated list box when we view the page in the browser.

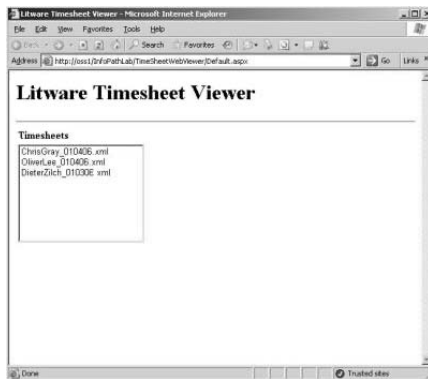


Figure 9-9 The list box populated from the items in the forms library in SharePoint

We now add the code that runs when a user selects a time sheet in the list box. The code renders a read-only view of the time sheet on the form itself in the InfoPath control. To create this view, we handle the *SelectedIndexChanged* event of the *listTimesheets* control. Not a lot of code is required here. We just grab the selected item and pass it to the *xmlFormViewTimesheets* control:

```

protected void listTimesheets_SelectedIndexChanged(object sender, EventArgs e)
{
    string selForm = listTimesheets.SelectedItem.Text;
    xmlFormViewTimesheets.XmlLocation = "http://oss1/Daily Timesheets/" + selForm;
    xmlFormViewTimesheets.Visible = true;
}

```

Figure 9-10 on the next page again shows the application previewed in the browser.

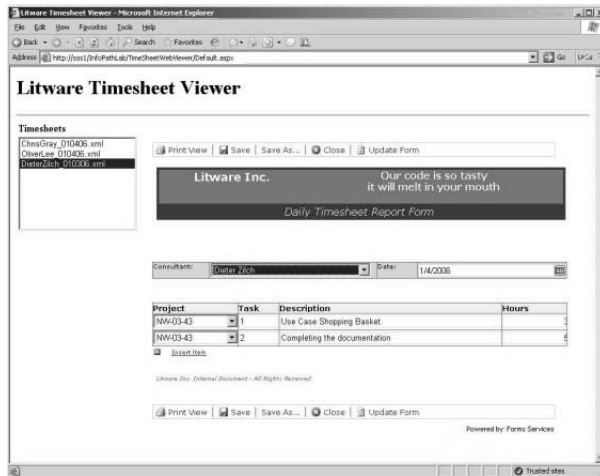


Figure 9-10 A preview of a specific time sheet is rendered in the *XmlFormView* control

Now we'll turn our attention to a Windows application that hosts an InfoPath control that the user can use to fill in a new time sheet report. The application provides assistance to the user by means of a panel that displays project information. We'll use a Visual Studio 2005 Windows application named *TimeSheetWinApp*.

After creating the application, we right-click in the toolbox and select *Choose Items*. In the *Choose Toolbox Items* dialog box, in the .NET components list, we select the item for the form control that is associated with the *Microsoft.Office.InfoPath* namespace. (If this control is not available, it can be added by browsing to *C:\Program Files\Microsoft Office\Microsoft.Office.InfoPath.dll*.)

We drop the form control on the design surface of *form1*, change the name to *formControl-Timesheet* and set the *Dock* property to *Left*. Then we add a splitter control and a panel control to the form as well. For the panel control, we set the *Dock* property to *Fill*.

Using the *New Data Source Wizard* in Visual Studio, we create a data source to an Office Access 2007 database that contains one table, named *ProjectTasks*, with a number of task items per project.

We drag the *ProjectTasks* fields from the *Data Source* pane to the panel on the form, and then add two buttons to the panel: *buttonInsert* with its *Text* property set to *Insert*, and *buttonSave* with its *Text* property set to *Save*.

The first coding task is to display the InfoPath form to be filled out in the control. In the constructor, we add the following line after the call to *InitializeComponent*:

```
formControlTimesheet.NewFromSolution("http://oss1/Daily%20Timesheets/Forms/template.xsn");
```

We can now run the application again and see the time sheet form to be filled in, shown in Figure 9-11.



Figure 9-11 A Windows application that hosts an InfoPath form

Return to the designer after closing the application and double-click the Save button. The control only allows the form to be saved to an absolute path. To handle this, we first need to add a couple of namespace declarations to the code:

```
using Microsoft.Office.InfoPath;
using System.Xml;
using System.Xml.XPath;
```

We construct the name of the file that we will save by concatenating the values for the consultant and the date. The form control exposes an *XMLForm* that provides you with an entry point to the XML that is stored in the form. You can access the XML through the *MainDataSource* property. From then on it is a matter of parsing the XML to construct the file name.

```
XPathNavigator nav =
    formControlTimesheet.XmlForm.MainDataSource.CreateNavigator().SelectSingleNode
        ("/my:timesheet/my:header/my:consultant",
        formControlTimesheet.XmlForm.NamespaceManager);
string file = @"C:\Ascend\Labs\08_InfoPath\Lab\Timesheets\";
file += nav.Value + "_";
nav.MoveToNext(XPathNodeType.Element);
file += nav.Value.Replace("-", "") + ".xml";
MessageBox.Show(file);
```

The purpose of the Insert button is to create a new item in the repeating table using the project and task displayed in the panel. To provide this functionality we will handle the *Click* event of the button.

If a user has not yet added any time sheet items, you can populate the row that already exists, which means you need to check, for example, whether the project field of the first item in the repeating table is empty.

```

XPathNavigator nav =
    FormControlTimesheet.XmlForm.MainDataSource.CreateNavigator().SelectSingleNode
        ("/my:timesheet/my:items/my:item[1]/my:project",
        FormControlTimesheet.XmlForm.NamespaceManager);
if (nav.Value == string.Empty) {
}
else {
}
}

```

If this item is the first element in the repeating table, then we can set the values of the different columns of the item.

```

nav.SetValue(projectTextBox.Text);
nav.MoveNext(XPathNodeType.Element);
nav.SetValue(taskIDTextBox.Text);
nav.MoveNext(XPathNodeType.Element);
nav.SetValue(descriptionTextBox.Text);

```

If this is not the first item, we need to create a row in the repeating table, and then make sure that we have access to the last row element added so that we can modify those values.

```

FormControlTimesheet.XmlForm.CurrentView.ExecuteAction(ActionType.XCollectionInsert,
    "group8_7");

XPathNavigator newNav =
    FormControlTimesheet.XmlForm.MainDataSource.CreateNavigator().SelectSingleNode
        ("/my:timesheet/my:items/my:item[last()]/my:project",
        FormControlTimesheet.XmlForm.NamespaceManager);
newNav.SetValue(projectTextBox.Text);
newNav.MoveNext(XPathNodeType.Element);
newNav.SetValue(taskIDTextBox.Text);
newNav.MoveNext(XPathNodeType.Element);
newNav.SetValue(descriptionTextBox.Text);

```