Kevin Laahs
Emer McKenna
Veli-Matti Vanamo

Microsoft® SharePoint 2007 Technologies

Planning, Design and Implementation

COMMUNICATIONS

Microsoft® SharePoint 2007 Technologies

This page intentionally left blank

Microsoft® SharePoint 2007 Technologies

Planning, Design and Implementation

Kevin Laahs
Emer McKenna
Veli-Matti Vanamo

Amsterdam • Boston • Heidelberg • London • New York • Oxford
Paris • San Diego • San Francisco • Singapore • Sydney • Tokyo

Digital Press is an imprint of Elsevier

Digital Press is an imprint of Elsevier
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
Linacre House, Jordan Hill, Oxford OX2 8DP, UK

Copyright © 2008 by Elsevier, Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, E-mail: permissions@elsevier.com. You may also complete your request online via the Elsevier homepage (http://elsevier.com), by selecting "Support & Contact" then "Copyright and Permission" and then "Obtaining Permissions."

Recognizing the importance of preserving what has been written, Elsevier prints its books on acid-free paper whenever possible.

Library of Congress Cataloging-in-Publication Data
Application Submitted.

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

ISBN: 978-0-12-373616-1

For information on all Digital Press publications
visit our Web site at www.books.elsevier.com

Printed in the United States of America
07 08 09 10 11 12 10 9 8 7 6 5 4 3 2 1

Working together to grow
libraries in developing countries
www.elsevier.com | www.bookaid.org | www.sabre.org
ELSEVIER BOOK AID International Sabre Foundation

Contents

Foreword **xv**

Preface **xvii**

Acknowledgements xviii
Personal Acknowledgments xix

1 2007 Microsoft Office System **1**

1.1 Introducing the 2007 Microsoft Office System 1
1.1.1 Investing in Solution Areas 3
1.1.2 Collaboration 5
1.1.3 Portal 6
1.1.4 Search 6
1.1.5 Enterprise Content Management 7
1.1.6 Business Processes 8
1.1.7 Business Intelligence 9

2 Windows SharePoint Services V3.0 **11**

2.1 A Necessary Platform 11
2.2 Main Architectural Ingredients 13
2.2.1 3-Tier Architecture 13
2.2.2 The Site Framework: Web Applications and Site Collections 15
2.2.3 Site Storage—Configuration and Content Databases 19
2.3 Rendering Sites 22
2.3.1 Master Pages, Page Layouts, and Content Pages 22
2.3.2 Site Definitions, Site Templates, and Features 23
2.3.3 Navigation 26
2.4 Storage 29

2.4.1 Metadata—Introducing Content Types and Site Columns 29
2.4.2 Administration—Folders, Recycle Bin, and AutoCopy 32
2.4.3 Versioning—A Major and Minor Enhancement 34
2.4.4 Search 37
2.5 Security Model 37
2.5.1 Authorization 37
2.5.2 Authentication 42
2.5.3 Granular Permissions 42
2.5.4 Security-Trimmed User Interface 43

3 Collaboration 45

3.1 Wikis, Blogs, and RSS (Really Simple Syndication) 45
3.1.1 Wikis 46
3.1.2 Blogs 51
3.1.3 RSS 53
3.2 Mobile Access 55
3.3 E-Mail and WSS 3.0 58
3.3.1 Outgoing E-Mail 58
3.3.2 Incoming E-Mail—Now We're Talking! 58
3.3.3 Directory Management Service 62

4 Microsoft Office SharePoint Server 2007 69

4.1 Shared Services 69
4.2 MOSS extensions to WSS 3.0 70
4.2.1 Administration Additions 70
4.2.2 MOSS Permissions and Groups 71
4.2.3 MOSS Features 72
4.2.4 MOSS Web Parts 75
4.2.5 MOSS Templates 77
4.3 Business Data Catalog 81
4.3.1 Application Definition File 82
4.3.2 Importing an Application Definition File 85
4.3.3 BDC Web Parts 86
4.3.4 BDC and Lists 87

***5* Enterprise Content Management 89**

5.1 Document Management 90
5.1.1 Compliance 90

5.1.2 Auditing 91
5.1.3 Information Management Policies 93
5.1.4 Rights Management 96
5.2 Records Management 97
5.2.1 Records Repository Site 99
5.2.2 Web Service and Populating the Repository 101
5.2.3 E-mail and Records 103
5.2.4 Holds 105
5.3 Web Content Management 106
5.3.1 Publishing Site Templates 107
5.3.2 Creating Pages and Modifying Content 109
5.3.3 Content Deployment 111

6 Search 115

6.1 A Common Search Engine 116
6.1.1 The Core Engine 116
6.1.2 Goals for the Search Experience 119
6.2 Configuring Search 120
6.2.1 Managing Search 120
6.2.2 Content Sources, Crawl Rules, Site Hit Frequencies 122
6.2.3 Search Scopes 123
6.2.4 Managing Properties 126
6.3 Consuming Search 128
6.3.1 Search Query Syntax 129
6.3.2 Relevance, Best Bets, and Keywords 131
6.3.3 Search Results Page and Search Web Parts 132
6.3.4 Search Tabs and Searching the BDC 137
6.3.5 Making Sense of Searches through Search Reports 140
6.3.6 More than One Search! 141

7 People 143

7.1 The SharePoint User Profile 144
7.1.1 Populating the User Profile 144
7.1.2 User Profile Properties 147
7.1.3 Searching the Profile 149
7.2 Personal Portal—A User's View 152
7.2.1 By Me—My Site 153
7.2.2 SharePoint Sites Web Part 155
7.2.3 Colleague Tracker Web Part 156
7.2.4 About Me—My Profile 156

7.2.5 For Me—Audiences and Personalization Sites 159
7.3 People Picker and Cross Forest Operations 161

8 Business Intelligence 163

8.1 Excel Services 164
8.1.1 Publishing Excel Workbooks 165
8.1.2 Viewing Workbooks in the Browser 166
8.1.3 Interacting with WorkBooks via Excel Web Access 168
8.2 Key Performance Indicators (KPIs) 169
8.2.1 Creating SharePoint KPIs 170
8.2.2 Displaying KPIs 171
8.3 Dashboards and Report Center Templates 171

9 Office and Exchange Integration 175

9.1 Outlook 2007 175
9.1.1 Synchronization Process 176
9.1.2 Lists 177
9.1.3 Libraries 179
9.2 Word 2007 180
9.3 PowerPoint 2007 182
9.4 Excel 2007 183
9.5 Access 2007 183
9.6 InfoPath 2007 184
9.7 My SharePoints—Navigating Sites, Opening, and Saving items 184
9.8 Accessing SharePoint Document Libraries through OWA 2007 185

10 Planning the SharePoint Deployment 189

10.1 Getting to Know the Requirements 189
10.2 Planning SharePoint Features 190
10.2.1 Authentication Providers 190
10.2.2 Enterprise Search Scoping 191
10.2.3 User Profiles and My Sites 192
10.2.4 RSS Feeds 193
10.2.5 Impact of Web Parts 194
10.3 Overview of Deployment Sizing 195
10.3.1 Performance Testing Methodology 196
10.3.2 Estimating Required Throughput for SharePoint 197
10.3.3 Recommended Application Limits 200
10.4 Sizing SharePoint Server 2007 203

10.4.1 Web Front-End Scalability 204
10.4.2 SQL Server Impact 205
10.4.3 Choosing Your Deployment Hardware Topology 206
10.4.4 High Availability Considerations 207
10.4.5 Network Impact 208
10.5 Disk Subsystem Planning 209
10.5.1 Database Sizing 210
10.5.2 Search Index Sizing 210
10.6 Global Deployment Considerations 211
10.6.1 Addressing Multiple Languages 211
10.6.2 Distributed or Regionalized Server Deployment 212
10.7 Preparing for Software Deployment 213
10.7.1 Service Accounts 213
10.7.2 Preparing Your DNS Architecture 215
10.7.3 SQL Server Configuration 216
10.7.4 Quota Management 217
10.8 Planning Backup and Restore 218
10.8.1 Out-of-the-Box Backup Options 218
10.8.2 Third-Party Backup Solutions 218
10.9 Planning Data Migrations 219
10.9.1 When Company Structures Change 219
10.9.2 Preparing for Cross-Platform Migrations 220
10.9.3 Addressing Functionality Changes 221
10.10 Planning Support for SharePoint 222
10.10.1 Planning a Support Strategy 222
10.10.2 Planning an Operational Model 224
10.10.3 Planning Service Level Agreements 226
10.10.4 Expected Support Call Volume 228
10.10.5 Staffing an Operational Team 229
10.10.6 Pathway to Production 230
10.10.7 Active Monitoring 231
10.10.8 Passive Analyzing 232
10.11 Performance Monitoring 232
10.11.1 Monitoring Strategy 233
10.11.2 Identifying Future Bottlenecks 234

11 Introduction to SharePoint Development 237

11.1 Development Architecture Overview 237
11.2 Microsoft Office SharePoint Designer 2007 243
11.3 Microsoft Visual Studio 2005 249
11.4 Windows SharePoint Services 3.0 Solution Framework 253

11.4.1 Solution File Composition 258
11.4.2 Building a Solution File using makecab.exe 259
11.4.3 Building a Solution File Using Visual Studio CAB Project Template 261
11.4.4 Building a Solution File Using Visual Studio Extensions for WSS 3.0 263

12 SharePoint Programmability 267

12.1 Custom Web Pages and Web Applications in SharePoint 267
12.1.1 Custom Web Pages 268
12.1.2 Best Practices for Creating Custom Web Pages 270
12.1.3 Accessing the SharePoint Object Model with Inline Code 271
12.1.4 Applying the Layouts Branding 273
12.1.5 Inheriting Master Page from Current Site 274
12.1.6 Storing Code behind Custom Web Pages 276
12.1.7 Create the Assembly File 278
12.1.8 Create the Web Form 280
12.1.9 Custom Web Applications 280
12.2 The Object Model 284
12.2.1 WSS 3.0 Object Model 284
12.2.2 Retrieving SharePoint Data 286
12.2.3 Adding Content to SharePoint 288
12.2.4 Removing Content from SharePoint 292
12.2.5 Using Elevated Privileges 294
12.2.6 SharePoint Server 2007 Object Model 296
12.3 Web Services 299
12.3.1 Working with Web Services 303

13 Building and Deploying Web Parts 309

13.1 Introduction 309
13.2 What is a Web Part? 309
13.2.1 Creating a Web Part Page 311
13.2.2 Adding, Modifying, and Arranging Web Parts 313
13.3 Microsoft Office SharePoint Server 2007 Web Parts 316
13.3.1 SharePoint Designer Data View Web Part 317
13.3.2 Data Source Library 318
13.3.3 Linked Data Sources 320
13.3.4 Creating a Data View with a Linked Data Source 322
13.3.5 SharePoint Server 2007 Content Query Web Part 325
13.3.6 Exercise Preparation 338

13.3.7 Content Query Web Part Exercise 338
13.4 Creating Custom Web Parts 341
13.4.1 Getting Started 342
13.4.2 Coding the Web Parts 347
13.4.3 Creating the Second Web Part 350
13.4.4 Deploying the Web Part 351
13.5 Coding Web Parts 353
13.5.1 Web Part Rendering Process 354
13.5.2 Web Part Properties and Methods 355
13.5.3 Implementing Child Controls 358
13.5.4 Customizing the Web Part Menu 361
13.5.5 Tool Panes and Editor Parts 362
13.6 Connecting Web Parts 367
13.6.1 Connection Interfaces 368
13.7 Debugging Web Parts 374
13.8 Packaging and Deploying Web Parts 375
13.8.1 Packaging the Web Part 376
13.8.2 Step 2: Package the solution 376

14 Branding your SharePoint 2007 Sites 383

14.1 What is Branding? 383
14.2 Branding Pain Points in SharePoint Portal Server 2003 384
14.3 SharePoint 2007 Enhancements 387
14.3.1 Enterprise Content Management 388
14.3.2 Master Pages 391
14.3.3 SharePoint Master Pages 392
14.3.4 Application.master—the Nemesis of Branding! 399
14.3.5 Page Layouts 400
14.3.6 Publishing Page Rendering 402
14.3.7 Navigation 402
14.4 Customizing the Color Scheme 406
14.4.1 Creating Custom Style Sheets 409
14.5 Themes 411
14.5.1 Creating a Custom Theme 415
14.5.2 To Theme or Not to Theme 418
14.6 Site Title, Description, and Icon 420
14.7 Basic Branding Exercises 421
14.7.1 Browser Customizations 422
14.7.2 Branding the entire site (continues from previous exercise) 423

15 Site Definitions and Templates 429

15.1 Defining Site Definitions, Site Templates, and Custom Templates 429
15.1.1 Site Definitions 431
15.1.2 Site Templates 435
15.1.3 Custom Site Definitions and Custom Site Templates 441
15.1.4 Feature Stapling 441
15.1.5 .STP Files 444
15.1.6 Portal Site Definitions (Also Known as Virtual Site Definitions) 446
15.1.7 Application Templates 447
15.2 Dissecting ONET.XML 447
15.2.1 The Project Element 448
15.2.2 NavBars 449
15.2.3 Configurations Element 450
15.3 Site Definition Solution Files 454
15.4 Site Definition Branding Exercise 456

16 Features 463

16.1 What is a Feature? 463
16.2 SharePoint 2007 Features 464
16.3 Feature Architecture 466
16.3.1 Element Manifests 471
16.3.2 Defining Modules 473
16.3.3 Defining Custom Actions 475
16.3.4 Feature Properties 479
16.3.5 Activation Dependencies 479
16.4 Feature Receivers 482
16.5 Feature Lifecycle 483
16.5.1 Creating a Feature 485
16.5.2 Installing a Feature 486
16.5.3 Activating a Feature 487
16.5.4 Deactivating a Feature 490
16.5.5 Uninstalling a Feature 491
16.6 Feature Stapling 492
16.7 Deploying Features using the Solution Framework 493
16.8 Branding Using Features Exercise 496
16.8.1 Branding Scenario 498
16.8.2 Create the Feature Receiver 506
16.8.3 Create the Branding Feature Set 507

17 Events 513

17.1 What is an Event? 513
17.1.1 Event Receivers 515
17.1.2 SPWebEventReceiver 516
17.1.3 SPItemEventReceiver 520
17.1.4 SPListEventReceiver 528
17.1.5 Additional Event Receivers 531
17.1.6 Support for WSS 2.0 Events 532
17.2 Working with Events 533
17.2.1 Creating an Event Handler 533
17.2.2 Deploying an Event Handler 534
17.2.3 Registering an Event Handler 534
17.2.4 Registering Event Handlers using Features 536
17.2.5 Registering Event Handlers using Content Types 540
17.2.6 Registering Event Handlers using the Object Model 543
17.2.7 Removing Event Handler Registrations using the Object Model 543
17.3 Working with Document Libraries—Event Exercise 545
17.3.1 Create and Deploy the Event Handler 548
17.3.2 Register the Event Handler 551
17.3.3 Testing the Event Handler 553

18 InfoPath Forms Services and Workflow 557

18.1 What is InfoPath Forms Services? 557
18.1.1 Microsoft Office Forms Server 2007 559
18.2 Configuring InfoPath Forms Services 560
18.3 Introduction to InfoPath 2007 570
18.3.1 What is a Form Template? 570
18.3.2 Form Template Designer 571
18.3.3 Template Parts 573
18.4 Data Connections 574
18.5 Creating a Browser-Enabled Form Template 578
18.6 Deploying and Managing InfoPath Forms 585
18.6.1 User Form Template Deployment 586
18.6.2 Administrator-Deployed Form Templates 596
18.7 Document Information Panel 603
18.8 Adding Code to a Form Template 607
18.9 Workflow and SharePoint Server 2007 616
18.9.1 Windows Workflow Foundation 618
18.9.2 Workflow in Windows SharePoint Services 619

18.9.3 Workflow in SharePoint Server 2007 623
18.10 Workflow Authoring 623
18.10.1 Authoring Workflows in Windows SharePoint Services 3.0 624
18.10.2 Authoring Workflows using SharePoint Designer 2007 628
18.10.3 Authoring Workflows in SharePoint Server 2007 630
18.11 Integrating InfoPath Forms with Workflow 632
18.12 Conclusion 641

Index 643

Foreword

In the fall of 2006, Microsoft unleashed the third generation of Web-based capabilities for information workers: Microsoft Office SharePoint Server 2007 and Windows SharePoint Services 3.0. The SharePoint technologies deliver, to put it bluntly, an avalanche of features and opportunities for customers and partners. Those of us working on SharePoint Products and Technologies really think of 2007 as a watershed release that stands on the shoulders of the core infrastructure to deliver real value across a broad spectrum of scenarios, including collaboration, enterprise content management, search, portals, business process, business intelligence, and the core development platform. This wave of technology not only lets information workers do things that they've never done before, but enables developers and IT organizations to reach further than ever in servicing their customers.

The core promise of SharePoint Products and Technologies is that this great breadth of capabilities is delivered within one framework: a consistent set of tools and metaphors for end users, developers and administrators that scale across scenarios of differing scope and complexity. For example, end users can learn how to interact with one set of document library tools that are consistently reused across their personal document libraries, their team's shared authoring space, their divisional document repository, and their intranet and Internet publishing sites. As another example, developers can learn one technology for building and deploying rich solutions across the same spectrum of places, whether targeting towards executive KPI scorecards or company-wide surveys.

Such a wave of capability naturally means that there is a lot to do, a lot to know, and a lot to learn. It is very exciting to see knowledgeable experts jump in to help; Kevin, Emer, and Veli are just such a group of smart and passionate experts. I have had the privilege of working with them personally for many years and have a great level of respect and appreciation for their depth of knowledge, excitement for technology, and commitment to teaching and learning. They have worked with SharePoint technologies since the very first release, and this book represents another round of comprehensive

analysis and understanding, coupled with detailed and patient explanation. The breadth of material provides great coverage across all of the SharePoint capabilities, and the authors drill into rich practical detail by describing real-world scenarios. This book stands as both a useful overview for initial learning and an essential detailed reference manual for specific problem-solving down the road.

If you are new to SharePoint Products and Technologies, then you have selected an excellent resource to assist you in coming up to speed on this new technology. If you are already familiar with previous releases, this book will definitely introduce you to and guide you through the newest innovations in the third generation of the technology. At the end of the day, this is all about helping people be more productive. For the users of SharePoint technology who find success in increased productivity, I am glad to have helped. For the administrators and developers who leverage SharePoint technology to deliver this success, I am grateful for the opportunity to have played a part. For champions of technology like Kevin, Emer, and Veli, I am simply honored at the passion and expertise that they demonstrate.

Rob Lefferts
Principal Group Program Manager
Windows SharePoint Services
Microsoft Corporation

Preface

When we approached Rob Lefferts to write the foreword for our third book about SharePoint Technologies, we reflected on how long we had known each other. Rob first met us during our preparation for Compaq's SharePoint Portal Server 2001 Academy, and he delivered the keynote for our first event in early 2001. So we've known each other for nigh on seven years now.

More than seven years ago, Microsoft had some very grand plans for a product that was code-named *Tahoe*. According to the marketing hype surrounding what became SharePoint Portal Server 2001, it was to be all things to all people. But it wasn't. And there were some very good reasons for that, because, just like a good scotch, it takes time to mature. Maturity comes from the experience of learning and doing over time, and SharePoint is no different in this regard. The experiences of the 2001 and 2003 releases of the technology have helped shape the 2007 release. And it is a release that, from a pure feature point of view, is much closer to the original goals of Tahoe.

The third generation of SharePoint Technologies targets six major solution areas and leverages core infrastructure features delivered in the Windows Server 2003 operating system. In addition to the significant enhancement of collaboration, portal, and search, we see a focus on business intelligence, business process, and enterprise content management. All of these solution areas integrate exceptionally well with each other and with the core platform services provided by Windows SharePoint Services V3.0.

All of us work in HP Services and are involved with SharePoint in many different ways—from consulting with our customers to presenting at industry conferences to working on our own internal knowledge systems, which heavily leverage SharePoint Technologies. Our experiences working with SharePoint and its complementary products have helped us assimilate its value and shape the content for this book, and we certainly hope that our experiences go a long way toward jump-starting yours.

So what does the book contain? Well, certainly not everything, and we apologize if the area that you are most interested in is not covered. What we

have tried to do is to first articulate what the major features are; how they help integrate people, processes, and information; and the subsequent value they can bring to your organization. We tend to view the functionality from a position of its practicality in large enterprises, and many of our opinions reflect that bias. We believe we give an honest view of both the good things and the things that just might need a little more maturing before they can live happily in this space. We then explain the important considerations for when you plan your deployment and, subsequently, how you can build upon the base features to customize and extend the platform for your specific business needs.

SharePoint has come a long way in the last seven years, and Microsoft has done a great job of listening to and acting upon the experiences of their customers and partners. So for those of you who currently use SharePoint, there should be no danger of you having a seven-year itch—and for those of you that have not yet ventured into this technolog, you can rest assured that a mature platform awaits you should you wish to hop on board.

Kevin, Emer, and Veli

Acknowledgements

A feeling of community is essential for people's experiences to be leveraged to the fullest. Only when you feel a true part of a community do you contribute to as well as consume the community experience. In HP, we are fortunate to have such a community, and we'd therefore like to thank everyone who is a part of it.

Our community extends outside of HP to our customers and partners and obviously in the SharePoint space to Microsoft. We have many great friends at Microsoft, and we call them friends due to the strength of the working relationships we have with them and the fun we have together on the occasions we get to meet up in person. Thanks to Rob Lefferts for writing the foreword and to everyone else we know in the SharePoint and Exchange space—you know who you are!

There is a major community in the industry who should not go unmentioned. In this day and age of information ubiquity, all of you who take the time to share your experiences via blogs, wikis, and so forth have contributed to our knowledge, and for that, we thank you.

Finally, we would like to acknowledge the plethora of people who have shown us much patience during the creation of this book; it was greatly appreciated. We hope the end result is worth any frustration that we caused!

Personal Acknowledgments

All of you know me too well. You know that you can't trust me when I say "never again." However, I know I can trust you to support me when I do break my word, so thanks to everyone for that. Thanks also to Emer and Veli for persevering and getting this book done, especially through the hardest of times. And to my wife, Wendy, and my children, Jenny and Euan—as usual, you didn't complain with about the late nights and lost weekends, so thanks; it won't happen again. ☺

—Kevin

Writing a book shares some parallels to the childbirth experience, in that as you are going through the actual labor, you often find yourself saying (or in some cases screaming), "Never again!" and you don't find yourself particularly fond of the folks yelling at you to push. However, once the labor part is over and you are holding your little bundle of joy, or finished product in book terms, your brain *immediately* starts playing tricks on you, making you think "That wasn't so bad after all." Such is life; without this distortion of memory, the human race would most likely be extinct, and bookshelves would probably be considerably lighter. So, without further ado, let the acknowledgements begin.

First off, I would like to thank Kevin for *patiently* waiting on me to finish my section, even though it was months after his work was complete. Thanks to Veli for finding the motivation to write despite his heartache. Thanks to Alan Rose for handling the production of the book and being as wonderful to work with as ever. Thanks to Paul White and Frank Curran, two wonderful teachers who are a big part of the reason I am in this industry at all. Thanks to my mum, dad, and sisters for keeping me motivated and reminding me of all the good times ahead. Thanks to Starbucks for keeping me awake in the wee hours of the morning; my only regret is that I didn't buy shares in the company before I started writing.

Regardless of my opening statement, childbirth is and always will be a miraculous and incredibly satisfying experience comparable to none. Everyday, I continue to be overwhelmed with the joy and sense of purpose that my two beautiful little girls bring to my life; thank you, Caoimhe and Niamh, for always making my heart sing. And last, but by no means least, I would like to thank my husband Michael for being so supportive and keeping me grounded during this process; you are and always will be my rock! I love you.

—Emer

First and foremost, I want to extend my thanks to Kevin and Emer on persevering with my ever-moving dates and changing content. You guys never gave up and helped me in every way possible, so thanks for letting me share this book with you once again.

Thank you to my wife Audrey, for her understanding and hard work during the long evenings and sleepless nights. Thanks to my family for motivation and encouragement throughout the years.

This book found me in between two personal life events: the birth of our first son and the unexpected departure of my father. As in a book, in life there is a beginning and an ending. And so I wish to welcome my son Christian Kullervo and say farewell to my father Kullervo.

—Veli-Matti

1

2007 Microsoft Office System

1.1 Introducing the 2007 Microsoft Office System

And so to our third book that has the word "SharePoint" in its title. Our first dealt solely with the product called Microsoft SharePoint Portal Server 2001 (SPS 2001). In the introduction to the second book, we explained how we were not going to be discussing merely Microsoft Office SharePoint Portal Server 2003 (SPS 2003) but also the wider focus of Microsoft SharePoint Products and Technologies—the umbrella name for those offerings from Microsoft that facilitate collaboration. That book essentially focused on two products—Windows SharePoint Services (WSS 2.0) and SPS 2003, with WSS V2.0 being the foundation upon which SPS 2003 was built.

So where are we now? You will still hear the phrase "Microsoft SharePoint Products and Technologies," and there are still multiple products that fit under this name, but more and more the industry just refers to the whole gamut as "SharePoint," clarifying individual products as and when required. This book, however, will primarily focus on Windows SharePoint Services V3.0, Microsoft Office SharePoint Server 2007, and complementary offerings such as SharePoint Designer 2007. I guess you'd think that this time we are merely talking about an update to the products we covered in our second book? But take a closer look at the product names…notice anything missing?

Hopefully you've noted that the word "Portal" no longer appears! Why not? The focus of SharePoint has expanded significantly in the interim years to meet the many different ways in which people communicate and collaborate, and much more than a portal is required to meet ever-changing business requirements. Today, SharePoint targets much more than just collaboration between groups of people. Rather, SharePoint targets connecting people, process, and information across organizational boundaries—a theme that not only permeates the 2007 Microsoft Office System but other Wave 12 products, such as Exchange Server 2007 and the recently released Office Communications Server 2007. The term "Wave 12" was used during development of

these products, as their internal development version was 12 and they were all due to hit the market around the same time.

As mentioned, the main focus of this book is Windows SharePoint Services V3.0 and Microsoft Office SharePoint Server 2007. You will hear and read numerous terms in the industry that describe these products, so to avoid any ambiguity, we will use the following naming conventions:

1. WSS 3.0 will refer to Windows SharePoint Services V3.0. In those instances when we need to distinguish between this and the previous version, we will explicitly say WSS 2.0.
2. MOSS, SharePoint Server 2007 and SharePoint 2007 will all be used to explicitly refer to Microsoft Office SharePoint Server 2007. In those instances where we need to refer to the previous version(s), we will use SPS 2003 and SPS 2001.
3. SharePoint will be used in those instances where the product we are talking about is irrelevant, i.e., the feature will generally be in both products or not be product specific at all. In other words, SharePoint will essentially refer to the global term SharePoint Products and Technologies.

SharePoint Server 2007 is part of the 2007 Microsoft Office System—a suite of desktop and server products designed to work together and streamline the way people work. A discussion on SharePoint would not be complete without considering the complementary products in the suite. For example, Outlook 2007 can act as a smart client to SharePoint providing both read and write access to information held in SharePoint lists. Although WSS 3.0 is not technically part of the 2007 Microsoft Office, since it is engineered under the Windows product group, it is the platform upon which SharePoint Server 2007 is built and so is a fundamental part of any SharePoint solution. The products that are currently available within the 2007 Microsoft Office System are as follows (more details can be found online at www.microsoft.com/office):

Desktop Programs

Access

Accounting

Communicator

Excel

FrontPage

Groove
InfoPath
OneNote
Outlook
PowerPoint
Project
Publisher
SharePoint Designer
Visio
Word

Servers
Forms Server
Groove Server
Live Communications Server
PerformancePoint Server
Project Portfolio Server
Project Server
SharePoint Server
SharePoint Server Search

1.1.1 Investing in Solution Areas

One very pleasing aspect of the whole Wave 12 development cycle was the consistent Microsoft discussion centered around Investment Areas rather than individual product offerings. The result of this attitude is the ability to deliver more tightly integrated solutions without having to jump through hoops to do so. By focusing on overall solution areas, Microsoft has come up with a platform (in WSS 3.0) that provides relevant and essential services to allow such solutions to become a reality.

Consider SPS 2003 in terms of the solution areas that it addresses: collaboration, portal and search, as depicted in Figure 1.1. WSS 2.0 provided the collaboration foundation upon which SPS 2003 added enterprise search and portal capabilities. Although SPS 2003 relied on WSS 2.0, the integration between them was lacking in several areas and there was a sense that they

Figure 1.1 *SPS 2003 solution areas.*

were not quite together. This sense is removed with WSS 3.0 and SharePoint Server 2007.

With SharePoint Server 2007, we see a much more complete picture with the introduction of other solution areas and a rich set of platform services, as depicted in Figure 1.2. MOSS offers an integrated suite of server capabilities for enterprise search, content management, business process facilitation, simplified information sharing, and enhanced business insight.

Along with the core platform services provided by WSS 3.0 and some operating system services provided by Windows 2003, such as ASP.NET and Windows Workflow Foundation, these solution areas complement the existing ones and together provide a compelling platform for connecting people, process and information across boundaries. This diagram shapes the rest of Part I of this book, in that we will cover each of the core platform services provided in WSS 3.0, followed by a discussion on each solution area. Before we do that, though, let's introduce each solution area here to give a flavor of the types of functionality you can expect to see.

Figure 1.2 *SharePoint 2007 solution areas.*

1.1.2 Collaboration

Collaboration focuses on enabling us to work smarter with others and to ensure that the important collateral we need to work with is at our fingertips while taking into account the different working environments we find ourselves in—for example, being able to seamlessly work on team documents while disconnected from the network or being able to contribute to team calendars as easily as you can to your own personal calendar.

Sharing information easily is key to successful collaboration, and users are able to publish information using modern techniques such as Web logs (blogs) and wikis (see http://en.wikipedia.org/wiki/Wiki for information on what a wiki is) as well as to contribute information to discussions/lists, etc. via e-mail. Similarly, consuming such information is possible using RSS readers and aggregators (Outlook 2007 provides an inbuilt RSS aggregator).

When collaborating in small teams, there is often the need to allocate and monitor a set of tasks and have simple visibility into the current progress of these tasks. A full-blown project management system is usually overkill for

such small teams, so features that enable "lite" project management are also available, with the ability to have common project management type views such as Gaant charts.

Of course e-mail is probably the most popular way of communicating with others, and we see the introduction of capabilities in this area such as being able to keep in touch with team members through e-mail, contributing to lists and libraries through e-mail, and using team sites as archives for discussion lists.

1.1.3 Portal

Portal is an area that is primarily concerned with making sure the right information gets to the right people. Actually, it's a little more than that, as it also focuses on ensuring that you can find the people that are instrumental to the information.

Being able to quickly provision Intranet portal sites and place relevant information on them for consumers is therefore a key solution area. Features that we are used to, such as My Sites and the Site Directory, are significantly enhanced, as is the delivery of personalized information through audience targeting and the ability to have roll-up views of important business details.

Social networking—the ability to find out who knows who and who knows what—was supposed to be shipped as an add-on to SharePoint 2007, but unfortunately it missed the cut. This is an important area for organizations to fully exploit their intellectual capital so hopefully we will see it out soon (perhaps even before this book gets published!).

1.1.4 Search

Almost everyone can recall growing up at home, "Where are my shoes?" would be quickly answered, "Where you left them!" Of course in this day and age of information ubiquity, it is nearly impossible to remember where you "left" information. Some people claim that if you can organize information, then you can find it by merely browsing to it, but anyone who has spent a small amount of time trying to decide on a taxonomy that can suit the needs of an unknown user base (e.g., Internet users) would quickly agree that a search engine is a necessity.

As an example of why it is difficult to build a browsing taxonomy to suit everyone, consider going into a music store to buy a CD. Some stores arrange their stock alphabetically, whereas others arrange it by genre. Whatever you are looking for, you, the end user, must adapt your method of finding something to match the method those who came up with the taxonomy in the first place. Shouldn't it be the other way around?

A robust, reliable, and efficient search engine is therefore a requirement for users to be able to find relevant information quickly and easily. And the engine should be capable of finding information regardless of its source, be it in a local portal site, an Internet site, or a line of business application. The SharePoint 2007 Business Data Catalog (BDC) is a key enabler that can help unlock data held in backend enterprise/business applications. Once unlocked, that data can contribute to many of the solution areas that SharePoint 2007 addresses.

Generating relevant results every time is a strict requirement of a search engine, and this goal can be achieved assuming end-users supply enough information about the type of thing they are looking for in the first place. Relevance is tuned for enterprise content, with the best results across structured and unstructured data sources determined by a rich and broad range of factors.

Search extends across most functional areas and is integrated with the collaboration, portal, content management, business, process and business intelligence areas within SharePoint Server 2007. It can also be integrated with other 2007 Office system products (for example, via the Research pane in Word 2007) to help users easily find, use, and share information and increase productivity.

1.1.5 Enterprise Content Management

The solution area targeted in SharePoint Server 2007 is Enterprise content management, meaning much more than just managing documents in a library. Indeed, content management is a very broad topic that can range from the simple authoring and publishing of a document, to retaining information for regulatory compliance, to the creation, management, and deployment of Web content that makes up internal and Internet-facing (and typically common branded) Web sites. Many services must be available to cover the different scenarios, including robust workflow, granular security, and protecting content usage.

Functionality such as per-item level permissions, recycle bins, full major/minor versioning, and content types for the creation and revealing of metadata all combine to deliver a unified approach to managing content across all Office 2007 client and server products. For example, the user can manage documents and take them offline using Outlook 2007, be alerted via tasks and e-mail when content requires your attention and have content automatically submitted to records management archives. The information panel in Office 2007 client applications allows you to find out more about the content you are working with by directly revealing its metadata in an easy-to-use interface.

As an example of the scope of enterprise content management, consider that SharePoint 2007 includes integrated records management capabilities that give organizations the ability to store and protect business records in their final state. Expiration policies applied to these records help ensure that the records are retained for the appropriate time period to comply with regulations or corporate business polices, thereby mitigating legal risk to the organization. Features such as audit trails can be used to prove to internal and external auditors that records have been retained appropriately.

All the Web masters out there may be wondering whether the Web content management piece conflicts with Content Management Server (CMS) as it exists today. The answer is no, as CMS is no more and its functionality has been included in SharePoint Server 2007! Indeed, Figure 1.3 is a good representation of the evolution of SharePoint Products and Technologies and shows how CMS has been brought into the fold along with other legacy SharePoint products.

1.1.6 Business Processes

Forms lie at the heart of many business processes—and it is typically an arduous task to fill in forms correctly so that the information gathered from the user is actually what is required for the business process at hand. More often than not, end users are confused as to why they are being asked for certain types of information or what format the desired response should take, and the designer of the form is not always on hand to help explain. This whole area therefore focuses on making it much easier to design robust forms, and ensure that they are readily available, easy to complete, and manageable throughout their life cycle. The core component that lies at the heart of this is the InfoPath Forms Service—basically InfoPath as it exists today, but running server-side and consumable by a browser, rather than having to be installed locally.

Forms need to interact with data from many different places if they are to provide valid information, and users should not have to jump through hoops to ensure validity. Therefore, seamless integration with back-end line of business applications is essential so that, for example, customer ids and product ids can be chosen from drop-down lists to ensure validity. As mentioned previously, the BDC contributes here by unlocking back-end business data.

Most business processes involve multiple people; therefore, a robust workflow engine is a crucial ingredient for success. Harnessing Windows Workflow Foundation allows built-in or custom templates to be used to automate approval, review, and archiving processes.

1.1.7 Business Intelligence

Business intelligence is all about making smarter use of your business information. Analyzing, reporting, managing, and publishing results, as well as keeping an eye on key performance indicators (KPI), is crucial to determining how your business is performing. KPIs can be exposed through team and portal sites and can monitor the performance of manually entered criteria, data in SharePoint lists, Excel workbooks, or SQL Server Analysis Servers.

Excel Services, part of SharePoint Server 2007, are a key component when providing solutions in this area. They basically extend the capabilities of the desktop Excel 2007 by allowing the broad sharing of spreadsheets, improved manageability and security, and the ability to reuse spreadsheet models using a scalable server-based calculation service and interactive Web-based user interface.

It is also easy to build dashboards and report centers that assemble and display business information from disparate sources via the many built-in Web Parts. Again, the BDC comes into play here to unlock that back-end data.

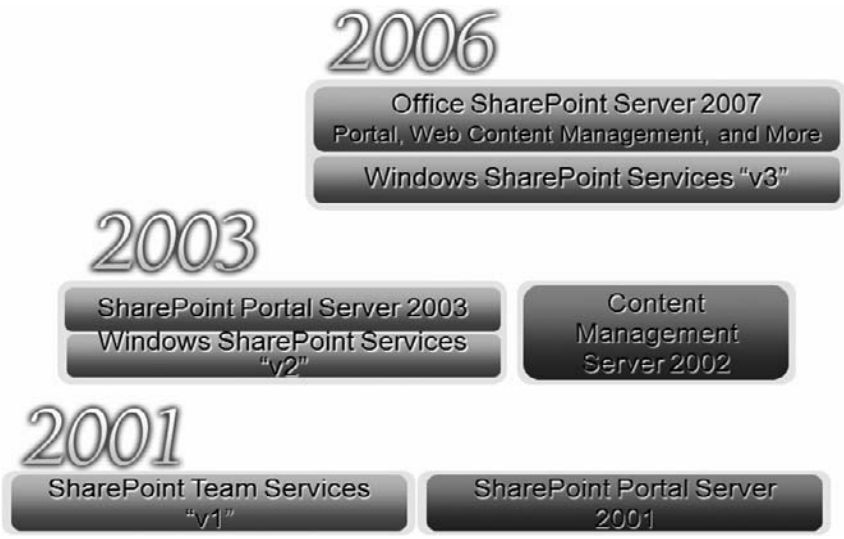


Figure 1.3 *Technology evolution in SharePoint.*

This page intentionally left blank

2

Windows SharePoint Services V3.0

2.1 A Necessary Platform

As mentioned in Chapter 1, WSS 2.0 provided the foundation upon which SPS 2003 was built. It essentially took the file share paradigm to a new level and empowered users by allowing them to collaborate and share documents, tasks, events and more. It was both an enabling technology and a team collaboration solution. Indeed, WSS 2.0 could be installed purely for its team collaboration solution and there would be no need to venture anywhere near SPS 2003.

The enabling technology was leveraged by SPS 2003 to deliver a portal solution that worked but was not perfect, as there were some incompatibilities between what WSS 2.0 provided and how SPS 2003 used it. As an example of what we mean by not perfect, consider the fact that the security model in a pure team site was not the same as that in a portal area even though they both used the same underlying site model that WSS 2.0 delivered. What is this enabling technology? We are basically referring to the Web site framework and flexible storage model that empowered end-users to create and manage team sites. Content stored in the sites, in terms of document libraries and lists, was primarily accessed via a browser and rendered using Web Part pages essentially ASP.NET pages with zones that housed Web Parts. Web Parts themselves are based on ASP.NET form controls, and they typically display some content that is stored in a library or list. Web services and a server-based API also allowed other forms of access to the site content, such as from Office 2003 and the Windows Explorer.

SharePoint, therefore, should be viewed as two parts—a platform and the solutions built upon this platform. This was the case with WSS 2.0 and SPS 2003, but, to meet the requirements of the expanded solution areas in SharePoint 2007, a much richer platform with extended services was required. And this is where WSS 3.0 comes into the picture. As we will discover, it delivers core services that permeate throughout the six solution areas we talked of earlier. Additionally, WSS 3.0 on its own provides the

collaboration solution and, unlike before, there are no incompatibilities with the core services and the solutions that are ultimately implemented using them.

The first W in WSS stands for Windows, since it is actually part of the Windows Server 2003 operating system and included in your licence. Initially, WSS was only available as a Web download, but it will become a part of the base kit and be available for installing through the Configure Your Server/Manage Server wizards. Therefore it is not intended to be solely used by SharePoint 2007, and you can expect other Microsoft and third-party applications to fully exploit the core platform services. The components in the WSS 3.0 platform, such as the Web site framework, security model, list, and document storage—plus its support for ASP.NET 2.0 and Windows Workflow Foundation—make it much easier for developers to build, deploy, and integrate rich applications rather than rolling their own framework from scratch. SharePoint 2007, with its six application areas, is a great example of this. Imagine how complex it would have been to build these applications without a common set of core services and framework?

WSS 3.0 therefore has a lot to live up to, and we believe it has taken this responsibility fully on board and delivered a very compelling platform. Figure 2.1 shows the platform services and collaboration solution that come with a default WSS 3.0 installation. Note the platform and operating system services. In this chapter we will touch on all but workflow.

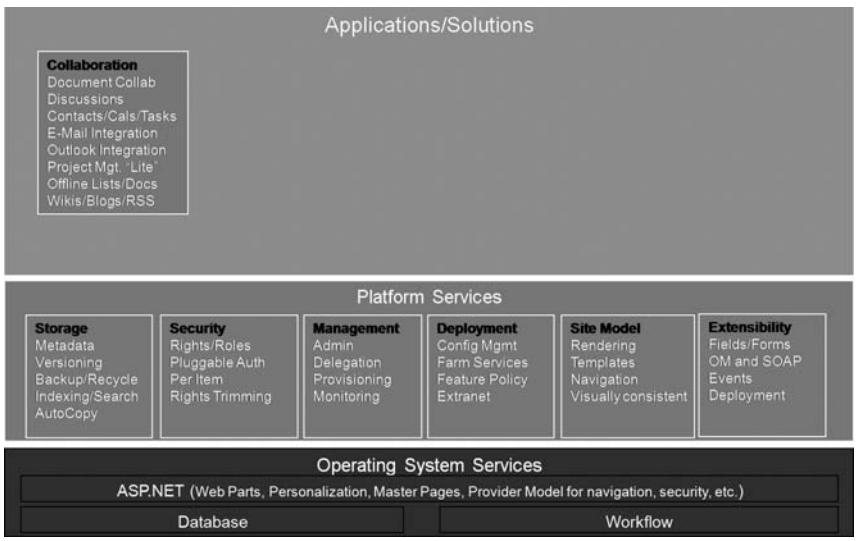


Figure 2.1
WSS 3.0 platform services.

It's interesting to look at some of the services WSS 3.0 provides and see how different solutions use these services. Take versioning, for example. This

is the ability to keep major and minor versions of list and library items. In the Collaboration solution area, a wiki team site uses this capability automatically to highlight the changes that have been made to a wiki page as shown in Figure 2.2. This same capability is harnessed by Web content management to edit a draft copy of a Web page that will ultimately be approved and deployed into an Internet facing Web site as shown in Figure 2.3. Both of these are just items stored in a document library, but the application itself uses the feature to deliver specific functionality according to the task at hand. As we cover the various solutions throughout the rest of this book, we'll try and highlight the different ways in which these core services are used.

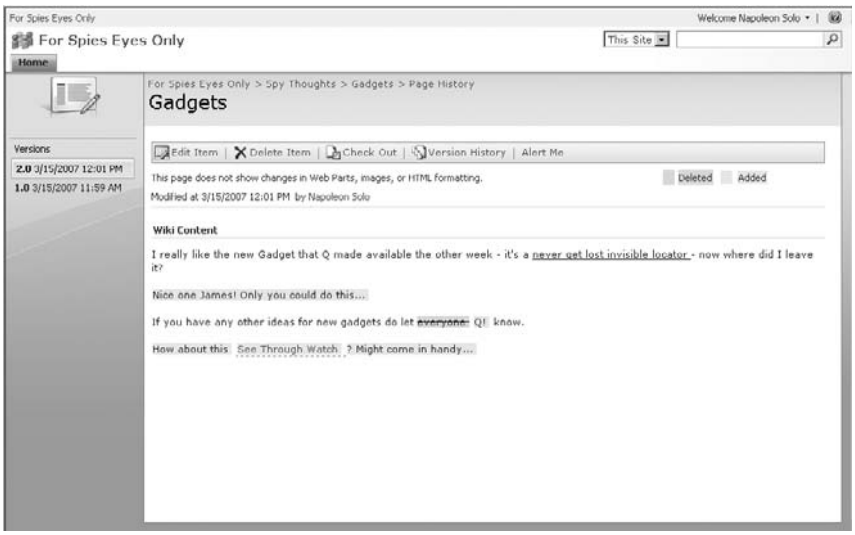


Figure 2.2 *Versioning on a wiki page.*

Planning deployments is covered in Chapter 10. We will take a look here at storage, security, and site model services. Before we do, though, you need to have an appreciation of the major architectural ingredients of any WSS 3.0 deployment in order to understand some core concepts.

2.2 Main Architectural Ingredients

2.2.1 3-Tier Architecture

Like any good Web-based application, WSS 3.0 is a 3-tier application that can be scaled out and up to support company-specific needs, from small organizations to the largest of enterprises. From a logical point of view, each tier provides the following:

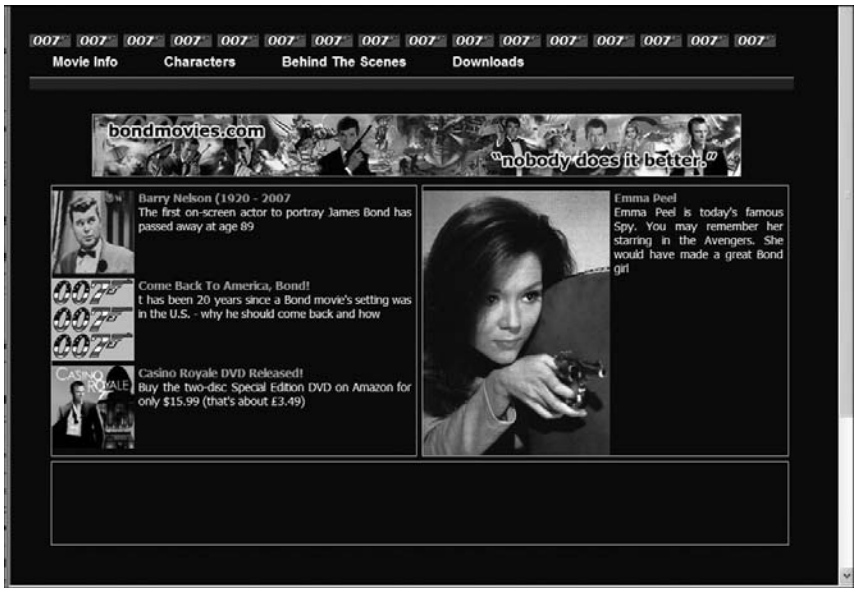


Figure 2.3 *Publishing Site and versioning.*

- *Tier 1*: Commonly known as the *Web Front End* (WFE) tier, this tier is responsible for receiving user requests (typically from a browser or client application such as Office 2007), soliciting data from the database or other servers as needed, and returning the desired information to the user. The format of the returned data depends on the incoming request but it will typically either be via a Web Part page, an XML response to a Web service request, or a physical download of an item from a library.
- *Tier 2*: Commonly known as the *Application* tier, this tier is responsible for dynamically generating content according to the specific task at hand. For example, in a pure WSS 3.0 deployment, the search service is logically within the Application tier. Installing SharePoint 2007 adds more application services to this tier.
- *Tier 3*: Commonly known as the *Database* tier, this tier is responsible for storing and returning the data to tier 1 and tier 2 services. In WSS and SharePoint 2007 all user data and the vast majority of configuration data is held in a SQL server.

Like all good logical architectures, this gives us great flexibility when it comes to the physical architecture. Each tier is logically separate but all tiers can be displayed on a single server for the needs of a small business or on any number of servers to support the largest of enterprises with full redundancy built in. Indeed, the ability to easily scale the architecture for performance and/or avail-

ability requirements is crucial if SharePoint is to host business-critical information and respond to the needs of dynamic organizations.

Depending upon your needs, you can deploy WSS 3.0 on a single-server (a stand-alone installation) or onto multiple servers (a server farm). A big change in the topology options from the previous version of SharePoint (specifically SPS 2003) is the fact that a single server can host any number of roles (WFE, application, database), and thus the previous topology restrictions are now gone. We still talk about small, medium, and large farms, but the number of servers needed to support such farms is no longer fixed. Indeed, you can create a farm with any number of servers each running any number of roles.

To finish off this introduction to the tiered architecture and deployment topologies, let's take a pictorial look at some typical WSS 3.0 deployments. Figure 2.4 shows examples of small, medium, and large farm deployments—note that we say examples, as these topologies are flexible and can change to suit your needs. For example in the large farm scenario it may be desirable to have multiple content databases served by multiple clustered or mirrored SQL servers.

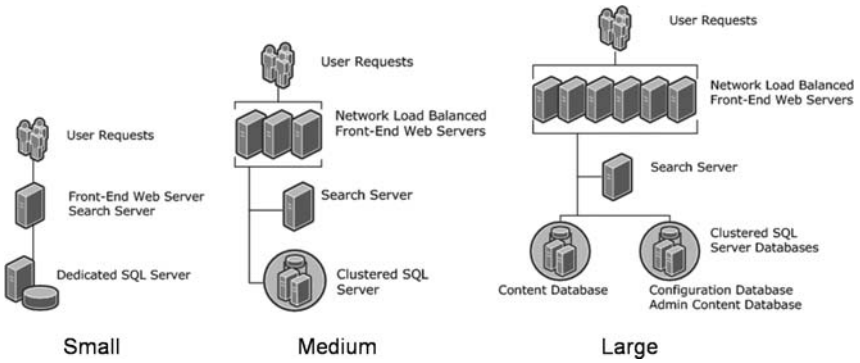


Figure 2.4
Sample topologies.

Front-end Web servers are stateless and, as such, high availability is achieved by having multiple clones of them with some form of hardware or software-based load balancing of user requests.

2.2.2 The Site Framework: Web Applications and Site Collections

WSS 3.0 is a Web-based application, and Internet Information Services (IIS), running on the Web front end servers, initially processes user requests. Ultimately these requests come in the form of a Uniform Resource Locator (URL)—for example, http://spies.spysrus.com or http://spysrus.com/sites/

spies or http://spysrus.com/sites/spies/secrets/gadgets.doc—so somehow, WSS specific processing must occur based on the incoming URL.

IIS itself can support multiple Web sites, each one uniquely defined using a combination of IP address, port number, and host header. Inside each Web site you can have one or more virtual directories which essentially map to the URLs that endusers type into their browsers to access a particular application. For example, Figure 2.5 shows IIS on an Exchange 2007 Server called DC1 that has had the client access role installed onto it. Users can access the OWA (Outlook Web Access) application by typing http://dc1.spysrus.com/owa into their browser (assuming the infrastructure manages to resolve the name dc1.spysrus.com to the IP address associated with the default Web site on the server called dc1!).

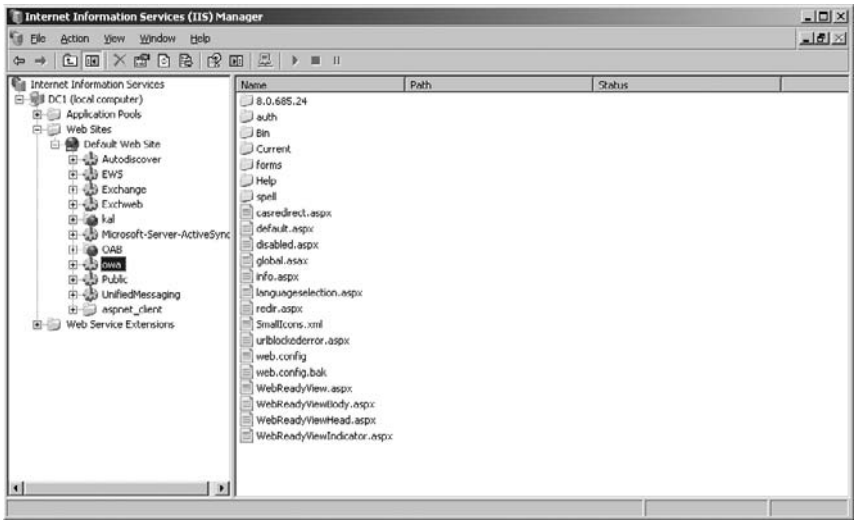


Figure 2.5 *A typical IIS installation.*

When you want WSS to handle incoming URLs, you have to transform an IIS Web site into what is known as a Web application (in WSS 2.0/SPS 2003 terminology, a Web application was known as a Virtual Server). Essentially, you extend an IIS Web site, turning it into a SharePoint *Web application* whereupon WSS handles the URL requests. And just as IIS itself can host multiple Web sites so too can you have multiple Web applications. Would you need multiple? That's a question that needs to be answered as part of your deployment planning, but there are many reasons that would influence this decision, such as differing security needs, content isolation, and namespace requirements. For now, since we are basically just introducing concepts, it's sufficient to know that it is possible to have multiple Web applications on a WFE server. Indeed, as we mentioned at the closing of the previ-

ous section, WFE servers are clones of each other, so each Web application's definition is replicated across all WFEs. In WSS 3.0, this is done automatically with the topology platform service, whereas in WSS 2.0, any virtual servers had to be manually cloned across physical WFE servers.

Web applications host one or more *site collections*. As its name suggests, the site collection is a collection of sites consisting of one top-level site and one or more *subsites*. All the sites in a site collection will be part of the same URL namespace from the top-level site down—so, for example, if we have http://spysrus.com/sites/spies as our top-level site, then a subsite called gadgets will have the URL http://spysrus.com/sites/spies/gadgets. Various resources are shared between the different sites in a collection such as site-wide columns and quotas. Additionally, security settings can be inherited from a parent site or can be unique for a specific site in the collection. We can see in Figure 2.6 how site collections are organized within a farm, and Figure 2.7 provides a sample deployment that supports multiple namespaces through multiple Web applications.

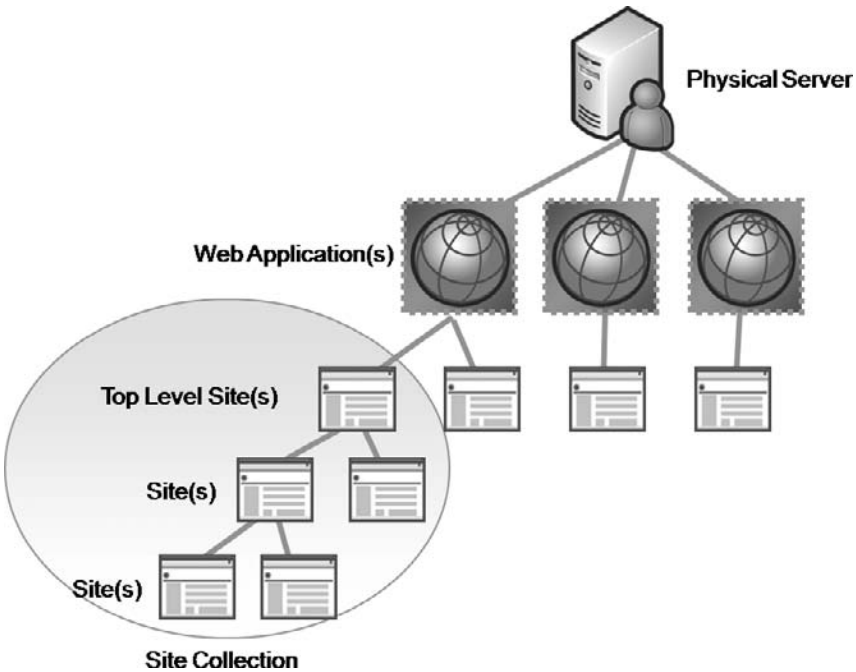


Figure 2.6
Site collections.

Part of your deployment planning process will be to decide how many site collections are needed within each Web application and what the namespace for these site collections will be. There will be many influencing factors here, including identity and the resources that need to be shared between different sites. By identity, we are referring to whether the URL is

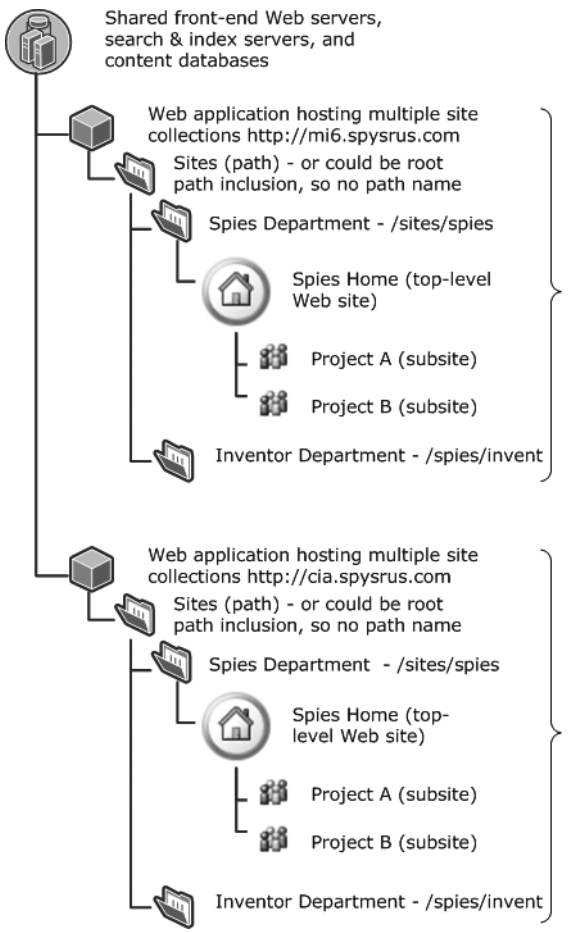


Figure 2.7 *Sample site hierarchy.*

unique for each business that is sharing the farm. For example, your company may be an ISP that hosts multiple companies, and each company has one site collection. In this case, you will probably want to have namespaces such as http://companya.com/ and http://companyb.com/ rather than something like http://isp/sites/companya and http://isp/sites/companyb. When talking about resources that need to be shared, we are referring to site collection–wide resources such as site columns and lookup lists. The consumers of these resources must live in the same site collection. Indeed from a site point of view the top-most unit of administration is the site collection.

In WSS 3.0, there are two ways in which a namespace for a site collection can be constructed. When a site collection is created through the user interface, the URL will be calculated from the domain name associated with the Web application followed by an optional path (such as /sites, /personal etc.) followed by the top-level site name. Site collections can also be named

using host header mode, but this can only be done from the command line at site collection creation time. Host header mode also requires extra infrastructural work to resolve the host header name to the Web application but it is typically used to enable completely separate identities for sites hosted on the same farm. If you also consider that separate Web applications can also be used to configure different namespaces, then you will realize that you have a lot of choice. Ultimately, you need to consider the purpose of the underlying sites, what content they will contain, who will use them, and who will manage them before you can decide how to partition your namespace and what this will ultimately mean to the number of Web applications and site collections you create. Figure 2.8 shows the usage of two different namespaces to refer to two site collections hosted on the same Web application (called wss.spysrus.com). The mi6.spysrus.com site collection was created using the command line utility called stsadm (located in \Program Files\Common Files\Microsoft Shared\Web server extensions\12\BIN), as follows:

```
Stsadm –o createsite -url http://mi6.spysrus.com -ownerlogin
spysrus\jane -
owneremail moneypenny@spysrus.com -hhurl http://
wss.spysrus.com
```

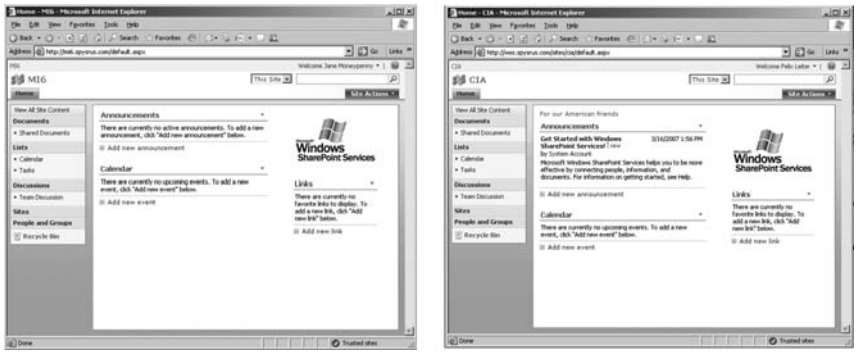


Figure 2.8
Site collection URLs

2.2.3 Site Storage—Configuration and Content Databases

So we now know that we can provision Web applications and create site collections inside them. And, as we'll come to discover, we can create content inside our sites in terms of documents and lists and Web pages that can display this content. So where is all this stored?

Almost all site content and configuration information is held in SQL databases in tier 3 and we must consider two main databases. The first, the

configuration database, controls the overall farm configuration—for example it holds details of all the Web applications that are provisioned on the farm. As such, each server that is attached to the farm connects to the same configuration database. Of course, you may have multiple farms in your organization so how does a server know which configuration database to connect to when it starts up? After you install SharePoint on a server and begin to configure it using the configuration wizard, it will ask you whether you want to join an existing farm (in which case you have to supply the name of the configuration database for that farm) or create a new instance, in which case a new configuration database is created. In both cases, the name of the configuration database is held in the registry on the local server at HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\Web Server Extensions\12.0\Secure\ConfigDB.

The second type of database is the *content database*, and each Web application can have one or more of these attached to it. By default, a single content database is created when you first create the Web application. The content databases are where the core content of a site collection is stored, and each site collection's content is stored wholly within one content database. As you create content databases, you indicate how many site collections each one can contain and, at site collection creation time, the content database with the freest number of "slots" will be chosen to host the content for the new site collection. Note that this choice is purely based on number of site collections already in the content databases; it does not consider how full they are in terms of size. This is the same as it was in WSS 2.0, and therefore you need to have a controlled process for site collection creation if you want to control which content databases hold which content. Content databases can be disconnected from a Web application and ported into another Web application or indeed another farm.

Planning the number of content databases and number of site collections within them is a key design issue for any SharePoint implementation since it will affect things like backup, restore, and service level agreements. Incidentally, the SharePoint Central Administration program is used to manage your Web applications, content databases, and many other things. And guess what? The administration program runs in its own Web application, created automatically when you install WSS 3.0.

To close this introductory discussion on databases, let's look at a couple of important tables that exist within them. Web applications, databases, and various other system resources each have an entry in a table called Objects in the configuration database. From this table, for example, we can figure out which content databases belong to which Web application. Another very important table in the configuration database is the SiteMap table, shown in Figure 2.9. An entry exists in here for every site collection in the farm, and this entry allows us to establish the Web application and specific content

database where a site collection is housed. Furthermore, this table lets us know whether the namespace of a site collection is a host header or a "normal" namespace. Note the last row in Figure 2.9. This is our mi6.spysrus.com site collection and the Path and HostHeaderIsSiteName columns reflect this. Note also how the site collections for our wss.spysrus.com Web application are split across two content databases (look at the ApplicationId and DatabaseId columns).

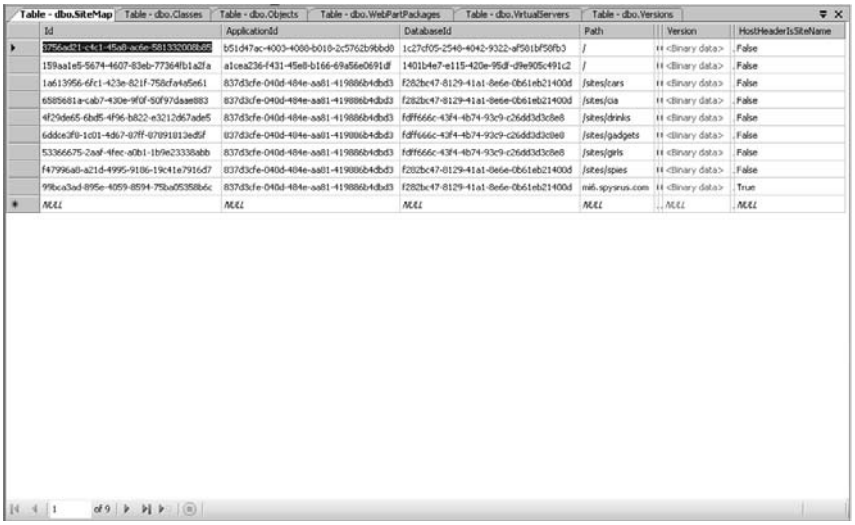


Figure 2.9
SiteMap table in the configuration database.

The SiteMap table in the configuration database is mainly used to give a farm-wide view of all site collections, but there is also a table called sites within each content database, that describes the site collections within that particular database in more detail. The presence of this table also makes it possible to detach a content database and import it into a different Web application and/or farm.

The last table we would like to draw your attention to is the Webs table, which exists in each content database. As mentioned previously, a site collection consists of a top-level site and one or more subsites. You can think of the Sites table as a directory of site collections, and the Webs table contains the items within that directory. Therefore, each site collection will have at least one entry in the Webs table, and each entry is tied back to its site collection entry in the Sites table. Each entry in the Webs table corresponds to a physical site that can be accessed via a unique URL. Incidentally, the reason the table is called Webs is that in earlier versions of SharePoint, a physical site was referred to as a "Web." From now on, when we use the term "site" we are referring to an individual site, whether it be the top-level site in a collection or a subsite.

You should now have a reasonable understanding of how site collections are structured and where the content within them is stored, so let's now take a look at the core platform services in terms of how sites are displayed, what features we get for our content in terms of storage, and how security works against our site collections.

2.3 Rendering Sites

Take a look at the two screenshots in Figure 2.10. I think you'll admit that from a visual point of view, they feel the same in terms of cosmetics and navigation—yet the content they deliver and the purpose of their being is completely different. However they both live in the same site collection (take a close look at their URLs if you don't believe me: http://mi6.spysrus.com and http://mi6.spysrus.com/secrets), and ultimately the content within each site boils down to some items in libraries and lists. So what is it that allows them to feel the same yet address a different business need and present a different face to the enduser? The answer lies in the fact that WSS 3.0 harnesses ASP.NET 2.0 (as we saw in Figure 2.1) and offers its own services through site definitions, templates, and features.

Figure 2.10
Same look and feel - different purpose

What ASP.NET 2.0 brings to the table is the ability to offer a consistent user interface through something called *master pages*, *page layouts* and *content pages*. In addition, WSS makes available *site definitions*, *site templates, features*, and *common navigation*. The combination of these powerful technologies results in easy-to-use Web sites that are rich in functionality. We cover all of these topics in much more detail in the developer's section of this book, but let's just touch on each of them here to give you a basic understanding.

2.3.1 Master Pages, Page Layouts, and Content Pages

Master pages are an ASP.NET 2.0 feature that lets child pages inherit and reuse a common design. This feature makes it very easy to ensure that all pages within a site share a consistent look and feel and makes the development and maintenance of the site much simpler. For example, it makes it easier to

change the look and feel, which is a common occurrence in these days of mergers and acquisitions. Typically, master pages contain the branding, navigation, header, and footer for the site and include placeholders for specific content. The definition of that content is taken from other pages known as page layouts. A page layout is a template for a content page and defines the structure and layout of the page. A content page is an instance of a page layout upon which you enter the actual content.

Page layouts offer consistency in the way that specific content is presented on a page, even though the information on each page may be about different subject matter. For example, the content associated with product descriptions, press releases, and executive biographies can all be presented with a standard look and feel by using page layouts with details such as title and descriptions coming from items stored inside a WSS 3.0 list. Page layouts are used extensively in publishing type sites.

Master pages, page layouts, and content (typically held in a list within a WSS 3.0 site) all combine to ultimately deliver the HTML page to the requesting client. Master pages can be associated with a site definition: by default, there is one master page per site, but you can have as many master pages as you like for different purposes. Within each site there is a master page gallery where you can upload your master pages and page layouts. However, in order to apply a master page to a WSS 3.0 site, you have to use a WSS 3.0–compatible Web design tool such as SharePoint Designer 2007—which we cover in the developer part of this book! Do note, however, that when you install MOSS you can apply the master page to a site and all its subsites through the browser UI.

2.3.2 Site Definitions, Site Templates, and Features

Site definitions provide the same functionality as they did in WSS 2.0, but they have been much simplified due to the introduction of Features. Every site is initially provisioned from a site definition, which defines things like the initial lists, libraries, views, Web Parts, and pages that will be contained in a site. A site definition is defined via a set of XML files that live under \Program Files\Common Files\Microsoft Shared\Web server extensions\12\Templates\SiteTemplates. Each definition can contain one or more configurations, which are presented to the enduser during site creation in the Template Selection section, as shown in Figure 2.11.

You can also create snapshots of existing sites and store them as custom site templates, which augment the list of templates a user can choose from when creating a site.

Site definitions were quite cumbersome and fairly inflexible in WSS 2.0, and it was often very difficult to add new functionality to a site after it had been provisioned. Rather than dwell on why this was the case let's take a look

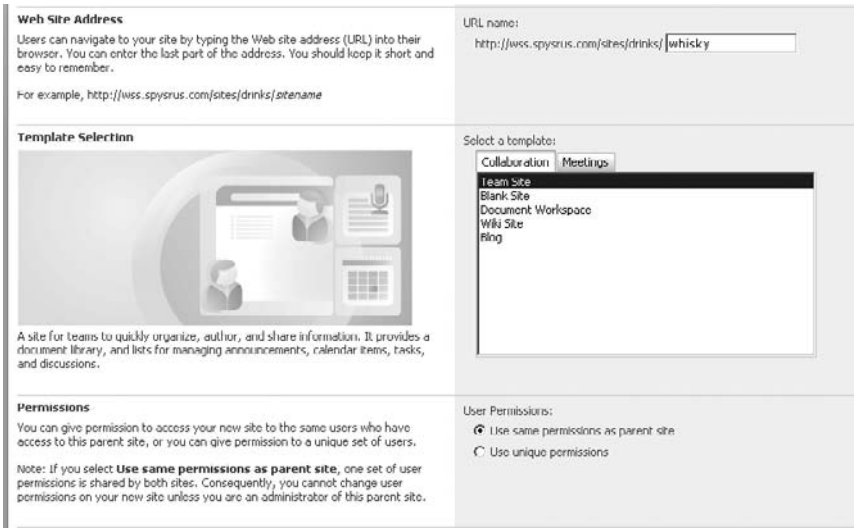


Figure 2.11
Template selection.

at how this whole area has been improved. Features are a new feature (you read correctly—this new feature is called Features!) in WSS 3.0 that reduce the complexity of site customizations, allowing you to install chunks of functionality and then "light-up" individual sites by merely activating the new functionality. Features take many forms (for example, lists are now a feature, as are various administration functions on a site), can be grouped together, and are themselves defined using XML files located at C:\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\Features. Inside the XML, you will find references to assemblies that implement the feature and other supporting files such as aspx pages.

There are 31 features installed in a base WSS 3.0 deployment; we will see this number expand significantly when we install SharePoint Server 2007 on top. We will also see the introduction of many more site templates, but when it all boils down these will just be base sites with several new features activated.

The simplest way to show the features capability for now is to look at some screenshots. The most basic site you can create out of the box is a blank site—it has no precreated lists or libraries in it and is basically a shell into which you would create the lists and libraries that suit your purpose. It has one activated feature called team collaboration lists; this feature allows you to create various lists and libraries. If we were to deactivate this feature, you could not create any of the team-based lists and libraries. Figure 2.12 shows the various lists available for creation in our blank team site (which I called Features!). Figure 2.13 shows the contents of the team collaboration lists definition at C:\Program Files\Common Files\Microsoft Shared\Web server extensions\12\Template\Features\TeamCollab\feature.xml. The features defined here are mainly lists, but there is also a workflow feature that

can be applied to items within a list. We can see the workflow settings option in Figure 2.14 available within a document library.

If we now deactivate the feature (via Site Actions\Site Settings\Site Administration\Site Features, and shown in Figure 2.15) we can see in Figure 2.16 how we no longer have the option of creating any lists or libraries and, in Figure 2.17, how our workflow feature has been removed.

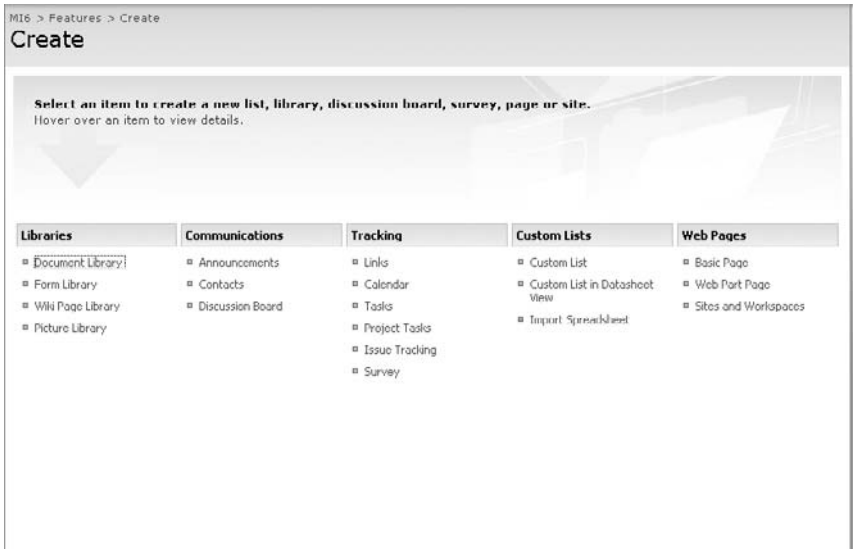


Figure 2.12 *Team Collaboration Lists*

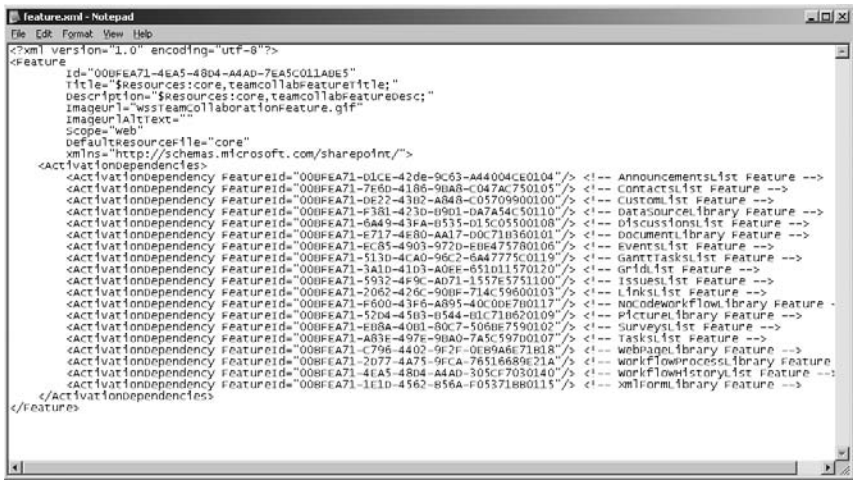


Figure 2.13 *Feature.XML for Team Collaboration Lists*

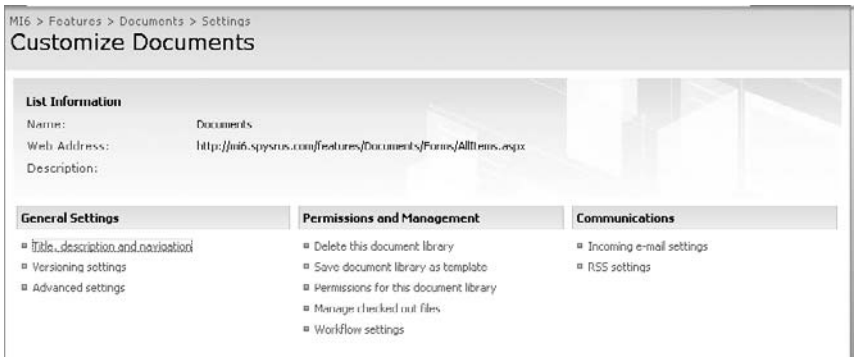


Figure 2.14 *Workflow settings available.*

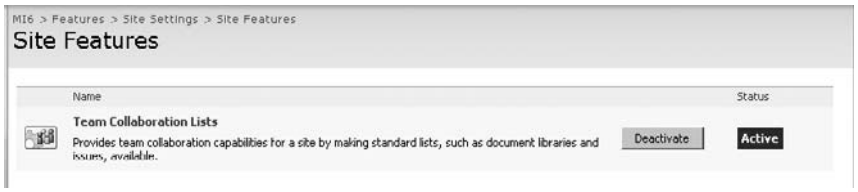


Figure 2.15 *Deactivating the feature.*

Figure 2.16 *No more lists available for creation!*

2.3.3 Navigation

Anyone who has used WSS 2.0 will know that the navigation around sites within a site collection was, how shall we put it, not the best experience in the world. We are happy to report that this situation has been drastically improved in WSS 3.0, and we now have a structure that can be used to tie multiple sites together in any form of hierarchy you see fit.

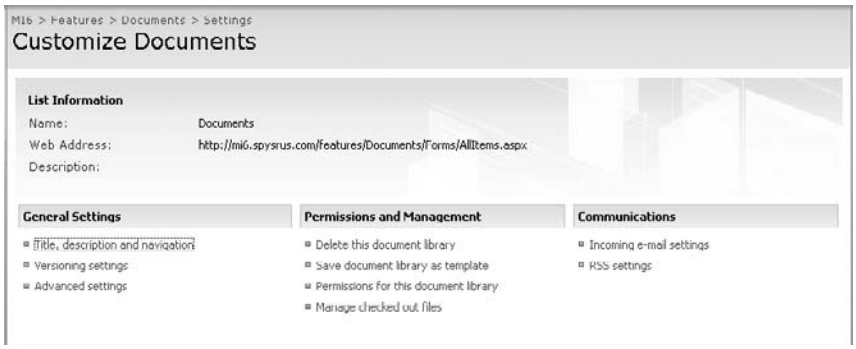


Figure 2.17
And no workflow available anymore.

This is important when you consider that in WSS 3.0, a site is a site is a site, and you form your own topologies for your particular needs. WSS 2.0 had no in-built way of forming topologies, but SPS 2003 did, in terms of navigable portal areas. Now you build a portal by tying together any number of sites you like, as we will see when you talk about SharePoint Server 2007.

The core site model supports three forms of navigation. A horizontal top link bar allows you to quickly jump to different sites, the vertical quick launch bar has site-specific navigation in it, and a breadcrumb trail makes it easy for you to find your way back from whence you came as you drill down within a site.

When you create a subsite within a site collection, you have the option of adding a link to that site on the quick launch bar of its parent and adding a link to the subsite onto the parent's top link bar. Additionally you have a choice as to what top-link bar should be used for the subsite itself—whether is should inherit that of its parent or use its own. These options are shown in Figure 2.18. Navigation inheritance is typically used in a portal site, in which different "sections" of the portal are for different business divisions. Figure 2.10 actually shows navigation inheritance in action: the top-level site and the subsite have a consistent top link bar. You can also manually add links to the top-link and quick launch bars of any site, but remember that when these links are taken, you will be taken into the context of the destination of that link, and therefore there is no guarantee that there will be any form of navigation to get you back to where you came from—except, perhaps, the back button on your browser.

One other addition aids navigation: the tree view control, which shows the content of your site collection in a navigable hierarchy and provides a very powerful way of browsing site content. You can easily navigate to subsites and subfolders within libraries and lists. You need to turn on the tree view on a site-by-site basis via Site Actions/Site Settings/Tree View. When you do, you reveal a user interface, as shown in Figure 2.19. Note that you can change the order of the items in the Quick Launch bar via the browser

Figure 2.18
Navigation options when creating a site

through Site Settings but, unfortunately, the tree view is not included in this ability, so it always appears at the bottom. That said the quick launch and tree view controls are actually stored in the master page for the site (so a quick edit of the master page can change the layout for you).

Figure 2.19
Tree view in action.

Hopefully by now, you have a good understanding of how the core site model allows you to create sites of different types, how you can control the namespaces of these sites, and where the vast majority of the content of the sites are stored. But what core services do we see in the storage space? It's important to understand this, as it will help you better appreciate how the various solutions use these features.

2.4 Storage

The core storage mechanism within a site is that of *lists* and *libraries*, with libraries just being a special form of list. Lists can be used for whatever purpose you see fit; some standard lists such as Announcements, Contacts, and Events ship out of the box and are contained within various site definitions. A list is defined in terms of *columns* (AKA *metadata* or *properties*); these columns can be of differing types (string, date, number, etc.) and can have validation rules applied to them to maintain consistent and relevant data. WSS 3.0 introduces many list enhancements, including the ability to have multi-valued and append-only columns—the latter being very useful for tracking changes made to items. You can also define your own custom columns that could, for example, validate the data against a back-end enterprise application or ensure the data is of a particular format like a Social Security Number. Different *views* can be applied to lists, allowing flexibility in the display of data within a list (for example you can group, sort and filter the items in a list via a view), and you can add your own custom views as necessary.

Different types of libraries are available within WSS 3.0, with the main one being a document library (there are also picture and form libraries available out of the box). The main difference between a library and a list is that one of the columns in the library contains the physical document itself. There are many ways to interact with content stored in lists and libraries. The primary vehicle from an enduser's perspective is the browser, where content is typically displayed via Web Part pages and new items can be created directly in a list or library (you can upload documents through the browser into a library). The rich API and Web services that WSS 3.0 supports enable many other forms of interacting with the content within a site. For example, library documents can be opened and saved directly from an Office 2007 application such as Word or Excel, and Outlook 2007 can be used as a smart client to WSS 3.0 content, facilitating the reading and writing of list items directly from Outlook! Indeed, as we will see later, you can even e-mail content into a list or library.

Lists and libraries existed in WSS 2.0 and served their purpose fairly well. However, they had some major restrictions that would have prevented the rich set of solutions areas being delivered, had major enhancements not been made. These enhancements fall into the metadata, administration, security, and versioning areas. Let's take a look at each of these.

2.4.1 Metadata—Introducing Content Types and Site Columns

A big disappointment for many customers who moved from SPS 2001 to SPS 2003 was the removal of document profiles. These were essentially definitions

of metadata for different types of documents, enabling you to support multiple profiles within a single document library. In SPS 2003 (and WSS 2.0), each library and list can only have one metadata definition associated with it (essentially the list of columns that the list/library supports), and this therefore dictated that multiple document libraries be created to support different document types. This was clearly an inconvenience for many situations, not least because it meant existing filing systems had to be redesigned and single document libraries split into multiple to support the same scenarios.

WSS 3.0 introduces *content types* through which multiple metadata definitions can be associated with a single library or list. This is obviously a good thing, but it will provide some challenges in migrating to WSS 3.0 if multiple libraries need to be consolidated into a fewer number. Content types are used for many different purposes in SharePoint 2007—defining page layouts, controlling workflow on a particular item, and creating search scopes—and reveal themselves in many different places, such as when navigating a list/library through the browser or via the document properties pane within Office 2007 applications. As an example, look at Figure 2.20, where we have defined a new content type called Spy Movies that has a simple piece of metadata associated with it called genre. Here we are looking at that metadata through the browser and through Word 2007 for a document that has been associated with the content type Spy Movies.

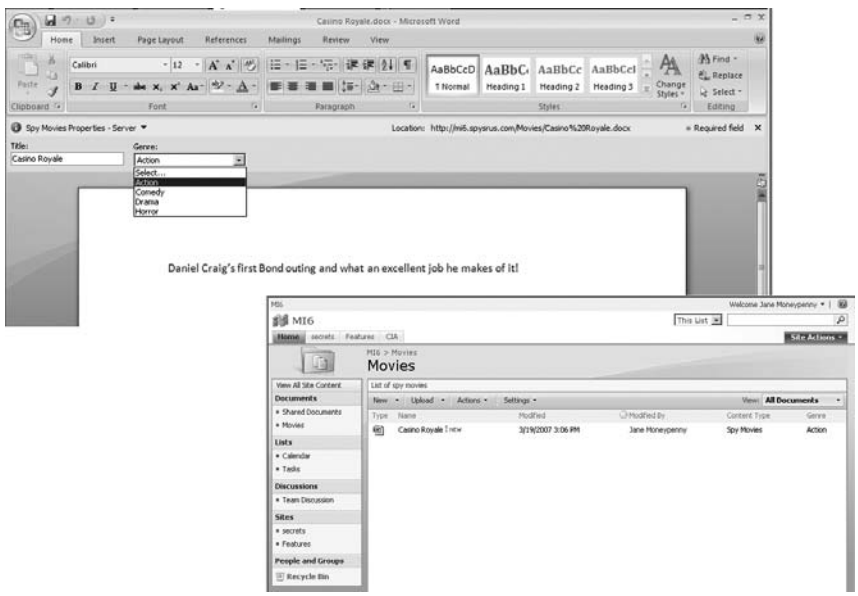


Figure 2.20
Metadata and content types.

Content types are categorized into groups for ease of management, and you can create new groups to suit your own needs. Each site in a collection

has its own content type gallery, and a site can use any content type defined in its own gallery or any of its parent's galleries. When you create a new content type, it is always instantiated from an existing parent content type, shown in Figure 2.21. When you view the available content types via Site Actions/Site Settings/Site Content Types, the list presented to you is a consolidated list of all the content types available from the current site and any parent sites. Content types from the current site are highlighted in blue in the user interface; We've highlighted this on the Spy Movies content type in Figure 2.21.

Although content types are inherited as soon as you apply a type to an item within a list or library, an instance of the type is physically created inside that list or library, which is then known as a *list content type*. You can therefore think of the content type itself as a template that is used to instantiate list content types. Indeed, the standard documentation differentiates the two types as site content type and list content type. Once a list content type is created you can further customize it by adding site columns to it, but that will only affect that single instance. When you modify a site content type, you have the option of updating any list content types that were instantiated from the site content type.

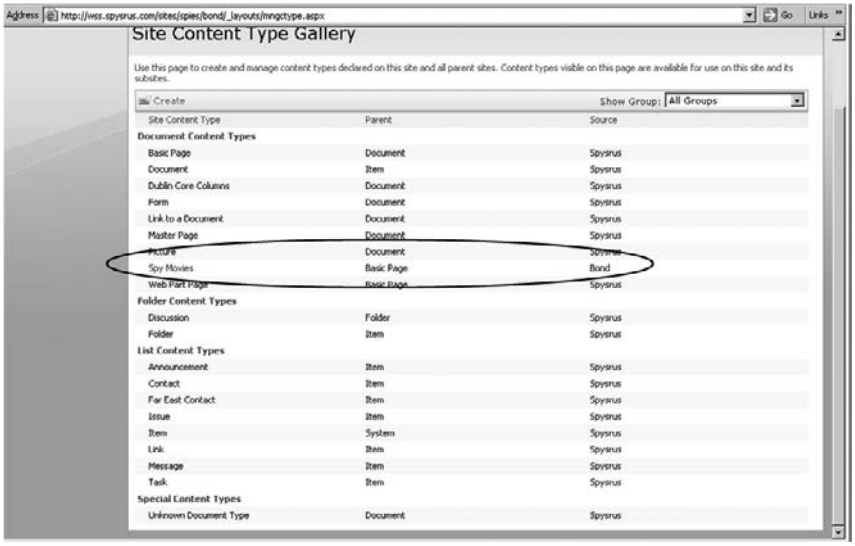


Figure 2.21
Content types and inheritance.

Content types can be enabled on a per-list or library level, and you may notice that in a default team site they are disabled on most libraries and lists yet a discussion list is enabled supporting the discussion and message content types. How is this controlled? It's all related to features. Lists are enabled via features, and the site definition for a team site contains the discussion list feature. The schema for the discussion list Feature enables content types. You

can see this by viewing the schema.xml, located at \Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\FEATURES\DiscussionsList\Discuss. You can also use features to deploy a new content type if you so wish.

We have been talking about metadata and how you can group it together to form a content type that can help describe a particular piece of information. This metadata is applied to items via columns, so let's take a look at what site columns are.

A content type consists of one or more *site columns*, and these columns essentially define the base schema for the content. You can, however, define site columns and apply them to individual lists or libraries regardless of the content types that are enabled on that list or library. In fact, a common operation for any site administrator is to create custom lists from a list of available site columns. Every site has a site column library, and you can apply any site column from the current site and any of its parent sites. The value in being able to apply a parent column to a child site is that you can ensure consistency of data across the whole of your site collection. For example, you may want to maintain a central list of product codes at the root of your site collection and have any down-level lists in child sites validate their data against this list when they create any item that requires a product code. You can see this in action in Figure 2.22, where we have defined a site column at the root of our site collection that validates its contents from a list in the top-level site of the site collection (called Code Names). This column is then added to a Contacts list in a subsite of our site collection, thus ensuring consistent data.

2.4.2 Administration—Folders, Recycle Bin, and AutoCopy

WSS 3.0 supports much more scalable lists than in WSS 2.0 in terms of the number of items and the "width" (i.e., number of site columns) of each list. In WSS 2.0, a list was limited to 64 columns; this has now been extended to 2,000. Of course, if you do have very large numbers of items in a list (in theory, a list or library can hold millions of items) then you are faced with a challenge on how best to display this content to the enduser. Generally speaking, if you want to display more than 1,000 items in a single view, you need to consider indexing certain columns. Indexing a column and then using that column in a filtered view is the most efficient way to display a subset of items from a large corpus.

Unlike WSS 2.0, which only supported folders within document libraries, you can now choose to enable folders for lists as well. This can also help in organizing large amounts of items in a list, as long as the enduser knows which folders should be used for which items, so enduser training is key here, as is some planning on filing structures. While folders can help with brows-

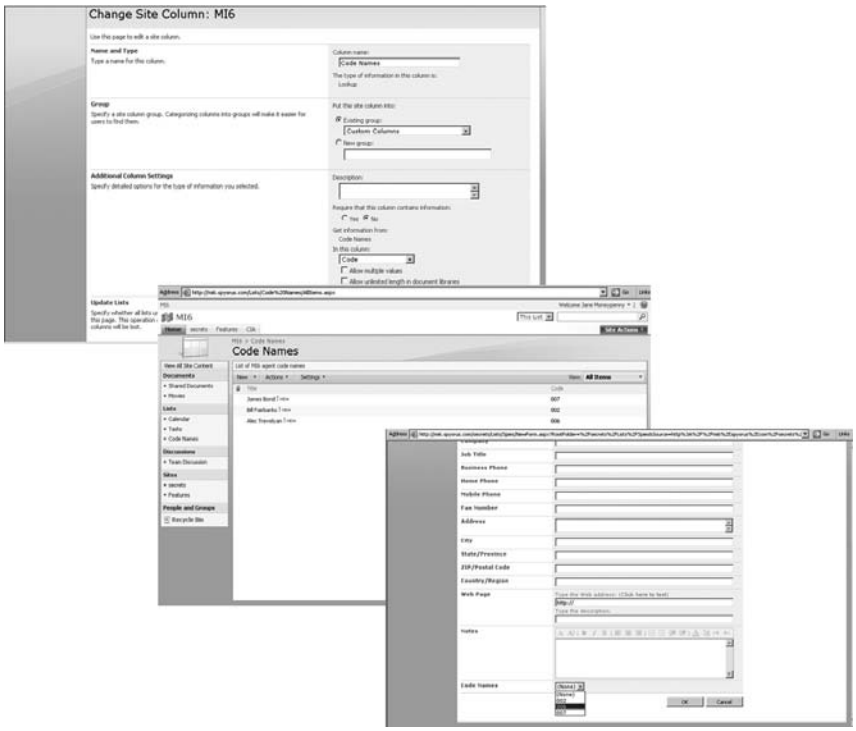


Figure 2.22
Site columns.

ing information, you should still consider using a filtered view on an indexed column when there are large numbers of items.

Another improvement in the storage area with WSS 3.0 is the introduction of a *recycle bin*, which can act as a content recovery tool when accidental deletion has occurred. Anyone who used WSS 2.0/SPS 2003 must be applauding this, since the lack of such a feature has certainly placed an unnecessary load on central IT operations for common restore requests. The implementation in WSS 3.0 is actually two-stage: there is a first-stage recycle bin per site that suitably privileged end-users can access and a second-stage one that is scoped to the site collection. If a site-specific recycle bin has been emptied, a site administrator can still recover content from the site collection recycle bin. This second-stage recycle bin is actually shared by all site collections within a Web Application and, if it is disabled for the Web application via SharePoint Central Administration, all second-stage recycle bins within that Web Application will be emptied. SharePoint Central Administrators decide how long items stay in a recycle bin, with the default being set to thirty days for the first-stage bin. The second-stage bin does not have a number of days associated with it, but a percentage of the site quota for each site collection. Note that only deleted items go into the recycle bin—if you were

to accidentally delete a whole site, you'd have to call on IT operations to restore for you.

AutoCopy is the term used to describe a feature that allows you to copy content in one library to another library either within the same site collection or in a completely different collection. A source document can be copied to multiple locations, and you can be reminded to update the copies when the source copy changes, either via an alert or by a dialog presented to the author who is updating and checking the source copy. The person that is setting up the copy must have write access to the destination document library. Copies of documents retain a link to the source, and it is therefore simple to navigate back to the source copy or break the link between source and copy if so desired. AutoCopy is a crucial component when it comes to building publishing sites: typically, you'd have a development site that pushes content out to a staging site and then ultimately out to an Internet-facing site.

2.4.3 Versioning—A Major and Minor Enhancement

Versioning and many of its surrounding aspects has been significantly enhanced in WSS 3.0. In WSS 2.0, versioning was only available within document libraries, only major versions were supported, and there was no end-user support regarding management of checked out copies, which generally caused confusion and accidental overwriting of information. All of this has changed in WSS 3.0 with versioning now available for list items in all default list types and for all file types that can be stored in libraries. Tracking changes to the metadata of an item is also done, so that you can see, for example, who changed the due date of a task.

Simple versioning, which only deals with major versions, can be enabled on a list or library. *Advanced versioning*, which tracks both major and minor revisions, can only be enabled on a document library. When simple versioning is enabled on a list or library, a new major version of the item is created any time the item is updated—be it by modifying a property or by updating the underlying document. If you are editing the document in a SharePoint compatible application such as Office 2007, the major version number is only incremented the first time you hit the Save button from within the application. This avoids unnecessary versions as you edit the document and hit the save button repeatedly. If you close the document and re-edit it via the Office program, then a new version will be created when you first hit the save button.

You can control the number of major and minor versions you want to keep and the version history option (available from the drop-down menu you get when hovering over an item in a list or library) allows you to view the different versions. You can also see any metadata that was changed between versions, as shown in Figure 2.23—this is a task list that was ena-

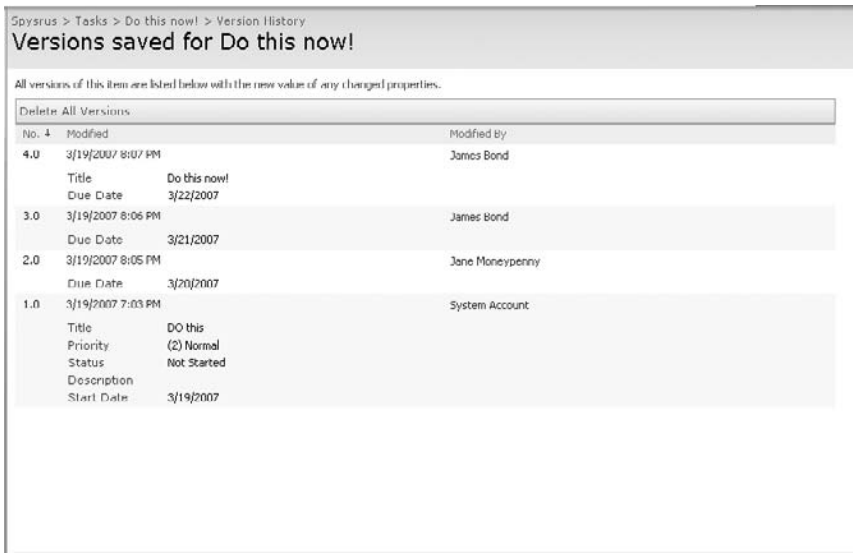


Figure 2.23
Version history on a task item.

bled for versioning. By hovering over one of the items in the history, you can restore it, making it the most current major version.

When both major and minor versioning is enabled on a library the major version is only incremented when an act of "publishing" is performed. The minor version is incremented whenever a property or the underlying document is changed, and you can control who is allowed to see minor versions. Typically, only authors would be allowed to see minor versions, with all other users only being able to see the most recently published (and therefore major) version. The publishing act can be performed in multiple ways, but typically someone with author rights to the library in question will do this as part of checking-in an updated copy of the document. When both major and minor versioning are in force, you can chose how many major versions to keep and also how many of those major versions also have their minor versions kept. Through the version history, you can see which version is the most recently published version (as shown in Figure 2.24); there are also options available to unpublish the current version.

Versioning also integrates with two other features—*content approval* and *check-in/check-out*. When content approval is required, a list item or file remains in a draft or pending state until it is approved or rejected by someone who has the appropriate permission. It is possible to associate a workflow to achieve the approval process. You can also control who is allowed to see draft items—typically this would be the creator of the item and users with the right to approve it, and other users would not see the item until it is approved.

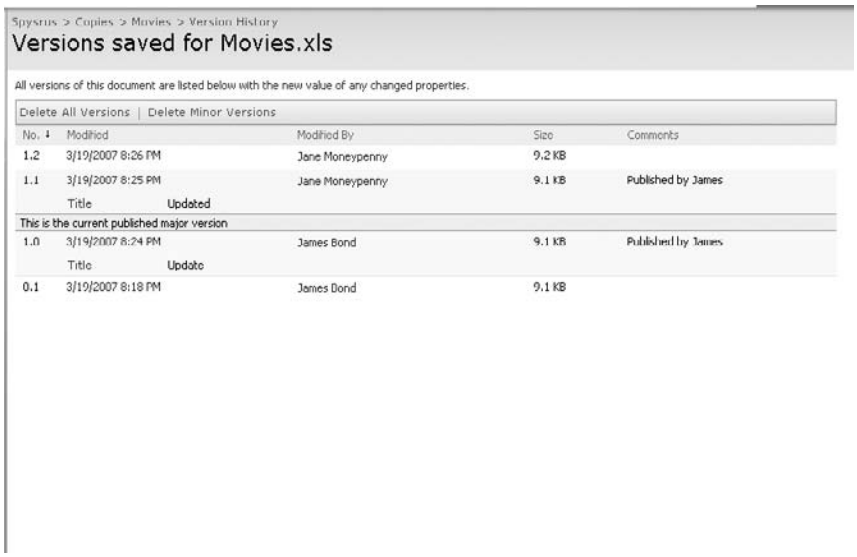


Figure 2.24
Major and minor versioning in force.

Libraries support check-in and check-out, and you can configure them such that a check-out is mandatory before an item can be changed. Doing so helps avoid one person's changes being overwritten by another's, since only the person who has checked an item out can check it back in again (or a site administrator). When you attempt to edit an item in a library where a mandatory check-out is required, a dialog box asks whether you want to check the item out there and then. This happens for both the editing of a property or the underlying document itself. Office 2007 applications are aware of this check-in/check-out functionality and will offer the user the chance to check-in when the document is saved. When you do check in a document, you are given the option of publishing a minor version or a major version, and also whether you want to keep the document checked out.

Checking out the document also offers you the ability to take a copy onto your local machine, so that you can work offline with the document. If the document is Office 2007, there are some other integration points that allow you to automatically update the document into the library when you next have a network connection. We cover Office 2007 integration in more detail in a later chapter.

Before we close this section, you should know that the vast majority of the options we have discussed here reveal themselves in the user interface via the drop-down menu that appears when you hover over an item in a list or library.

2.4.4 Search

Search is a core service in WSS 3.0 that allows you to search a site collection or individual site for content. The search itself is completely different from that in WSS 2.0 and delivers far more value. We cover both WSS 3.0 and MOSS search in Chapter 6.

2.5 Security Model

Security covers a wide gamut of features, from *item-level permissions* to *security-trimmed user interface* and *pluggable authentication*. The new security model is key in being able to meet the needs of the different solutions areas, so let's take a closer look at it.

2.5.1 Authorization

By authorization we are referring to who is authorized to do what. Figuring out someone's identity is the purpose of authentication, and it goes hand in hand with authorization. For now, let's assume we know who someone is and examine the ways we can control what that person is and isn't allowed to do.

A major change from WSS 2.0 security is in the separation of groups and permissions. In WSS 2.0, permissions were associated with site groups (such as Administrators, Readers, Web Designer, etc.) and you added security principles (users and Windows security groups) to the site groups. Doing so automatically granted the security principle the permissions associated with the site group. In WSS 3.0, you still have the concept of groups that contain security principles, but you now have separate permissions that have to be assigned to a user or group before they can have access to a securable object such as a site, list, or library or item.

There are five main elements for authorization:

- *Individual permissions*: Grant the ability to perform specific actions. For example, the View Items permission grants the user the ability to view items in a list. The list of individual permissions that are available are farm-wide, but can be controlled at the Web Application level by a farm-level administrator.
- *Permission level*: A way of grouping individual permissions together for easier management and assignment. There are five default permission levels: Limited Access, Read, Contribute, Design, and Full Control. You can add new permission levels or change the default levels. Permission levels are per site and can either be inherited from a parent site or explicitly set at a sub site, library, or item level.

- *User*: A person with a user account that can be authenticated through the authentication method used for the server. If we are using the default authentication method, this would typically be an Active Directory user object.
- *Group*: A group of users. Can be a Windows security group that you add to the site, or a *SharePoint Group* such as Site Owners, Site Members, or Site Visitors. SharePoint Groups are new to WSS 3.0 and essentially take over from site groups. They provide a way for SharePoint site collection administrators to group users together without having to rely on IT to create Windows Security Groups. There are some default SharePoint Groups, but you can create your own. Anyone assigned a permission level that includes the Create Groups permission (included in the Full Control permission level by default) can create custom SharePoint groups.
- *Securable object*: Users or groups (either Windows Security groups or SharePoint Groups) are assigned a permission level for a specific securable object: site, list, library, folder, document, or item. By default, permissions for a list, library, folder, document, or item are inherited from the parent site, parent list, or library. However, anyone assigned a permission level for a particular securable object that includes the Manage Permissions permission can change the permissions for that securable object. By default, permissions are initially controlled at the site collection level, with all lists and libraries inheriting the site permissions. Use list-level, folder-level, and item-level permissions to further control which users can view or interact with the site content. You can return to inheriting permissions from a parent list, the site as a whole, or a parent site, at any time. Note that it is best practice to always use a group when assigning a permission level to a securable object for ease of maintenance. Granting individual user access should only be done on an exception basis.

Table 2.1 *Personal Permissions*

Permission	Definition
Manage Personal Views	Create, change, and delete personal views of lists
Add/Remove Personal Web Parts	Add or remove personal Web Parts on a Web Part Page
Update Personal Web Parts	Update Web Parts to display personalized information

The list of individual permissions are grouped into those relating to personal operations, list processing, and site actions. They are shown in Tables 2.1, 2.2, and 2.3.

Table 2.2 *List Level Permissions*

Permission	Definition
Manage Lists	Create and delete lists, add or remove columns in a list, and add or remove public views of a list
Override Check Out	Discard or check in a document that is checked out to another user
Add Items	Add items to lists, add documents to document libraries, and add Web discussion comments
Edit Items	Edit items in lists, edit documents in document libraries, edit Web discussion comments in documents, and customize Web Part Pages in document libraries.
Delete Items	Delete items from a list, documents from a document library, and Web discussion comments in documents
View Items	View items in lists, documents in document libraries, and view Web discussion comments
Approve Items	Approve a minor version of a list item or document
Open Items	View the source of documents with server-side file handlers
View Versions	View past versions of a list item or document
Delete Versions	Delete past versions of a list item or document
Create Alerts	Create e-mail alerts
View Application Pages	View forms, views, and application pages; enumerate lists

SharePoint Groups are used to manage sets of users, rather than individuals. As well as being able to assign a permission level to a SharePoint Group, they are also used elsewhere in SharePoint—for example, targeting content and e-mailing site members. SharePoint Groups can be inherited from a parent site or created for a specific sub-site by breaking the inheritance. There are three main SQL tables that support SharePoint Groups and their membership: groups, GroupMembership, and UserInfo. As soon as a user is given any form of access to a site collection, an entry for them is created in the Use-

Table 2.3 *Site Permissions*

Permission	Definition
Manage Permissions	Create and change permission levels on the site and assign permissions to users and groups
View Usage Data	View reports on site usage
Create Subsites	Create subsites such as team sites, Meeting Workspace sites, and Document Workspace sites
Manage Web Site	Grants the ability to perform all administration tasks for the site, as well as manage content
Add and Customize Pages	Add, change, or delete HTML pages or Web Part Pages, and edit the site using a Windows SharePoint Services–compatible editor
Apply Themes and Borders	Apply a theme or borders to the entire site
Apply Style Sheets	Apply a style sheet (.CSS file) to the site
Create Groups	Create a group of users that can be used anywhere within the site collection
Browse Directories	Enumerate files and folders in a site using SharePoint Designer and Web DAV interfaces
Use Self-Service Site Creation	Create a site using self-service site creation
View Pages	View pages in a site
Enumerate Permissions	Enumerate permissions on the site, list, folder, document, or list item
Browse User Information	View information about users of the site
Manage Alerts	Manage alerts for all users of the site
Use Remote Interfaces	Use SOAP, Web DAV, or SharePoint Designer interfaces to access the site
Use Client Integration Features	Use features which launch client applications. Without this permission, users will have to work on documents locally and upload their changes
Open	Allows users to open a site, list, or folder to access items inside that container

Table 2.3 *Site Permissions (continued)*

Permission	Definition
Edit Personal User Information	Allows a user to change his or her own user information, such as adding a picture

rInfo table. Thus, if a user has access to multiple site collections they will have multiple entries in the UserInfo table. When a SharePoint group is defined, its details are stored in the Groups table, and the GroupMembership table has links to the individual users, as defined in the UserInfo table. Of course these tables all exist per content database, as discussed previously.

The default SharePoint Groups created for sites in WSS 3.0 are listed in Table 2.4.

Table 2.4 *Default SharePoint Groups*

Group name	Default permission level
<Site name> Visitors	Read
<Site name> Members	Contribute
<Site name> Owners	Full Control

As an example of a permission level, the following lists the individual permissions contained within the Read permission level out of the box. These relate back to the individual permissions above, and are all geared toward allowing the user to only view information within a site:

- **Personal Permissions:** None
- **List Permissions:** View Items, Open Items, View Versions, Create Alerts, View Application Pages
- **Site Permissions:** Use Self-Service Site Creation, Browse User Information, Use Remote Interfaces, Use Client Integration Features, Open

SharePoint Groups are used to authorize access to individual sites, but there are some other places you need to consider that provide for site collection administration, farm-wide administration, and physical server administration. *Site collection administrators* have full control over all the sites in a site collection. You can add one or more users as the primary and secondary site

lection administrators can audit all site content and are the contacts for the site should any administrative alerts be sent out (such as verifying if the site is still in use) or requests for access to the site.

The *Farm Administrators group* is an addition to WSS 3.0 and controls who can manage server and server farm settings. In WSS 2.0, you had to add such users to the local Administrator group on the server, which then gave them unnecessarily elevated permissions and also allowed them to gain access to all site collections without any auditing taking place. Thankfully, the introduction of the Farm Administrators group avoids this particular scenario; they can still chose to add themselves to a site collection, but this action is now audited. You specify the members of the Farm Administrators group using the Operations tab of the SharePoint Central Administration program. Note that the group is a SharePoint Group (defined in the content database associated with the Central Administration Web application) and, just as described above, you can add any security principle to it. Therefore, if you have a lot of farm administrators, the best practice would be to create a Windows security group and add it as a member of the Farm Administrators group.

Members of the local *Administrators* group on the physical server can perform all farm administrator actions and more, including deploying Web Parts and features and creating new Web applications. Like farm administrators, they do not implicitly have access to all content, but they can add themselves to any site collection, which again is an audited event.

2.5.2 Authentication

Authentication is the process of identifying someone. In WSS 2.0, you had to identify yourself as a Windows user account before you could access a site. WSS 3.0, however, introduces the ability to do "pluggable authentication" that other security mechanisms can be used. This may be necessary, for example, when you want to have an extranet deployment but do not want to house those users inside your Active Directory. An alternate authentication mechanism, such as an LDAP provider or SQL database, could be used instead. Pluggable authentication is possible via the ASP.NET provider model.

2.5.3 Granular Permissions

In WSS 2.0, the granularity of permissions was down at the list or library level. With SPS 2003, even though its site model was built on WSS 2.0, the granularity was at the "portal area" level, which meant that if you had write access to one list or library in that area then you had write access to all the lists and libraries in that area. Not a great design if you wanted, for

example, users to be able to contribute to a survey in a portal area, since giving them access granted them write access to every other list and library in that portal area!

In WSS 3.0, the level of granularity is now at the item level, which is a welcome progression and ultimately results in far more functional solutions. As we mentioned previously, a securable object in SharePoint can be a site, list, library, document, or item. Permission levels are, by default, inherited from the parent object, but this inheritance can be broken. When this is done, the existing permission levels from the parent are copied onto the child object, where they can be specifically managed. At any point in time you can revert to the inheritance model, in which case the current specific permission levels are discarded.

2.5.4 Security-Trimmed User Interface

In WSS 2.0 every user who had at least read access to a site could see the user interface for all the possible operations on that site, regardless of their level of authorization to the site. Only when they tried to take an option was their authorization level checked: if they were not suitably authorized, an access denied error would occur. This was very confusing to innocent users, who merely tried the options available to them and could not have been expected to know any better.

Fixing this particular issue was not straightforward, and clearly involves extra processing on the server to ensure that only options that the user can take are presented to them. But it's a fantastic leap forward in usability. Consider the previous paragraph on item-level permissions. Now, not only can you now control access to individual items, but the user interface will not even show you items you have no rights to access. And this applies not only to the items in a library but to parts of the user interface as well. Figure 2.25 shows the different user interface displayed to two users, Felix and James. Felix is a member of the task list, and therefore sees the New button. James, on the other hand, is a reader of the task list and therefore does not see the New button. But hang on, James can also see an item that Felix can't—what is that all about? It's because explicit permissions have been set on that individual item to give only James full control. Pretty powerful, huh?

We've now covered the core platform services that are pervasive throughout the various solution areas that SharePoint addresses. In the next chapter, we will examine the collaboration solution area, which will complete the discussion on WSS 3.0, as everything in this area is available solely with WSS 3.0.

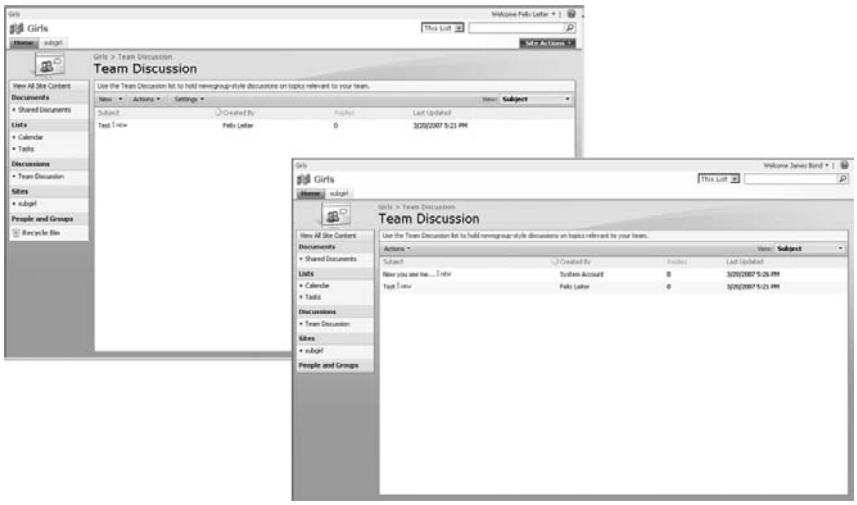


Figure 2.25
Security trimmed user interface.

3

Collaboration

Back in the days before WSS 2.0, the only vehicle that the native Windows operating system offered in terms of collaborating with others was file shares. WSS 2.0 changed that, providing rich sharing capabilities well beyond that of file shares in the form of Web-based team sites and various collaboration type Web Parts. (SharePoint Team Services V1.0 was available before WSS 2.0, but was not part of the operating system. However, it was free, became extremely popular, and essentially laid the ground work for adopting WSS 2.0 into the operating system).WSS 3.0 significantly extends the ways you can collaborate with others and adds some great enhancements to the capabilities you are used to with WSS 2.0. And there is good reason to extend the methods of collaboration—many new and commonly used ways of collaborating that grew up in the consumer world are now seen as crucial ingredients for any enterprise that wants to make the most of its intellectual capital. We're not only talking about the ways we communicate, but also the devices we use—mobile devices are more and more common, as is accessing corporate data from outside the firewall.

So let's take a look at how WSS 3.0 supports these new ways of working and offers a powerful collaboration solution. For the rest of this chapter, we will focus on wikis, blogs, RSS, mobility, e-mail integration, and enhancements to standard libraries and lists. Note that while we may at times mention integration with Office 2007, we will cover this subject in more detail in Chapter 9, Office and Exchange Integration.

3.1 Wikis, Blogs, and RSS (Really Simple Syndication)

Exciting new site definitions are shipping with WSS 3.0 to support modern ways of working using wikis and blogs. Furthermore, WSS 3.0 also allows information in lists and libraries to be consumed using RSS, thereby extending the reach of your intellectual capital. Let's take a look at each of these in turn and highlight their capabilities.

3.1.1 Wikis

What is a wiki? Don't know? Just ask Wikipedia! Oh, hang on, if you don't know what a wiki is, you probably don't know what Wikipedia is, so check out this URL: http://en.wikipedia.org/wiki/Wiki. From that definition you can see that, in computer terms, a wiki is "…a Web site that allows visitors to add, remove, edit, and change content, typically without the need for registration. It also allows for linking among any number of pages. This ease of interaction and operation makes a wiki an effective tool for mass collaborative authoring."

WSS 3.0 has the necessary core platform services to build "an effective tool for mass collaborative authoring" in terms of storage, authorization, and flexibility in displaying content. We have shown some screenshots of a wiki site, but let's take a closer look and dissect the site shown in Figure 3.1 in order to understand how WSS 3.0 meets the business need of "mass collaborative authoring."

Figure 3.1
A Wiki Site

Ultimately each page in a wiki site is a Web page (aspx page) to which anyone with write access can make changes. When you author something in a wiki site, you are creating such a page, and it is being stored for you in a library within the site. Since every item contained within a site has a URL that is within the same namespace of the site itself, you can link to other pages. The URLs for these links point to other pages within the library. Again, this is all done automatically for you.

When you provision a site using the wiki template, the library is pre-created for you with a couple of pages already in it. The site definition for a wiki

site creates a library called wiki pages. This is a major-versioned library that does not require approval before an item is published and does not require check-out as shown in Figure 3.2. This figure shows the document library and a snippet from the ONET.XML file that is part of the site definition (located at C:\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\SiteTemplates\Wiki\Xml). This is the part of the ONET.XML that provisions files into a site, either as standalone pages or in a document library (the Type="GhostableInLibrary" signifies that the file should be placed into a library and a reference to the file is added to the content database). In this case, the contents of the files are built from XML nodes within a language-specific centralized resource file located at C:\Program Files\Common Files\Microsoft Shared\Web server extensions\12\Resources\core.EN-US.resx, rather than being physical files on disk.

The fact that the library is versioned and requires neither approval nor check-out means that users can simply edit a page and make their desired changes. These changes will automatically be live and available for any reader of the site. Reversing the changes made by one user to a page is simple: merely revert to a previous version. Also, identifying those changes in the first place is done simply by comparing two versions, which is possible because WSS 3.0 provides the underlying rich storage for us.

```
<Module Name="DefaultWikiPages" List="119" Url="$Resources:core,WikiWebLibPages_Folder;" Path="" SetupPath="DocumentTempl
  <File Url="wkpstd.aspx" Name="$Resources:core,nav_Home;.aspx" Type="GhostableInLibrary">
    <Property Name="WikiField" Value="$Resources:core,WikiHomeContent;" />
    <NavBarPage Name="$Resources:core,nav_Home;" ID="1002" Position="Start" />
    <NavBarPage Name="$Resources:core,nav_Home;" ID="1010" Position="Start" />
  </File>
  <File Url="wkpstd.aspx" Name="$Resources:core,nav_HowTouseThisWikisite;.aspx" Type="GhostableInLibrary">
    <Property Name="WikiField" Value="$Resources:core,WikiHowTouseContent_Part1;$Resources:core,WikiHowTouseContent_Part2
    <NavBarPage Name="$Resources:core,nav_HowTouseThisWikiSite;" ID="1010" />
  </File>
</Module>
```

Figure 3.2 *Wiki pages in document library.*

Every library can have one or more associated templates, and these templates are launched as starting points for a new item to be placed into the library. Thus, we can create a new wiki page directly from the library, which results in a rich text box being displayed for our content as shown in Figure 3.3. This rich text control is part of WSS 3.0 and allows "What you see is what you get (WYSIWYG)" editing plus simple insertion of tables, hyperlinks, and images. This control generates HTML code, and you can choose to edit the HTML source directly if more sophisticated editing is required.

The result of all of this is that a new aspx page is created in the library and, when viewed, is rendered as a wiki page. We can see this in Figure 3.4. Note that if two users attempt to edit a wiki page concurrently, the last user to save edits will be given a save conflict error and will have to resolve the situation manually. Of course if they want to see the changes the other person made, they can just view the version history.

Now if you take a closer look at Figure 3.3 and Figure 3.4, you will notice that at creation time I inserted some text in double square brackets—such as [[Laphroaig]]. This is how you link to other pages within your wiki, so, for example, if we wanted to link to our newly created page from another page we would type [[Let's talk about whisky]] into that other page at the place we wanted our link to appear. When a wiki page is rendered, if any linked pages do not yet exist they are highlighted with a dotted underline. If you click on the link, a new page to host that link is created in the library—automatically! The net result of this can be seen in Figure 3.5, which is the state of affairs after I clicked the [[Laphroaig]] link in my original page. As you can see, this simple and automatic linking of wiki pages allows you to build a navigable corpus of information in whichever way fits your business need. Note that, by default, the text of a link matches that of the actual wiki page, but you can supply an alternate text for the link by using a pipe—for example [[Laphroaig|The finest of all Scotch!]]. When rendered the link would read "The finest of all Scotch!" and the link would take you to the page called Laphroaig.aspx. Note that you can only use the [[]] notation to link to and create pages inside the current Wiki; you cannot automatically create pages in another wiki site, but you can use regular hyperlinks instead to link to a different site, library, page, or to content outside of your SharePoint farm.

The Edit, History, and Incoming Links options are rendered on all wiki pages allowing you to quickly and easily modify and create pages, to discovering what changes have been made and by whom, to reverting to previous versions of the page, and also to see what other pages are linking to the current wiki page.

A lot of the magic behind a wiki site is contained inside the aspx pages created in the library. These pages have ASP.NET controls in them that gather the content and also display user interface elements, such as the recent changes list that appears in the Quick Launch bar. And where is the actual content? Remember that the pages themselves are stored in a library, and a library has metadata associated with it, including a column called wiki content that takes multiple lines of text. This is where the actual HTML associated with a wiki Page is stored. (If you want the full gory details, then column data is stored in the table called AllUserData inside the content database associated with the Web application that contains your site collection). These pages also have a Web Part zone at the bottom, enabling you to

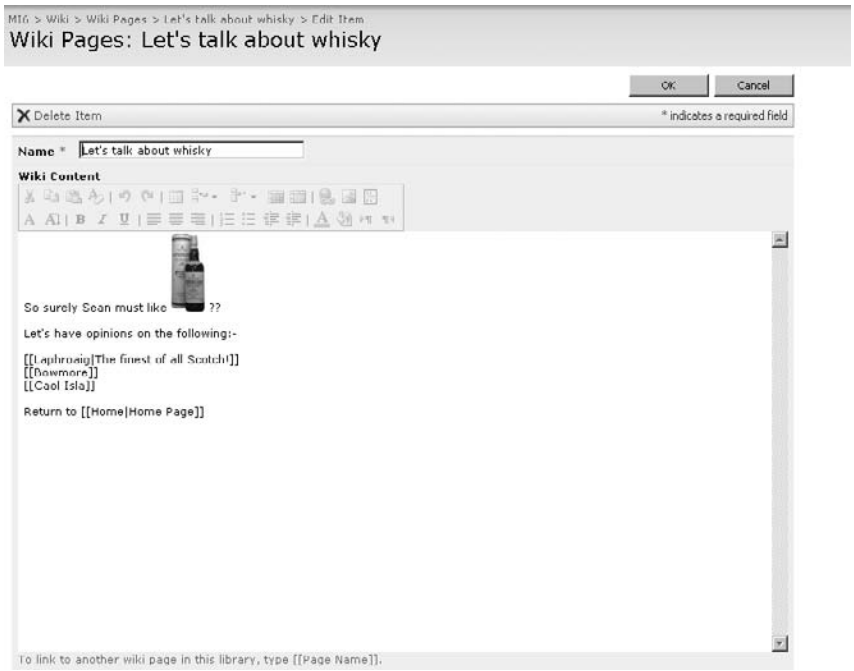


Figure 3.3
Rich text editor for creating pages.

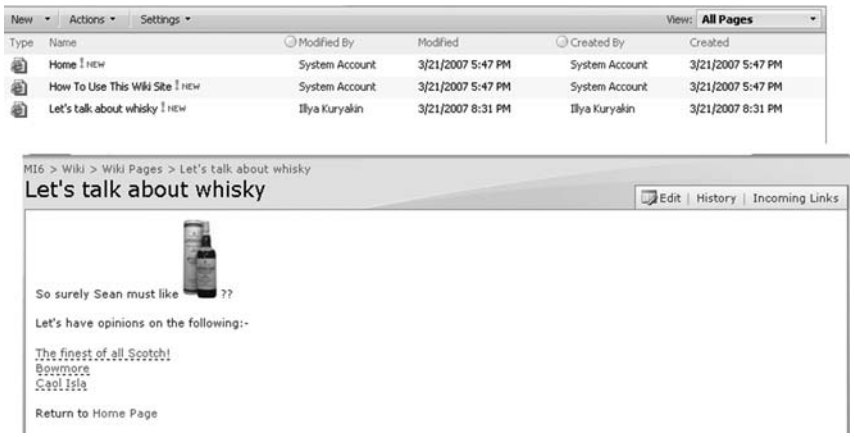


Figure 3.4
Wiki page from library being rendered.

enhance each wiki page with any Web Parts that are available to you. Isn't this powerful? Take a look at Figure 3.6. It's a wiki page that contains a calendar Web Part. Furthermore, since wiki pages are in a library we can extend the metadata for that library to gather extra information—perhaps to categorize the wiki page or to add some detail about the people that are experts in the area discussed by the page. This extra metadata is then displayed at the

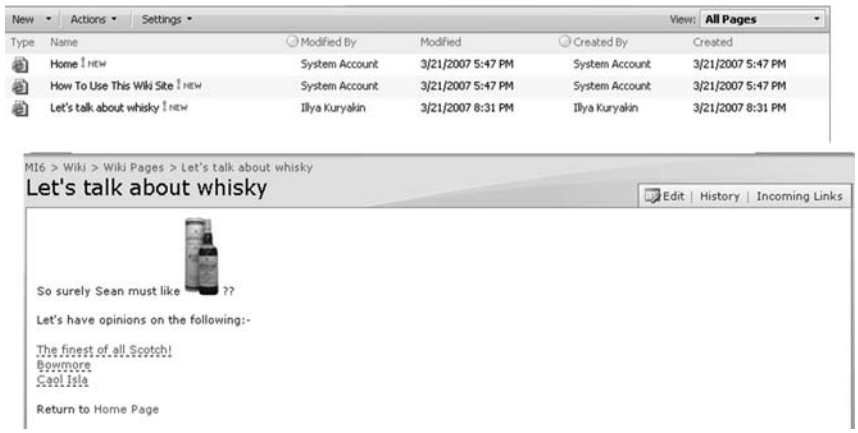


Figure 3.5
Link taken and new page created.

bottom of the aspx page as shown in Figure 3.7, where we have added an Area Expert piece of metadata.

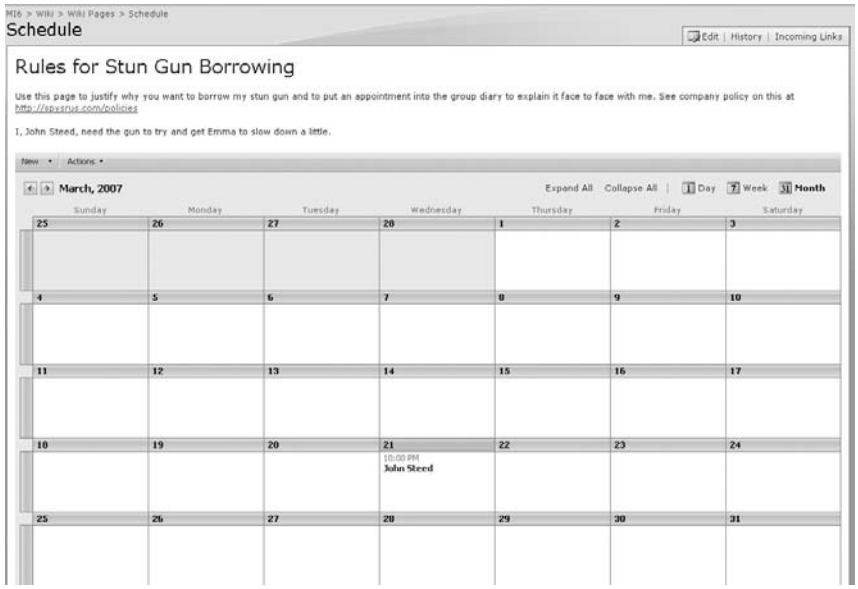


Figure 3.6
Web Parts in Wiki Pages.

Within any site you can create a wiki library; there is no need to create a separate site for your wiki. As described above, the wiki itself is controlled by the items that are in that library; therefore, you can have as many different wikis hosted inside one site as you wish. A wiki site is essentially just a normal site with its home page set to an item in a wiki library!

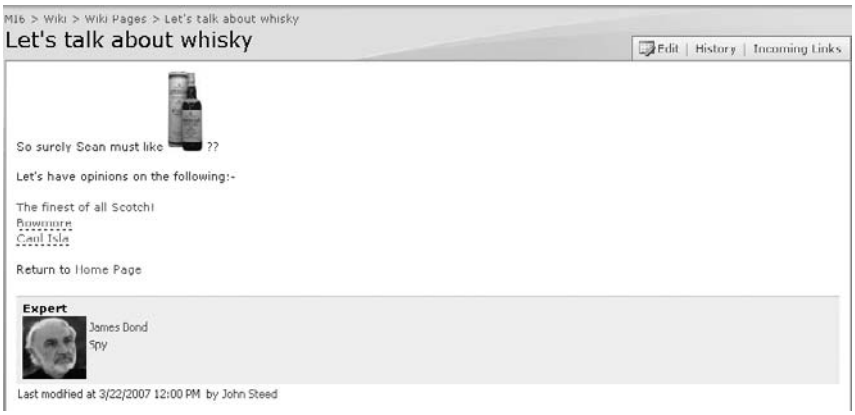


Figure 3.7 *Wiki page with metadata.*

In closing, remember that a wiki is nothing more than a URL to a site with some items in a library, and therefore, other site-based core platform services are available to it, such as master pages, custom columns, item-level permissions (useful if you want to work on a page with a handful of people before releasing it to everyone), alerts, navigation, and search for example. You can search the contents of your wikis in the same way that you can search normal document libraries and lists.

3.1.2 Blogs

Blogs are basically Web-based diaries where people publish information on many different subjects. Other people can then link to that information and benefit from it, even adding their own comments to the subject at hand. As we will see, their implementation is very similar to that of wikis, and they build upon (and benefit from) the core WSS 3.0 platform services.

Let's dissect the blog site shown in Figure 3.8, which is an out-of-the-box instantiation of a site using the blog template. It should come as no surprise to you now (at least if you've gotten this far in the book I hope it isn't!) that all the content that makes up this blog site is contained in lists and libraries as we can see in Figure 3.9. The quick launch bar takes its content from the Categories, Other Blogs, and Links lists. Categories is a single-column list and the other two are just straightforward link lists with each item containing a URL, description, and notes.

The Photos link is to a standard picture library, and the two archive links actually just show different views on the core content, which is held in a list called Posts. The items in this list are the individual posts to the blog, and the content of each post is held in a multi-line string column called Body. There is also a column called Category that is a lookup field on the Categories list ensuring that posts can only be allocated to valid categories.

The Archive (Calendar) link allows you to see at a glance a calendar indicating what posts were posted when. This is a common facility on commercial blogs. Indeed, most of the common features you would expect to find on a blog are available (comments, permalinks, date navigation etc.). The Posts list requires, by default, content to be approved before it is shown to readers of the site. Thus, as authors create content, they must take an action to move it from a draft state to a published state.

Comments are stored in the Comments list. The Comments list has a lookup column that validates against the Posts lists and links to each post.

Figure 3.8
Blog site.

When you create a post through the browser, you are presented with the rich text control as is the case when editing a wiki. On the create page, you have the option of categorizing the post and, if you are suitably authorized, you can choose to either save it as a draft or publish it immediately. If you do not have the right to approve, then you can only save as a draft. There are other ways to create a post and, indeed, to consume posts, since a SharePoint blog site supports the MetaWebLog API (the MetaWeblog API—see http://www.xmlrpc.com/metaWeblogApi for details—is a programming interface that allows external programs to get and set the text and attributes of blog posts). And guess what? Word 2007 supports publishing to a MetaWebLog supported site, so if the editing capabilities of the rich text control are not sufficient for your needs, you can use the full power of Word 2007 instead.

Note that you can control at the Web application level whether the MetaWebLog API is supported or not.

The content of a blog is usually displayed in reverse chronological order, and it is the default.aspx—which is set as the home page for the site at creation time—that handles the display of the content on the home page. This, and other supporting pages, is a Web Part page, and the Admin Links Web Part is only shown to users who have at least Member access to the site. As

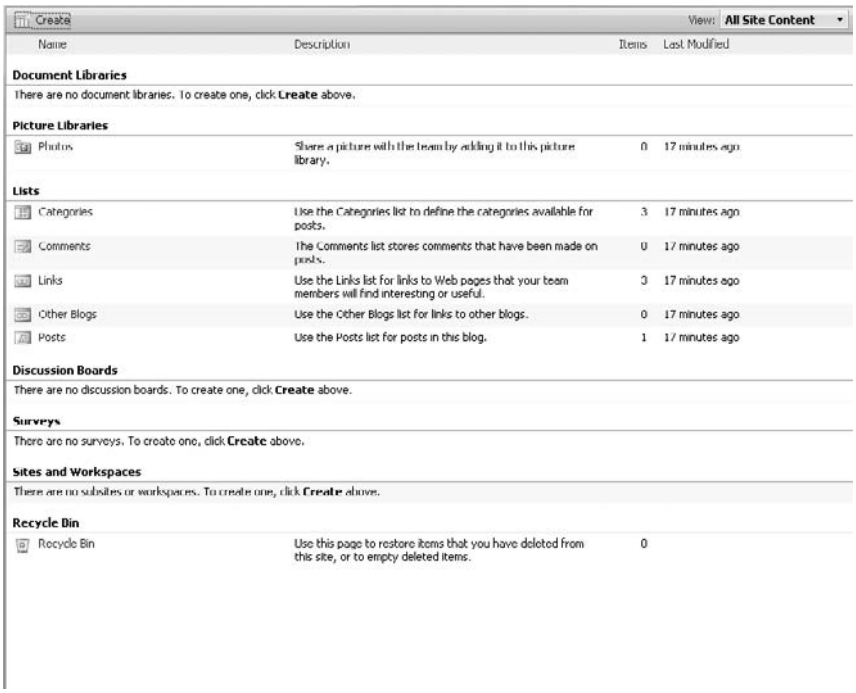


Figure 3.9
Blog site content in lists and libraries.

this is a Web Part page, you can add more parts and personalize your own copy of the page. We can see our blog being updated with a new post and a comment in Figure 3.10; note the ability to e-mail the body of a post via your registered e-mail client. Also, note that we have changed the out-of-the-box categories in this picture. Lastly, note the permalink. This is a link to a URL that points to a specific blogging entry and remains unchanged indefinitely, even after the entry has passed from the main blog into the blog archives. It points to the specific item id in the Posts list—for example, our last post in Figure 3.10 has a permalink of http://mi6.spysrus.com/blog/Lists/Posts/Post.aspx?ID=3.

3.1.3 RSS

Really Simple Syndication references a format used to publish frequently updated digital content, such as blogs, newsfeeds, and podcasts. It enables the users to subscribe to these updates, commonly known as *feeds*, and to have the latest information brought to the user and typically displayed through an *aggregator* or built-in extensions to e-mail and browser clients. There are many aggregators out there, such as Newsgator and Straw, and Outlook 2007 has an RSS aggregator built into it. Aggregators subscribe to the feed and automatically update the local content, allowing offline viewing.

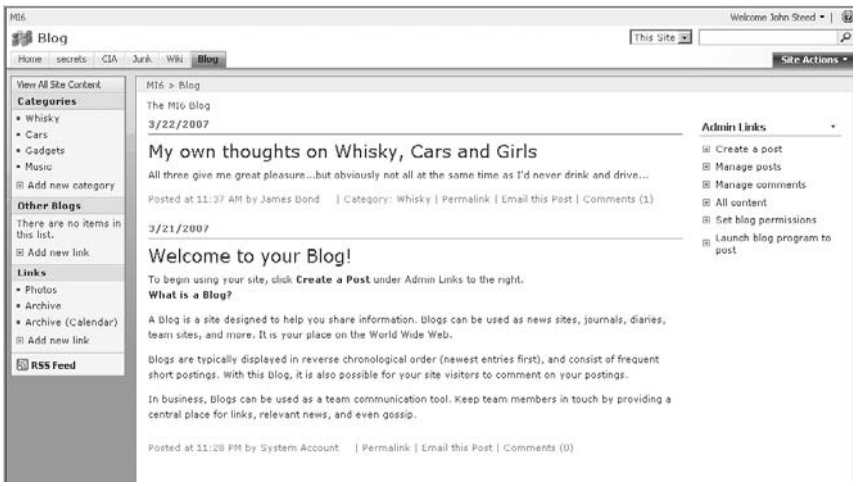


Figure 3.10 *New blog post and comment.*

There are different feed formats available; one of the more popular ones is ATOM (the Atom Syndication Format).

The feed itself is an XML file that contains details about the latest items to be published. WSS 3.0, in the spirit of collaboration, offers the ability to generate RSS feeds for lists and libraries. RSS has to first be enabled at the Web application level by a SharePoint Administrator. Next, global details such as whether RSS feeds are supported within the site collection, copyright notice, WebMaster contact, and time-to-live value (i.e., the recommended time that an aggregator should wait between checking for updates) is defined at the site collection level. Once this is enabled, you can then switch on feeds for individual sites and then for the lists, libraries and views contained within. You specify which columns should be contained in the feed. Libraries also support enclosures, meaning that documents within them can be contained within the feed itself. We can see all of this in action in Figure 3.11 and Figure 3.12; the latter shows Outlook 2007 being used as an RSS aggregator. End users subscribe to the feed by taking the View RSS feed option from the Actions menu, which then shows the actual feed itself in a separate browser window. From here there is a link to subscribe to the feed.

The XML that represents the feed is generated by the page called listfeed.aspx, which takes the id of the list and the id of the view for which the feed is being requested. So, for example, our RSS subscription for Outlook in Figure 3.12 is actually set to http://mi6.spysrus.com/_layouts/listfeed.aspx?List=84ecaf37-dbeb-4852-8659-0cc77c31ad6c&View=90301816-988e-4090-bd74-46f88f5ae616. You can generally find listids by taking a close look at the URLs you are navigating within a site!

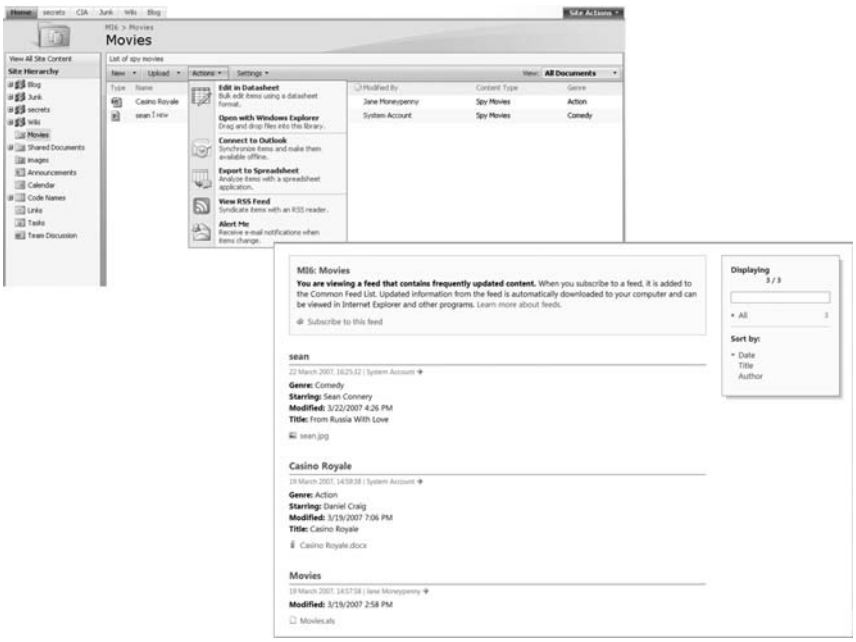


Figure 3.11 *Viewing an RSS feed.*

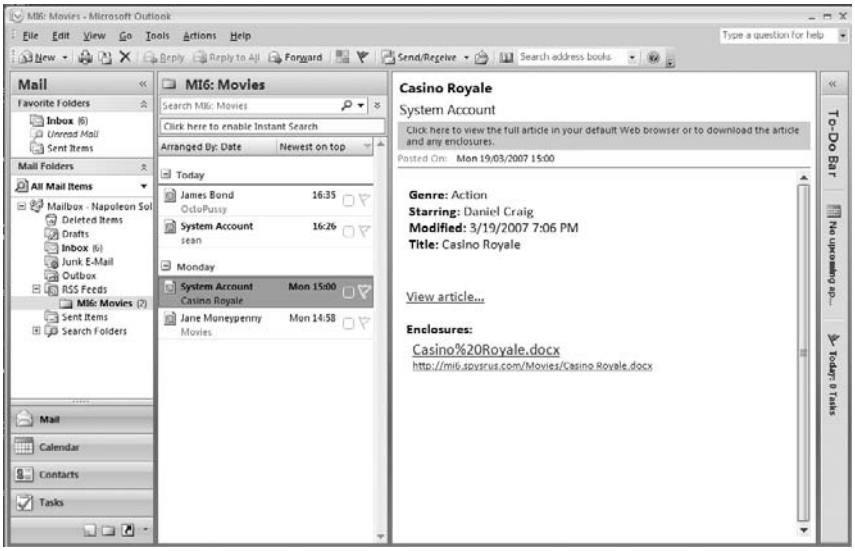


Figure 3.12 *Outlook 2007 as an RSS reader.*

3.2 Mobile Access

"Peripatetic" is a word we use to describe the fact that most users these days wander around, rather being tied to an office desk. But as they wander, they

still need access to information using whatever device they have at hand. More and more people have intelligent handheld devices these days, such as SmartPhones and personal data assistants, which are generally small with limited viewing real estate.

Any smart device that can run a browser can access a SharePoint site, but can you envisage what a full-blown site would look like on a small device? Can you imagine trying to interact with it? Your thumb would become quite tired within the space of a few minutes from scrolling around, and frustration would set in very quickly!

SharePoint attempts to connect people with information, but it also recognizes the capabilities and constraints of mobile devices. It therefore provides features that allow *appropriate* information to be delivered to mobile devices in the form of device-friendly pages. How is this done? SharePoint utilizes mobile controls from ASP.NET 2.0 (in the System.Web.UI.MobileControls namespace) to render HTML that is suitable for mobile devices. Most sites have mobile support, but there are some exceptions—for example, there is no mobile support for a wiki site. There are mobile home pages that are specific for the type of site we are dealing with. The home pages are physically located in C:\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\LAYOUTS\MOBILE, where you will find aspx pages such as mbllists.aspx (team site) and bloghome.aspx (blog). Users could just navigate directly to these homepages from their mobile devices, but that would not be very user friendly, as the URLs would be difficult to remember and to type in. Therefore, WSS 3.0 supports a simple way of re-directing to the appropriate page by adding a /m to the standard URL for a site. Therefore, if my blog site is http://mi6.spysrus.com/blog, then I can access the mobile version of this site by using http://mi6.spysrus.com/blog/m, seamlessly redirect me to http://mi6.spysrus.com/blog/_layouts/mobile/bloghome.aspx. We can see this in action in Figure 3.13, is a screenshot from an HP iPaq 510 Voice Messenger device which is a SmartPhone. This shows our blog site being accessed; note how we can filter the view with our categories and add new posts and comments. The screenshot shows the home page, a blog, and a comment.

For a team site, the mobile home page is a list of those lists within the site that have mobile views associated with them. You can mobile-enable most views by modifying the view and enabling it in the Mobile section, as shown in Figure 3.14.

When you access a document library on a mobile device, you have the option of downloading the actual document and viewing it, should you have an application on your device that can do so. We can see this in action in Figure 3.15, which shows a Word document in a library being viewed by Word Mobile. Note the custom metadata (Genre, Starring) being displayed from the Movies document library in the middle picture.

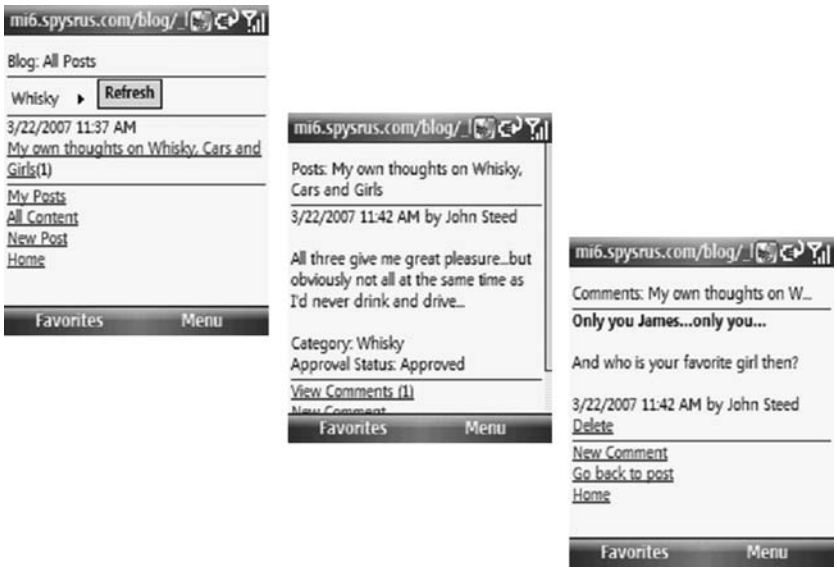


Figure 3.13
Blogs on a mobile.

Figure 3.14
Mobile enable a view.

Figure 3.15
Viewing a document from a library.

3.3 E-Mail and WSS 3.0

3.3.1 Outgoing E-Mail

WSS 3.0 leverages e-mail in two ways—outgoing and incoming. Outgoing e-mail is pretty straightforward and, depending on the operation at hand, will either call the local client's registered e-mail client (which will in turn send the e-mail) or will connect to an SMTP server to do the job. For example, alerts and administrator notifications are sent via an SMTP server, whereas if you take a "Send a Link" option on an item, it will fire up your local e-mail client and pre-create the message. Configuring outgoing e-mail settings is a farm-wide exercise done through Central Administration.

So much for outgoing e-mail—it's not that different from WSS 2.0. Incoming e-mail was possible in WSS 2.0 for document libraries, but required the assistance of a mail-enabled Exchange 2000 (or 2003) public folder. You would mail-enabled a document library by pointing it at the public folder, and a timer job would use WebDAV to connect to the public folder and extract any e-mail. Attachments to e-mail in the public folder would be created as documents in the library. While this worked, it was non-exhaustive and had many implementation difficulties.

3.3.2 Incoming E-Mail—Now We're Talking!

Enter WSS 3.0! Now we can mail-enable lists, libraries, and SharePoint groups! This is important for quite a few reasons not least that e-mail is a very common way of collaborating within a team—I'd be very surprised if anyone reading this book had never used a distribution list to communicate within a team, and it's highly likely that something like a mail-enabled public folder is used to hold the archive of such communications. Microsoft has publically stated that its current plan is to eventually stop supporting public folders (we are talking at least ten years here!), and indeed, you can run Exchange 2007 today with no public folders, assuming you have no pre-Outlook 2007 clients that want to access the Exchange Server. And it is no secret that SharePoint is being positioned as a vehicle that could ultimately be used instead of public folders. But used for what purpose? Well that depends on how you use public folders, and there are many and varied uses. Public folders will be around until Microsoft can provide an adequate replacement for all the functionality they provide. We will look later at whether SharePoint can replace public folders as a document repository, but for now let's look at how incoming e-mail works and then we'll comment on whether it can replace the public folders' ability to act as an archive for e-mail–based team discussions.

WFE servers have the ability to monitor a folder and look for SMTP-formatted messages deposited into that folder. If any mail is found, it is pro-

cessed and matched to a list or library available on that server farm. Depending upon various configuration options—such as who is allowed to send mail to a particular list and the type of list it is—the e-mail and/or attachments will be added as an item to that list. Once there, it is a normal list item and therefore benefits from all the core services such as search, views, etc.

For all of this to be successful, you have to configure various things such as how and where the WFE servers monitor the folder, how to match incoming e-mail addresses to lists and libraries, and how to control who is allowed to do what via e-mail. Let's take a look at these three areas.

Configuring incoming e-mail settings

Incoming e-mail has to first be enabled at the farm level via the Operations tab in SharePoint Central Administration. Once it is enabled, a background timer job called Windows SharePoint Services Incoming E-Mail will run every minute on all WFE servers and poll the drop folder. As with all timer jobs, this is actually controlled by the Windows SharePoint Services Timer service, so if it is not running then incoming e-mail will not be processed. The schedule for the timer job is fixed at one minute, and we know of no way to change the frequency of this.

The incoming e-mail job runs by default on all WFE servers, yet the configuration for the drop folder is set farm-wide. What this means in practice is that all WFEs in a farm that are running the incoming e-mail service must support the same drop folder, so if it is configured as C:\Drop, then all WFE servers must have a C:\Drop, and your mail routing topology must ensure that incoming mail gets deposited here for all WFE servers supporting incoming mail. This will generally be controlled via your overall SMTP routing infrastructure, which we will discuss in just a moment. Whether you need more than one incoming e-mail job running (and therefore by implication have incoming SMTP messages balanced over multiple WFEs) will depend upon your volume of incoming traffic.

It is anticipated that you will enable and configure the Windows SMTP service on your WFE servers to receive e-mail from other SMTP servers—for example, to be able to send messages directly from Exchange 2007 into WSS. If so you can just tell WSS to use the server's SMTP settings to figure out the location of the drop folder that it should monitor. Note that it can only monitor the local SMTP service; it is not possible to point it to a remote SMTP server. This is called automatic configuration and will meet the needs of some organizations. However, you may not be using the Windows SMTP service and may want to populate a completely different drop folder via some other mechanism, in which case you need to manually configure the settings using Advanced mode. Doing so reveals the ability in the user interface to configure the drop folder, but note that it disables the ability to specify which

SMTP servers are considered "safe." In automatic mode, you can indicate that mail coming from any server is acceptable, or you can specify a list of IP addresses that equate to the SMTP servers that you are happy to process mail from. What you do here will depend upon your overall SMTP routing topology. For example, you may have one central SMTP server that all mail passes through, and it is the only SMTP server from which you are willing to process mail. The incoming e-mail service will look at the SMTP headers inside the message that it picks up from the drop folder and analyze the "Received:" header to determine whether to accept the message or not.

One last note here: if you are not using automatic settings, you will have to ensure that the account used to run the Windows SharePoint Services Timer service has modify permissions on the specified drop folder so it can delete the messages after processing. Failing to do so will result in messages being delivered multiple times.

The last piece of configuration for this stage is to indicate which SMTP domain your WSS farm will process mail for. This is the piece of information on the right-hand side of an e-mail address after the @ sign—for example, for an e-mail address of mi6@wss.spysrus.com the SMTP domain is wss.spysrus.com. Ultimately, in this example, any list or library that is e-mail enabled will have an address of <something>@wss.spysrus.com. Therefore, your challenge from an infrastructure point of view is to ensure e-mail addressed to wss.spysrus.com routes its way to a WFE that has the incoming e-mail service enabled. There are many ways to do this, and it will more than likely involve setting up MX records in DNS and connectors from other internal mail systems. For example, in Exchange 2007, you could set up a Send Connector that handled the wss.spysrus.com namespace and indicate that you wanted all e-mail for that address space to be forwarded to a smart host—the smart host being your WFE server.

Enabling delivery to a list or library

Once e-mail is configured at the farm level, site administrators can enable e-mail on individual lists and libraries. Not all list types can be enabled for mail, but for those that can, you will see an incoming e-mail settings option under the Communications section when viewing the settings for the list. The settings required will depend upon the list type; for example a document library will ask you what you want to do with attachments, and a blog post list will allow you to choose whether the incoming item is published immediately. You can see the settings for a document library and blog post list in Figure 3.16.

What you will notice is that every list you mail-enable needs to have a unique e-mail address across the farm, and all the enduser can choose is the user part of the address. The domain part will be as configured at the farm

level, and the e-mail address will be stored in the AllLists table within the content database associated with the site collection. We will discuss later the implications of uniqueness when the Directory Management Service (DMS) is introduced into the mix, since this widens the scope for uniqueness to an Active Directory (AD) forest.

I question whether this way of allocating e-mail address is practical in any reasonably sized organization. Why? There is no governance at all, so it's a first-come, first-served situation with respect to the actual e-mail addresses. And it is highly likely that people will want to use common names for common lists. For example, what would you want the e-mail address to be for your Announcements list? Announcements@domain.com? So would everyone else, and there is no way to check what other e-mail addresses are currently in use across the farm other than by trial and error. So this in itself will lead to end-user frustration. There is also nothing to validate that the e-mail address conforms to any corporate policy in terms of formatting or use of invalid/inappropriate names.

There is also a whole boatload of issues that will occur should you want to move content databases between farms, such as in a merger or acquisition scenario or even, for normal operational purposes. Remember, the e-mail address only has to be unique across the farm, but what if you have more than one farm? What if you move a content database from one farm to another? Not only will the SMTP domain part of the email addresses change for the imported lists, but there is a high likelihood of conflicts occurring. In fact, what WSS does when you import a content database is to remove any e-mail addresses from lists that are not unique within the incoming farm—so they just, disappear into thin air! Clearly this is not an ideal situation, and some deep thought will need to go into the operational aspects of e-mail enabling lists and libraries.

Another issue springs to mind: there is no way for the sender of an e-mail to know whether the message was successfully delivered to the right SharePoint list (if indeed it was delivered at all) except by physically visiting the supposed destination site. The SMTP service initially receives the incoming e-mail, and it is not tied to SharePoint in any way and therefore cannot validate that the e-mail address does point to a list or library. As a result there is no way to generate delivery failures for undelivered messages and no way for things like read receipts to validate where the message was delivered.

The last aspect to mention at this stage is the security you can set for incoming e-mail. You can decide to only accept mail based on the permissions set on the list or library itself or from any sender.

We've now talked about how to set up e-mail addresses for lists and libraries. But raw e-mail addresses in themselves are not very user friendly, and most e-mail systems offer some form of address lists where e-mail addresses can be looked up using more user friendly terms like firstname and

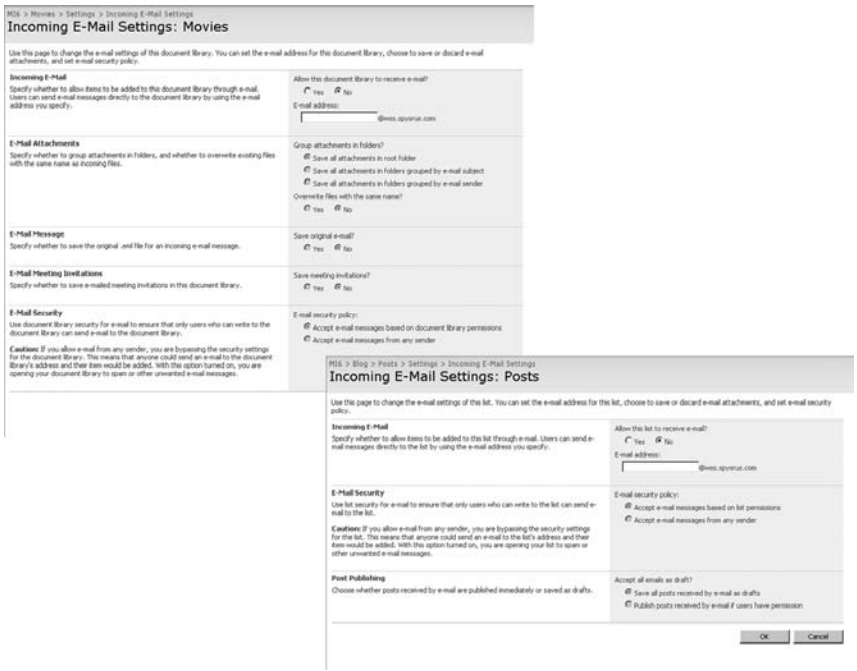


Figure 3.16
Mail enabling a document library and a blog post.

lastname. WSS does have the ability to leverage the AD (or indeed a custom directory) for this very purpose via something called the Directory Management Service (DMS). Additionally, the DMS is required to facilitate the e-mail enabling of a site itself, resulting in distribution lists being created to house the individual members of a team site. Let's now see how the DMS works and what problems it solves—and introduces into the mix.

3.3.3 Directory Management Service

The DMS is a Web service (SharepointEmailWS.asmx) installed out-of-the-box on a WFE server. It is enabled at the farm level using the incoming e-mail settings, and you can choose to use the out-of-the-box web service or point to a remote one. The latter option could be applied if you want to write your own Web service that supports the same methods as the out of the box one but perhaps uses a directory other than the AD.

Figure 3.17 shows the methods that the Web service supports; you can see that they are all related to contacts and distribution groups. The job of the DMS is to create contact objects in the AD to represent e-mail enabled lists and libraries and distribution groups to represent the members of a team site. In this way, your e-mail system should be able to benefit from these entries and offer endusers the same address lookup functionality and

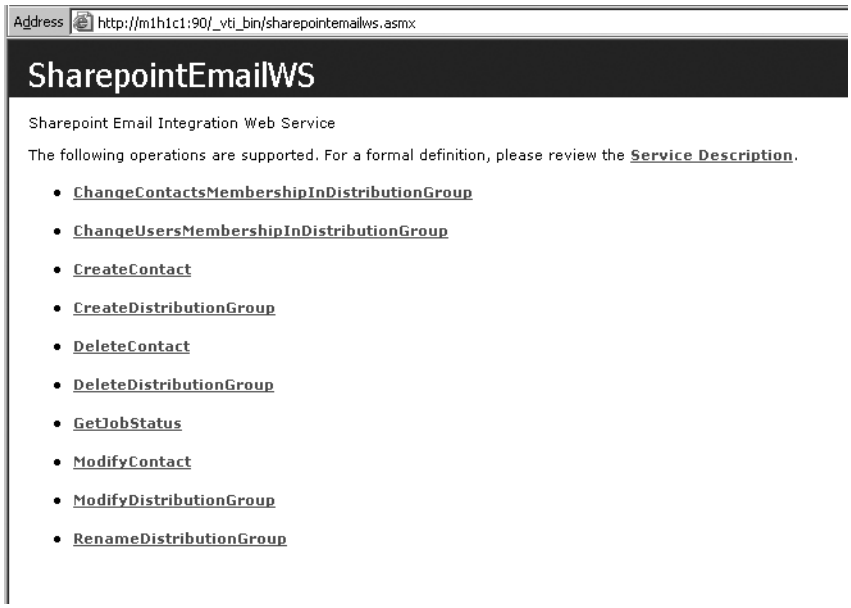


Figure 3.17
The DMS and its methods.

e-mail distribution as with other e-mail enabled objects. I say "should" because there are some considerations to be aware of to fully understand the implications of enabling the DMS.

Configuring the DMS is a matter of specifying an organizational unit (OU) in the AD where the contacts and distribution groups will be created. For this to work, you need to provide the SharePoint Central Administration application pool account with write access to the OU; therefore, you will have to liaise with the people who are in charge of your AD. You must also specify the name of the SMTP server for incoming mail; ultimately, this will find its way onto the e-mail addresses associated with the contacts and distribution groups that the DMS creates. Note that this can be different from the incoming e-mail server display address, which is also set during incoming e-mail configuration. This latter address is the more user-friendly one displayed through the user interface to endusers, and you may have to subsequently update the AD-created objects with this address for it to be a valid address to which mail can be routed. You can also indicate whether the objects that are created will only receive mail from authenticated users. This is an Exchange-specific setting, so if you are using some other form of e-mail system, this setting may not be applicable.

Let's take a closer look at the Contact and Distribution Group objects that are created, and then we'll discuss on the implications that can arise from this.

Contacts

Aside from standard system type attributes, the main attributes set on a Contact object by the DMS are listed in Table 3.1.

Table 3.1

AD Attribute	Taken from
cn	The name part of the e-mail address for the list specified by the user
displayName	The title of the site the list belongs in, appended with the title of the list itself
givenName	The title of the site the list belongs in
mailNickname	Same as cn
msExchRequireAuthToSendTo	True or False, depending upon DMS configuration
name	Same as cn
sn	The title of the list
targetAddress	The name part of the e-mail address for the list specified by the user-followed by the incoming mail server specified during DMS configuration

There are a few issues with the above settings. Not least is the fact that, in a pure Exchange 2007 environment, these settings are insufficient for a valid entry to appear in the Global Address List (GAL). The reason is that in Exchange 2000/2003 environments, there is an asynchronous component called the Recipient Update Service (RUS) that stamps the necessary attributes on AD objects—for example, details like proxyAddresses and showInAddressBook—Exchange relies on to identify recipients and route mail. This asynchronous method of stamping addresses has been removed from Exchange 2007, and all attributes are stamped at creation time—but only if you use the Exchange 2007 Management Console or Shell to create your objects in the first place!

Therefore, to get your contacts correctly stamped in an environment that does not have a RUS, you need to fix them up after the event. This is discussed in the Exchange team's blog at http://msexchangeteam.com/archive/2006/10/02/429053.aspx. Essentially, you need to update all relevant address lists by running the following Exchange Management Shell cmdlets:

```
Get-EmailAddressPolicy | Update-EmailAddressPolicy
Get-AddressList | Update-AddressList
Get-GlobalAddressList | Update-GlobalAddressList
```

The second issue with the Contacts is that they are not suitably configured for attachments to be correctly sent to them. This is documented at http://support.microsoft.com/default.aspx?scid=kb;en-us;926891, and it explains how you have to set the mapiRecipient attribute to *false* and the internetEncoding attribute to 1310720 before attachments will be correctly sent to the Contact object. Note that if you are NOT using the DMS and just typing in the e-mail address of your list or library directly into e-mail messages, then these types of issues do not occur!

The third issue is to do with the display name that is created and ultimately finds its way into the GAL. This is usually the primary vehicle by which someone chooses an entry from the GAL. However, it is highly likely that users will have very similar list and site names, which in turn, will mean that there is little to differentiate entries in the GAL. (There is nothing in the user interface to stop sites and lists having the same titles.) Add to that the length of the overall display name and you begin to see that the existence of Contacts in the GAL may be a hindrance rather than a help. Figure 3.18 shows a sample of this: two team sites called Group Team Site with the out-of-the-box Team Discussion list mail enabled. Which one should a user chose? To know which one to pick, they'd need to know the e-mail address of the appropriate list—which would defeat the purpose of having a GAL entry in the first place, wouldn't it? You can also see that there is no way for a team site owner to populate other attributes in the contact that may help in its identification (such as location, title, etc.); they are basically stuck with just the site title, list title, and the user part of the associated e-mail address.

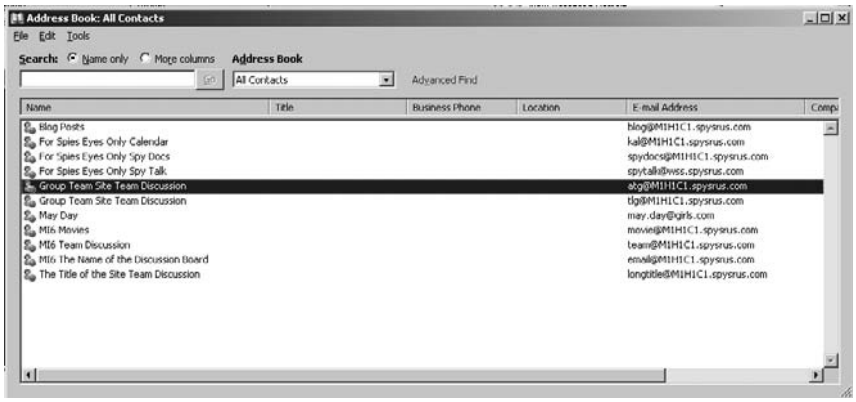


Figure 3.18 *Contacts in the GAL.*

A related issue here is the scope of uniqueness for the actual e-mail address. When the DMS is disabled, the scope is the SharePoint farm, but when it is enabled, the scope changes to that of the AD forest. This should not cause any more issues than that described earlier, since the domain part of the e-mail should be specific to the SharePoint farm, but it is something to remember. Note that, once enabled, the DMS does its work at the time a list or library is mail enabled or subsequently modified. Therefore, if you already have e-mail enabled lists and libraries and then switch the DMS on, you will have to modify all those existing lists and libraries for the DMS to generate Contact objects for them.

The last issue I'd like to mention is associated with the previous one and that is lack of governance. There is no administrative control over what gets put into the GAL—once you decide to enable the DMS, it's a free-for-all and chaos can certainly reign. Can you imagine if every single team site e-mail enabled, say, three lists? How many entries in the GAL would that equate to in your organization? We can tell you that in HP, that would be reaching the million mark. And, given the design of the DMS, all of these entries would be in the same OU within the AD. We're not sure we know of any AD administrator that would happily open up their AD to the potential for such abuse.

Distribution Groups

While Contacts are optional for e-mail to be sent to a list, Distribution Groups have to be used if you want to e-mail enable a SharePoint group. E-mail enabling a SharePoint group results in a Distribution Group object being created that holds the members of the SharePoint Group and, optionally, a Contact object that is used as an archive of the e-mail that is sent to the group. This is one way to emulate a typical use of an e-mail enabled public folder.

While there is no governance on the creation of Contacts, you can configure the DMS to require approval before a Distribution Group is created or modified. This approval is done at the farm level by a farm-level administrator using the Operations tab of the Central Administrator program.

You can mail enable any SharePoint Group, which ultimately leads to a Global Distribution Group object being created in the DMS-configured OU. The cn, mailNickname, and name attributes are set to the user part of the e-mail address and the displayName is set to the SharePoint Group title. As security principles are added to the SharePoint Group, they are also added to the AD Distribution Group as a member. Additionally, the owner of the SharePoint Group can add a mail enabled list as a member so it can act as the archive.

Similar issues exist with regards to an Exchange 2007–only environment as to Contacts, and you need to do the same work before the Distribution Groups appear in the GAL. Additionally, Exchange 2007 will only create

Universal Distribution Groups for receipt of mail. You should convert the Global group to a Universal one using either the Exchange Management Console or the Exchange Management Shell.

As with Contacts, endusers are empowered to basically create groups with any display names they like, which may lead to confusion in your GAL. Also note that WSS, unlike Exchange, does not check the e-mail validity of a member being added to the underlying Distribution Group, and therefore, it is entirely possible to have user objects in the group that are not mail enabled, which will result in delivery failures when e-mail is sent to the group.

Other WSS 3.0 list enhancements

For the last part of this chapter, we are going to take a very quick look at some of the more general enhancements we see in WSS 3.0 lists.

Discussion lists (also known as discussion boards) have been enhanced to support better communication among team members. They support threaded views and the ability to show or hide the text to which a reply relates. E-mail enabling of team discussion lists is possible, so members can contribute through e-mail, through the browser or, as we will see later, through Outlook 2007. Indexing allows the contents of discussion lists to be easily searched.

A couple of new list types are designed for project and task management—not full-blown Project Management, but a "lite" version that allows you to keep up to date with various objectives. The main feature of this type of list is the ability to have a graphical view of all the tasks and their progress, just like a Gaant chart. Project task lists cannot be mail enabled, but they can be edited in a datasheet view and also exported to Excel.

The issue tracking list allows you to assign, link, and categorize issues and can be useful for something like a ticketing or helpdesk system. It cannot be mail enabled, but it can be edited in a datasheet view and is exportable to Excel.

Surveys have been enhanced to allow for the answer type to be a person/group and also to allow for page separators in the actual survey. Branching is now supported, allowing you to skip certain questions based on the response to a previous question.

This page intentionally left blank

4

Microsoft Office SharePoint Server 2007

We discussed in previous chapters how WSS 3.0 provides platform services and a collaboration solution. Microsoft Office SharePoint Server 2007, to give it its name in all its glory, is a complete superset of WSS 3.0. Oftentimes referred to as MOSS, it extends the solution areas and brings its own shared services to the table that these solution areas then consume. You can either upgrade an existing WSS 3.0 installation or deploy a new MOSS installation in which case WSS 3.0 will be installed for you.

What are the new solution areas that MOSS introduces? We gave a broad overview of them in Chapter 1, 2007 Microsoft Office System, but take a look at Figure 4.1. This is the same basic figure we used in Chapter 2, Windows SharePoint Services V3.0, to describe the functional areas that WSS 3.0 delivers extended with those of MOSS. Separate chapters will cover all the functional areas, so this chapter will focus mainly on the key generic additions you get with MOSS and, in particular, the Business Data Catalog, as this is a shared service that can be exploited in many different ways to ensure your users are connected to important business data that is typically locked in back-end enterprise applications.

4.1 Shared Services

SPS 2003 introduced a Shared Services model which, we think it is fair to say, was not fully baked. It did solve a couple of problems, but there were various implementation issues that meant it was not embraced as widely as it could have been.

But it was a good idea, and we are happy to say that its MOSS implementation is far superior. The idea behind Shared Services in SPS 2003 was to provide a central place that provided key services and to have multiple portals consume these shared services. You could share search and indexing, My Site and Alerts, Audience Targeting, People Profiles, and Single Sign-On. The implementation meant that you designated one portal as the parent and then all other portals created on that farm became child portals that con-

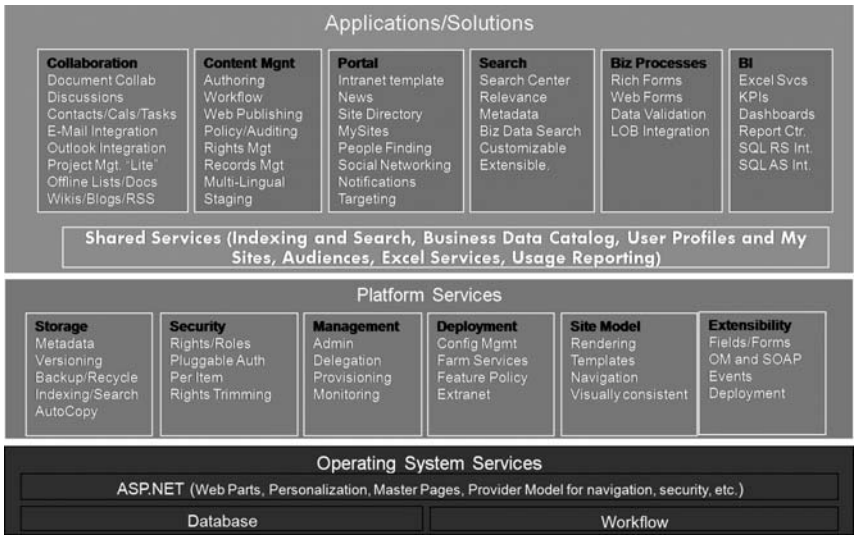


Figure 4.1 *MOSS 2007 solution areas.*

sumed the shared services. And there was no way to switch off shared services once you had enabled it other than a re-install!

Shared services are configured at the farm level and are consumed by the Web applications and site collections running on that farm. Furthermore you can have inter-farm shared services, which allow, for example, a single My Site across any number of site collections, regardless of which farm or Web application is hosting that site collection. Shared services themselves need to be managed; you will need to create a Shared Services Provider (SSP), which will have an associated Web application for its administration. Separate content databases are also created to hold the data associated with each service—for example, a search database. The majority of implementations will only require one SSP, but you can actually create multiple SSPs and scope them to individual Web applications should you need to—such as in the case of a hosted environment, or when you need secure isolation of services and service data.

The major shared services that are provided with MOSS are Indexing and Search, User Profiles and My Sites, Business Data Catalog (BDC), Audiences, Usage Reporting, and Excel Services.

4.2 MOSS extensions to WSS 3.0

4.2.1 Administration Additions

Additional solution areas bring additional administrative responsibilities in overall farm management, individual applications, and even individual site

collections. MOSS adds more components to tier 2 of the architecture—for example, an updated Search, Indexing, and Excel calculation server—and these have to be considered when you are designing your topologies from a farm-wide perspective. Another farm-wide addition is the area of Information Life Cycle Management (ILM), such as the ability to apply policies to document libraries and to audit their usage. We will take a closer look at these ILM capabilities in Chapter 5 on Content Management.

At the application level, we see additions for managing Search, InfoPath Forms Service, and the MOSS Shared Services. One thing I haven't mentioned so far is that the whole administration model is modular, enabling other applications to come along and plug in nicely. Proof of this can be seen in Figures 4.2 and 4.3, which show the Operations and Applications tab from Central Administration before and after a WSS 3.0 installation is upgraded to MOSS. You will also see the management of SSPs integrated into Central Administration.

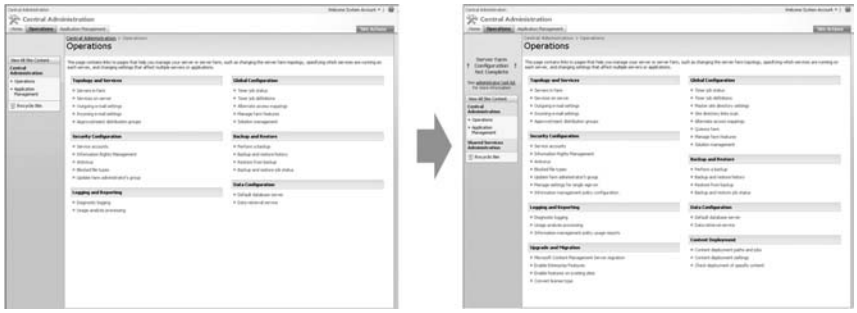


Figure 4.2 *Operations in WSS 3.0 and MOSS administration.*

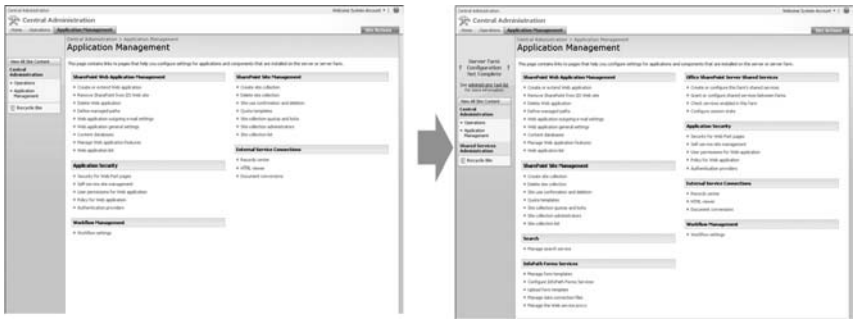


Figure 4.3 *Applications in WSS 3.0 and MOSS administration.*

4.2.2 MOSS Permissions and Groups

No extra user permissions are added when you install MOSS, but there are many more permission levels and SharePoint Groups created to make it easier to control the different types of activities that would be performed in the

different types of MOSS sites. For example, if you have a Web publishing site, you will want to control who can design and publish pages. For this purpose, we see groups such as Designers and Quick Deploy Users being introduced to sites that have the publishing feature enabled. Even in a MOSS team site, the number of groups and permissions levels are extended quite considerably, as shown in Figure 4.4. The permission level additions here to a WSS 3.0 team site are Manage Hierarchy, Approve, Restricted Read, View Only, and Records Center Submission Completion.

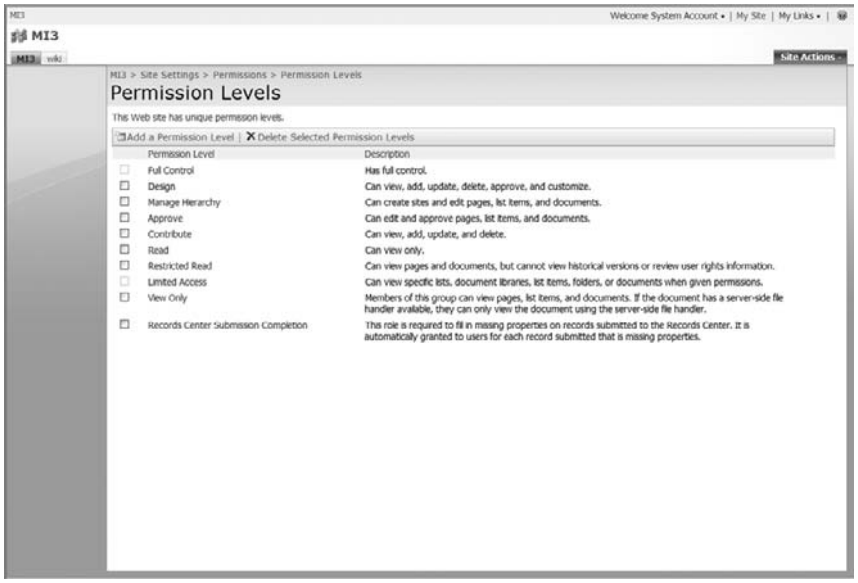


Figure 4.4 *MOSS permission levels on a team site.*

4.2.3 MOSS Features

Features light up sites, and MOSS introduces some excellent functionality that can be used to enhance existing and new sites. If you were to browse to \Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\FEATURES in a default WSS 3.0, installation you would see 31 folders containing features. In a MOSS installation, you will find 137—should give you an idea of the amount of extra functionality you get with MOSS! Features are enabled depending on whether you are running the Standard or Enterprise version of MOSS, but there is a central administrative option to enable Enterprise features should you upgrade from Standard (there is also an option to convert your license type by entering a new product key). New sites created after you install MOSS will have the features associated with the level of installation you have (Standard or Enterprise) but there is also an option to enable the new features on existing sites that you

may have created with WSS 3.0 before upgrading your farm to MOSS. Let's take a look at some of the key features and where they are used.

When we looked at a WSS 3.0 team site, we saw one site feature that was enabled: team collaboration. As you can see in Figure 4.5, a MOSS-created team site running on the Enterprise edition has five site features, and all but one is active by default (you can also see here that the Standard and Enterprise features are indeed separate entities).

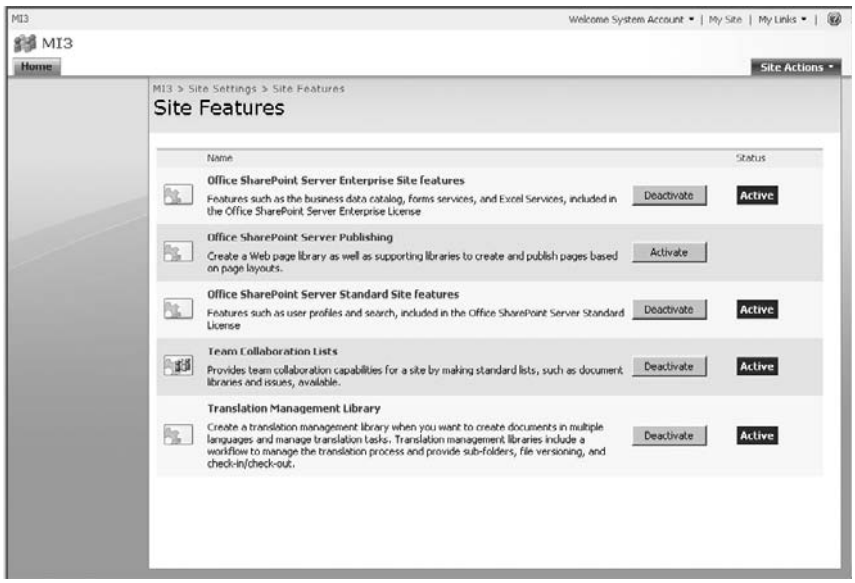


Figure 4.5
MOSS site features in a team site.

These additional features may not make an immediate impact on the naked eye at first glance because, if you compare a WSS 3.0 team site and a MOSS one, the only noticeable difference on the home page is the addition of My Site and My Links options on the link bar at the top right of the page. However, there is much more to this than meets the eye!

Features themselves are scoped to either the farm, Web application, site collection or individual site (also known as a Web), and MOSS adds features at all these levels. Also a feature itself could be a custom action in a drop-down menu, a control on a page, an event handler for a library or list, a button on a toolbar, to a site column, or a Web Part, library, or list. Features can be activated in many ways—either automatically at site provisioning time or manually by suitably authorized users via the Site Actions/Site Settings options. Indeed, an interesting aspect of features is the ability to have a feature activated based on the site definition used to provision the site. For example, the Translation Management Library we see in Figure 4.5 is available globally, but it is only automatically activated for a subset of site templates, namely, the Team

Site, Blank Site, Document Center, and a hidden template called SharePoint Portal Server Site. This is done through something that is known as "feature stapling," in which a farm wide feature basically staples features to sites during provisioning. This is how a MOSS installation can transform existing sites without actually changing the underlying site definition. We cover this in much more detail in Chapter 16, Features, but, if you are interested, you can get a flavor of what I am talking about by looking at C:\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\FEATURES\TransMgmtFunc\TransMgmt.XML.

Back to MOSS features. The MOSS features that you see in Figure 4.5 are Web-level features and equate to the PremiumWeb, PublishingWeb, BaseWeb, and TransMgmtLib feature folders on your Web Front End servers (incidentally, the list of site features is displayed by alphabetical Title). To give you a better flavor of the types of features that MOSS ultimately brings refer to Table 4.1 which highlights ten additional features that are made available and the feature folder that implements them.

Table 4.1 *Ten MOSS Features*

Feature	Type	Description	Folder
Site directory settings	Site collection settings option	Allows registration of team site in local site directory	LocalSiteDirectorySettingsLink
Related links scope settings	Site settings option	Allows grouping of sites for search purposes	RelatedLinks ScopeSettingsLink
Records Management	Event Handler	Refiles documents based on file plans	SearchAndProcess
Report library	Library	Allows creation and management of reports that communicate metrics, goals, and business intelligence information	ReportListTemplate
KPI list	List	Track and display a set of goals	BizAppsListTemplates

Table 4.1 *Ten MOSS Features (continued)*

Feature	Type	Description	Folder
Dashboard page	List Content Type	Contains multiple Web Parts that track KPIs and Excel workbooks. Enabled in a Report Library	BizAppscTypes
Goal Sheet	Site Column	Field that contains reference to underlying Excel sheet for Business Intelligence purposes	BizAppsFields
My Site	Menu option	Links you to your own personal site	MySite
People Search Box	Web Part	Allows you to do a search scoped to people	PortalLayouts
This Week In Pictures	Web Part	Displays contents from an image library in a film strip Web Part	PortalLayouts

4.2.4 MOSS Web Parts

Web Parts are key in delivering appropriate information, and MOSS delivers a multitude of Web Parts that can significantly enhance the usefulness of any SharePoint site. A WSS 3.0 installation provides fourteen default Web Parts for a team site. In MOSS, this extends to more than fifty, with many of them leveraging the new functional areas such as search and business process information. Add to that the Web Parts that are available for other types of sites, such as MySite and publishing, and you can see that there are plenty of options available to you for achieving certain goals. And although these Web Parts are delivered by MOSS, they can be used in straightforward team sites that you create within your MOSS farms. This was not the case with SPS 2003 Web Parts. Of course, if a suitable Web Part cannot be found out of the box, you can always write and install your own custom ones, which we show you how to do in Chapter 13, Web Parts. For now, we just want to highlight three general-purpose Web Parts that will become available to your sites when you deploy MOSS.

Content query

The content query Web Part displays a dynamic view of content. It runs a query that can span lists and sites in your site collection, and then presents the results of that query on the page. You can use this Web Part for many different purposes, such as displaying the most recently published items in a list or library from across all the sites in your site collection. Note that the scope of the search is the site collection, as it is with many of these "roll-up" Web Parts. This means that you cannot use the content query Web Part to rollup information from multiple site collections—a fact that may well influence how you set up your sites in the first place!

RSS viewer

This Web Part allows you to view RSS feeds. Recall that RSS feeds are basically just XML files, so this Web Part is very similar to the standard XML Web Part with a custom XSLT that transforms the XML for display purposes. It's a very useful Web Part because, as we have learned, you can generate RSS feeds from any library or list. Therefore, this capability allows you to display information from a separate site collection. It does not allow you to rollup multiple pieces of information, it can only take one feed at a time but it is a simple way of displaying, say, a list of items from one site collection in another site collection. Note that the RSS viewer connects to a feed anonymously; therefore, you can only consume RSS feeds from a SharePoint site if it is set-up for anonymous access. This is a two-stage operation: you enable anonymous access at the Web application level and then at the site level. You can use SharePoint Central Administration to enable it at the Web application level (under Application Management/Authentication Providers, modify the Default zone). You then use Site Actions/Site Settings/Advanced Permissions and then use the Settings menu to set anonymous access levels. If anonymous access is not applicable to your scenario, you can consider using Kerberos authentication instead by changing the IIS authentication settings for the Default authentication provider from NTLM to Negotiate(Kerberos).

Table of contents

This Web Part allows you to build a site map of your site collection, showing up to three levels from the starting site. You can show details of the sites, lists, libraries, and pages in your site collection and easily control the layout. This is a useful Web Part for showing visitors what content your site collection contains.

4.2.5 MOSS Templates

We have already discussed some of the templates that WSS 3.0 provides such as wiki and blog. MOSS extends the list considerably to support the new functional areas it introduces. On the Template selection user interface, you will find two additional tabs for enterprise and publishing. The enterprise templates deal with large-scale document libraries, records repositories, site listings, personalization, and various forms of search. The publishing templates deal mainly with Web content management.

A couple of publishing templates are only available when creating a root Web site (which is typically done via the SharePoint Central Administration program) and are starter templates for a typical divisional intranet portal (called a collaboration portal) and a typical Internet-facing Web site (called a publishing portal). We will look at the publishing template later, but for now let's take a closer look at the collaboration template as it is the one that is designed to be the replacement for a SPS 2003 portal. We can see the home page of a site created with the collaboration portal template in Figure 4.6.

This is a site collection containing a top-level site and five subsites instantiated from the site template called SPSPORTAL. An interesting point is that the control file that generates the site collection (which is stored in WebManifest\PortalWebManifest.xml) only has template entries for the following sites (template names are shown here, and you can find them in \Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\SiteTemplates):

```
Home Page (called Spies in our example)  SPS
Document Center      BDR
News                 SPSNHOME
Search               SRCHCEN
Sites                SPSSITES
```

So how does the reports subsite get created? It is specifically associated with the SPS template and called out as a site feature within its ONET.XML file. The feature itself is called ReportCenterCreation, and it is actually an event handler feature that is ultimately responsible for creating the subsite. The subsite uses the SPSREPORTCENTER template.

All but one of the sites in this collection are actually built on a base publishing site that has the publishing feature enabled. What does this mean to you? Well, in any normal site, each Web Part page has two views on it: shared and personal. Suitably privileged users can modify each view of a page by adding, moving, and deleting Web Parts. By default, you will see the shared view of a page unless you have customized it and therefore have a personal

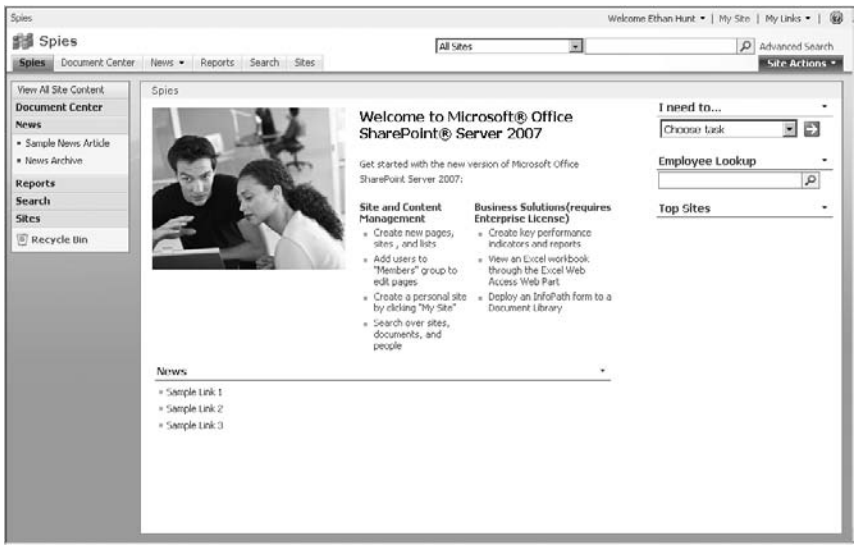


Figure 4.6 *A Moss Collaboration Portal*

view. You can switch between shared and personal view using the menu options that appear when you click the arrow next to your name in the navigation bar at the top right of a page (this navigation aid is sometimes called the Welcome bar).

When you modify a page in either view, your changes are made live as soon as you save the page. As a result, there is essentially no control over publishing updated pages. If you have the rights to modify a shared view then, as soon as you save that change everyone will see it. While this is fine for a small team site that allows only a few people to make changes, it is not ideal for a larger repository, such as our divisional portal, where many people may want to make changes to their piece of the pie, but where we also want to maintain standards, to undo changes easily, and to track and audit changes.

It is no accident, then, that the template for our collaboration portal is categorized under publishing, as it has full support for controlling the editing and publishing of pages. Indeed, as explained in Chapter 2, Windows SharePoint Services V3.0, the core platform services provide the ability to do versioning, check-out/in, etc—and a publishing-based site takes full advantage of this. Some very simple clues indicate whether a site has the publishing feature enabled, as we can see in Figures 4.7 and 4.8. These are the results of taking the Site Actions/Edit Page option in a normal site and a publishing-based site. Notice the extensive set of options and the editing toolbar that appears for the publishing-based site! You can see immediately why the reports site is created to support the publishing process, since it is used to show us the status of our site in terms of which pages are checked out,

require approval, etc. We will discuss publishing sites in more detail in Content Management.

You should now be equipped with enough information to figure out which site in the collection is not a publishing based-site! (Oh, okay—the answer is the Document Center.)

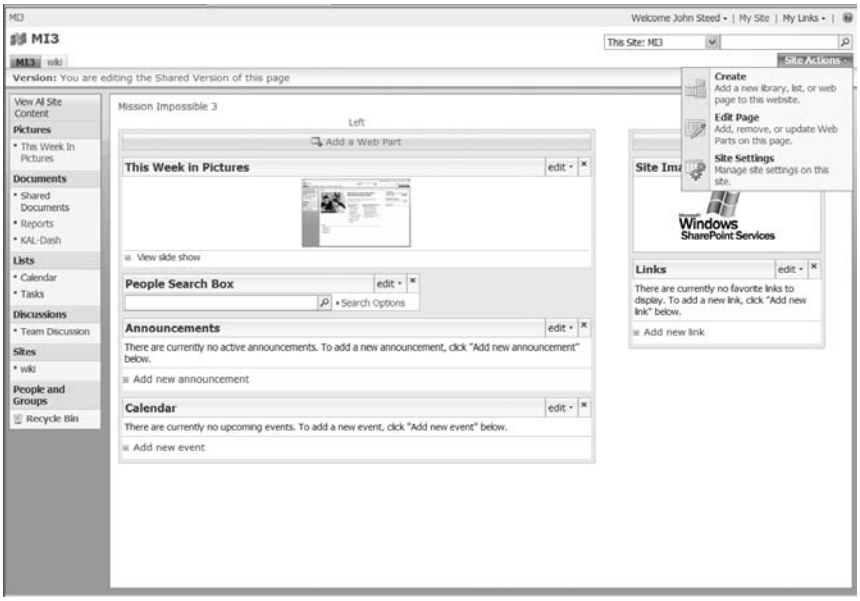


Figure 4.7 *Editing a standard page.*

Let's now take a brief look at the individual sites in our default collaboration portal.

Home Page

The home page of our collaboration portal, built using the SPS template, includes an image and a directly editable Welcome area on the page for use by those who are suitably authorized—this is referred to as page content. In terms of Web Parts, it has the following:

- **I need to…:** This is an instantiation of the task and tool Web Part (TaskAndTools.Webpart in the Web Part gallery). The intention is to give a list of helpful tasks that users may need to do. Behind the scenes, the data from this Web Part actually comes from entries in the Sites list in the Sites subsite. These entries contain URL links to a page that explains the task at hand. The entries are tagged as being Top Tasks, and this Web Part merely uses that as a filter.

Figure 4.8 *Editing a publishing page*

- **Employee LookUp:** This is an instantiation of the standard search Web Part (SearchBox.dwp). It has been configured to scope the search to people, and the query results are passed over the Search subsite, where a search results handling page called peopleresults.aspx handles the display.
- **Top Sites:** This is an instantiation of the content by query Web Part (ContentQuery.Webpart). Its properties target the Sites list in the Site subsite and filter the query results based on sites tagged as Top Sites.
- **News:** This is an instantiation of the summary links Web Part (SummaryLink.dwp). It provides a place for managing a set of links to other pertinent information.

Document center

The document center site (created from the BDR template) is designed as the central storage point for managing enterprise documents. You can create and organize as many document libraries in this site to suit your needs.

News

This site is used to publish news pages. Authors can create pages using predefined page layouts in order that all news articles conform to particular standards. Features such as attaching an image to a news article that is then used in content roll-up Web Parts (such as the content by query Web Part) are

supported. Once articles have gone through an approval process and published, they appear in the drop-down menu on the global horizontal navigation bar.

Reports

The report center site is used to monitor and report on key activities. It integrates with the business intelligence services delivered by MOSS and has a report library within it for storing documents and dashboards.

Search

The place where all search results are rendered!

Sites

This contains the site directory, which is essentially a list called Sites that is populated as sites are created or manually updated with links to already created sites (either SharePoint or non-SharePoint sites). By carefully controlling the metadata associated with the Site Directory, you can turn it into a very useful Yellow Pages-type directory so that everyone in your organization can benefit.

To finish this discussion on the collaboration portal template, we'd like to touch on managing site content, structure, and navigation. There is an option that is only available for sites that have the publishing feature enabled that allows administrators to manage a site collection's content and structure from one place, as shown in Figure 4.9. If you are used to managing the hierarchy of portal areas in SPS2003, then this is the equivalent place in a MOSS collaboration portal. From this bird's-eye view of your site, you can easily move content around and create new content, such as sites, lists, libraries, and items. You can even perform operations like viewing the recycle bin and site settings of individual sites and lists from this window.

4.3 Business Data Catalog

The Business Data Catalog Shared Service (BDC) is a metadata-driven connector that allows you to access data held in back-end line-of-business applications and surface that data in multiple places within your site collections. This means that back-end data can be used in site columns, look-up lists, user profiles, and Web Parts. Indeed, the data can even be surfaced through other shared services, such as search, which can significantly increase the value and reach of your business data. Although the surfaced data is read-only in SharePoint you could, for example, surface links to the actual back-end application and present these to the end users as custom actions. That

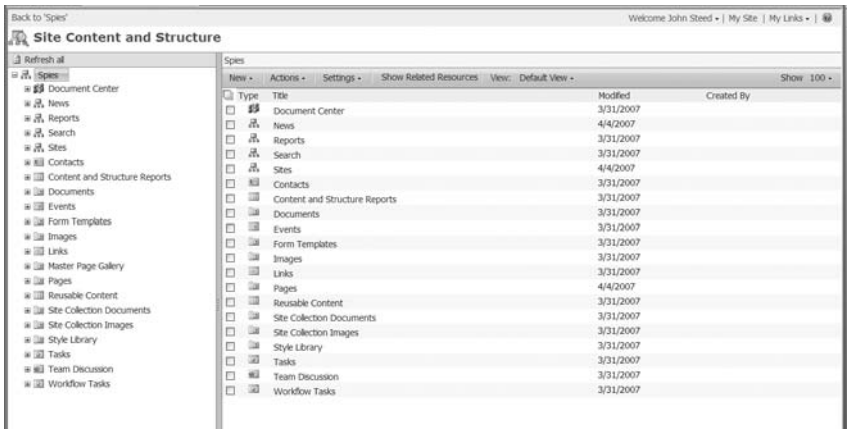


Figure 4.9
Manage site content and structure.

way the end user can easily launch the actual back-end application with the relevant item selected for processing.

You might think that a discussion about accessing data in back-end line-of-business applications would be a developer-type topic and, to some degree it is, since an awareness of the objects is certainly required. But the reason we are covering it in this section of the book is that you do not need to write any code to exploit the data. The intention here is to give you a high-level overview of how it works and to show it in action through some samples so that you can appreciate its power.

The BDC removes the complexity of accessing back-end data which, to date, has typically meant writing lots of glue code that cater for security requirements, open connections, execute queries, filter data, etc. Instead of expending your efforts writing this code, you now need to think about describing the objects and access methods to the back-end data in XML, and the BDC framework takes care of all the rest. The BDC supports two types of access methods—to practically all popular databases using ADO.NET, OLEDB, and ODBC drivers, and through application-specific Web services, assuming it supports a relevant retrieval method. A logical diagram of the BDC is shown in Figure 4.10.

4.3.1 Application Definition File

An application definition file, is used to define the access methods and objects in the back-end data source. In essence, it describes the APIs for the application, what can be done with the API, and the relationships between the different data entities found in the back-end database or application. The ADF is an XML file that must be well-formed and adhere to the schema defined at C:\Program Files\Microsoft Office Servers\12.0\Bin\bdcmetadata.xsd on a WFE server. The major components of the ADF are as follows:

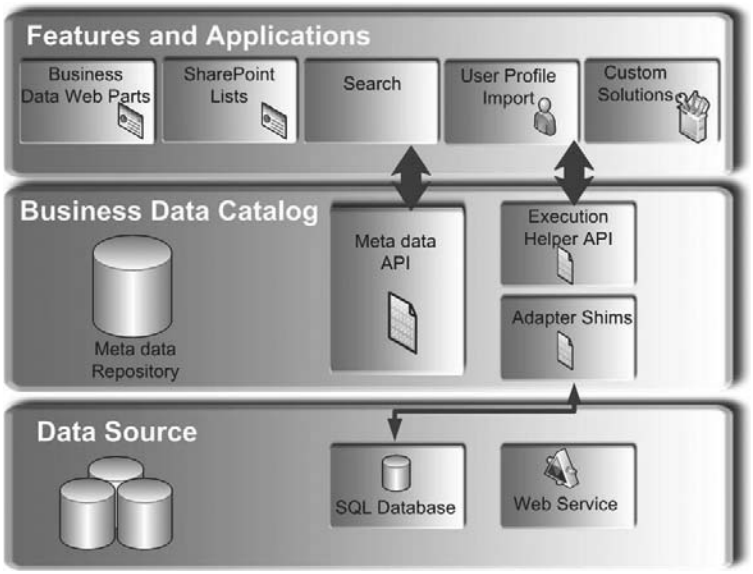


Figure 4.10
The BDC

- **LOBSystemInstance:** Defines authentication and connection information.
- **Entity:** Defines an actual item that is specific to the application—for example, a product or customer. Each entity contains definitions for identifiers, methods, filters, and actions. Think of the identifier as the primary key within a database table—it is used to uniquely identify a particular instance of an entity.
- **Methods:** Defines the methods by which we can locate instances of an entity—much like an interface definition. So, for example, this could be a SQL statement, a stored procedure for a database, or a Web method for a Web Service. Certain methods have to be defined for certain SharePoint functionality to function. For example, an IDEnumerator method has to be defined if the data is to be indexed by the search engine. This method is used to crawl through all instances of the entities.
- **Filters:** Used to control the number of entities returned by a method.
- **Actions:** These define links back to the data and are surfaced in SharePoint anywhere an entity is surfaced. Thus, they can be used to quickly access the source data through the source application for full access to the data or for linking to anywhere based on the returned data. For example, you may want to link to a Google or Windows live search, passing a piece of returned data into the search.

- **Associations:** These are used to link entities together. A typical example would be two tables in a database with a one-to-many relationship—for example, sales regions and sales personnel operating in the regions.

Defining the contents of the ADF demands knowledge of the implementation of the back-end application to define the entities, methods and associations. Also defining the correct XML is never a trivial task, but you can find samples that use the Adventure Works SQL sample database in the Office SharePoint Server 2007 SDK. There are also some community tools and third-party applications, such as http://www.bdcmetaman.com/default.aspx, starting to appear.

As a sample, consider the two simple SQL tables shown in Figure 4.11. One is a list of actors and the other is a list of movies. We can link the two tables together using the ActorId column in each table.

Figure 4.11
Actor and Movie tables.

Table - dbo.Actors | Table - dbo.Movies | Summary

	ActorId	ActorName	Nationality	WebLink	Age
	1	Sean Connery	Scottish	http://bonds/007	38
	2	Daniel Craig	English	http://moviestars/craig	50
✎	3	Pierce Brosnan	Irish	http://pierce/blog	53
*	NULL	NULL	NULL	NULL	NULL

Table - dbo.Actors | Table - dbo.Movies | Summary

	MovieId	MovieName	ActorId	Year
	1	Casino Royale	2	2006
	2	Dr No	1	1962
	3	From Russia Wit...	1	1963
	4	Goldfinger	1	1964
	5	Thunderball	1	1965
	6	Tomorrow Never...	3	1999
	7	Die Another Day	3	2002
✎	8	Layer Cake	2	2004
*	NULL	NULL	NULL	NULL

We can therefore define an Actor and Movie entity and the relationships between them in our ADF. You can see a sample of the ADF file in Figure 4.12, so let's just discuss a couple of the key XML nodes within the ADF file so you can relate them to the SQL tables in our sample.

You can see that we have an *entity* defined called Actor; this relates to the entries in our Actors table, and within that entity we have an *Identifier* that defines ActorId as the primary key for looking up entries. We then have two *methods* definitions. The first is called GetActors and the second is called

GetMoviesForActor. Since the data is coming from a database, we see typical SQL commands that will be executed by the BDC. If this had been a Web services definition, we would see Web service methods here instead. We will see these methods being called when we look at a couple of Web Parts that harness the BDC.

The next item we'd like to point out from the figure is the *Actions* node. This is a way to define custom actions on a selected piece of data. In the sample, when a custom action is taken, it will launch a new window to the Internet Movie Database (www.imdb.com) with the current Actor Name as a parameter. As you can see, this is a very powerful way to quickly link to relevant information, no matter where it might be located!

The last item for discussion is a node that you cannot actually see in the figure. It's the *Associations* node and it defines the relationship between the the Actor and Movie tables. It defines the GetMoviesForActor as the method to call when any linkage is required between the two tables. We will see this in action when we discuss the BDC Web Parts.

Figure 4.12 *Application definition file.*

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <LobSystem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.microsoft.com/office/2006/03/BusinessDataCatalog BDCMetadata.xsd" Type="Database" Version="1.0.0.1"
  Name="Spies" xmlns="http://schemas.microsoft.com/office/2006/03/BusinessDataCatalog">
 + <Properties>
 + <LobSystemInstances>
 - <Entities>
  - <Entity EstimatedInstanceCount="100" Name="Actor">
   - <Properties>
      <Property Name="Title" Type="System.String">ActorName</Property>
      <Property Name="DefaultAction" Type="System.String">View Profile</Property>
     </Properties>
   - <Identifiers>
      <Identifier Name="ActorId" TypeName="System.Int32" />
     </Identifiers>
   - <Methods>
    - <Method Name="GetActors">
     - <Properties>
        <Property Name="RdbCommandText" Type="System.String">SELECT * FROM dbo.Actors WHERE (ActorId >= @MinActorID) AND
          (ActorId <= @MaxActorId) And (ActorName LIKE @ActorName)</Property>
        <Property Name="RdbCommandType" Type="System.Data.CommandType">Text</Property>
       </Properties>
     + <FilterDescriptors>
     + <Parameters>
     + <MethodInstances>
     </Method>
    - <Method Name="GetMoviesForActor">
     - <Properties>
        <Property Name="RdbCommandText" Type="System.String">Select MovieId, MovieName, Year From Movies Where
          ActorId=@ActorId</Property>
        <Property Name="RdbCommandType" Type="System.String">Text</Property>
       </Properties>
     + <Parameters>
     </Method>
    </Methods>
  - <Actions>
    - <Action Name="IMDB Search" Position="1" IsOpenedInNewWindow="true" Url="http://imdb.com/find?q={0}" ImageUrl="">
     - <ActionParameters>
        <ActionParameter Name="ActorName" Index="0" />
       </ActionParameters>
```

4.3.2 Importing an Application Definition File

Once you have your ADF defined, you have to import it into the Shared Services Provider—remember, the BDC is a shared service. You do this from the SSP Administration Web application, which is accessible through Central Administration. The import basically analyzes the XML, verifies that it is conformant, and loads all the metadata definitions into the shared services database. Once this is complete, the entities within the shared services database are now available for consumption by other parts of SharePoint, such as

Web Parts, Lists, User Profiles, and Search. You can see the Actor entity as it appears in the SSP Admin after our ADF file has been successfully imported in Figure 4.13.

The fields you see in Figure 4.13 are those we defined in our ADF—and in our case they correspond to all the fields in our source table—but it's up to the designer of the ADF to decide how much data to surface from the underlying source. You can also see the relationship we defined and three actions. The first two we defined in our ADF, and the third, View Profile, is a default one that happens automatically with the BDC. The View Profile action ultimately calls an aspx page that shows all the data for the selected record. The page can be customized further with Web Parts, so it really is very powerful.

Note that you can also add more actions through the SSP Admin page. You do not have to actually define all your actions in the ADF.

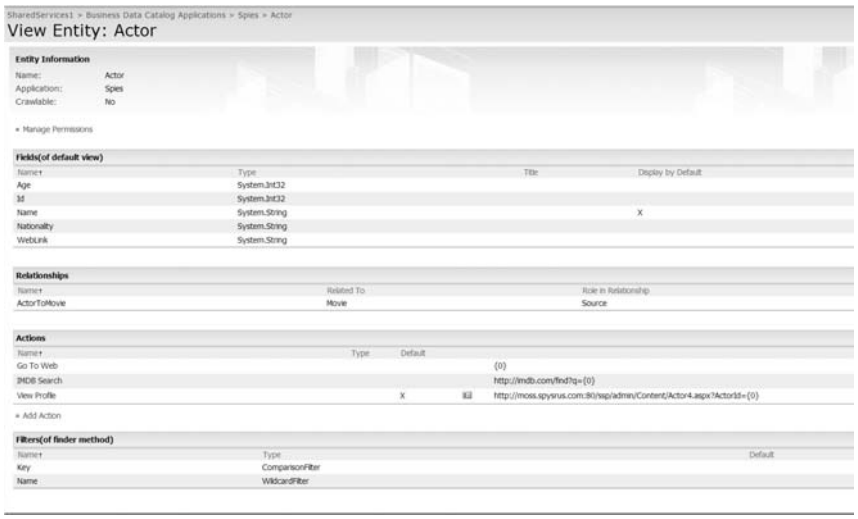


Figure 4.13 *Actor entity in the BDC.*

We now have our data source defined in the BDC. Let's now see how this catalog of definitions can be consumed by others.

4.3.3 BDC Web Parts

Out of the box you get six BDC Web Parts, which can be used to display entities from the BDC without lifting a coding finger. Through Web Part connections you can join them together to build very sophisticated Web pages. So let's see how we can use the business data list and business data related list Web Parts together.

The BDC list Web Part allows you to chose an entity from the catalog and display all the instances of that entity in the back-end data source. It will also allow you to filter the query on the back-end data based on the FilterDescriptions that were in your ADF file. You can also control the output via your own custom XSL stylesheet. Figure 4.14 shows the Actor entity being displayed on a team site—yes, a team site! Remember that any MOSS Web Part can work in a WSS site. The Web Part itself was placed on the blank page and then configured via the tool pane. The tool pane allows you to browse the BDC to pick the entity you are interested in. What you are seeing here is the Web Part out of the box, and, as I hope you have spotted, the Actions associated with the entity are available to us when we hover over them. Thus, if we were to take the IMDB search option, as shown, a separate window would be launched with the URL of http://imdb.com/find?q=Sean+Connery. The Web Part can be configured further to suit your needs, so let's see it working in conjunction with another Web Part.

Figure 4.14
BDC list Web Part.

We can now use the BDC related links Web Part to exploit the relationship we set up between the Actors and Movies database in our ADF. When we configure the related links Web Part, we are offered all the relationships defined in the BDC. Therefore, by choosing our ActorToMovie relationship and then using Web Part connections to feed the selected Actor into it, we get the result shown in Figure 4.15 (Sean isn't really that young, is he?).

I have configured the Actor Web Part to show all the items and removed some of the columns to make it more pleasing to the eye. But I think you will agree that this result is very impressive, given that not one line of code had to be written to achieve it!

4.3.4 BDC and Lists

Once an entity is defined in the catalog, it can be accessed in many places. We will look at user profiles and search in subsequent chapters, so let's finish by taking a quick look at how lists can benefit from the catalog.

Lists are defined in terms of columns. And you can define many column types; is one such is a business data type. When you do so, you are presented with a user interface that allows you to choose the entity from the catalog

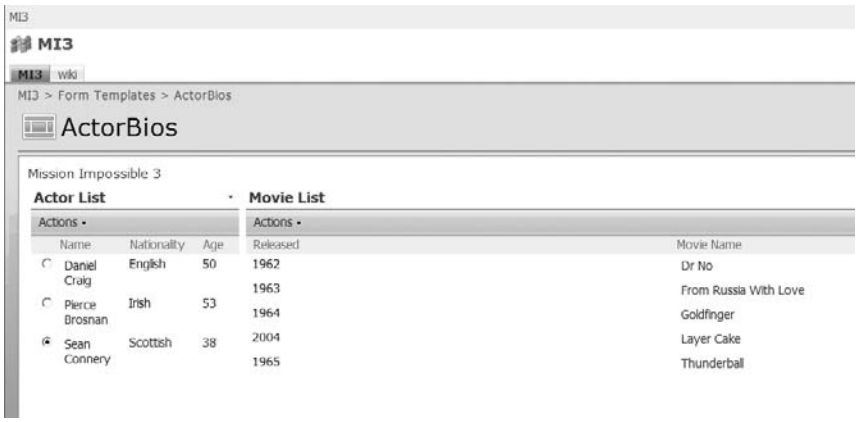


Figure 4.15
Linking Actors and Movies.

that you want to see in the column. And not just a single property—you can choose as many properties as you like from the entity, and they will be populated into the list. The chosen properties are entered into the list as read-only columns, but they will be dynamically updated when any of the writeable columns in the list item are updated. Actions are also supported. Our last figure in this chapter, Figure 4.16, shows our new page with another Web Part *on* it. This Web Part is a standard list part, and the list itself combines end-user–entered data with data that has been read from the back-end database.

Figure 4.16
BDC and lists.

Cool, huh? Do you have any data locked up in back-end business applications that you'd like to exploit? The BDC may well be the answer to this!

5

Enterprise Content Management

As pointed out in Chapter 1, 2007 Microsoft Office System, one of the solution areas targeted by SharePoint is Enterprise Content Management (ECM). Although ECM is a fairly commonly used term, it's generally quite difficult to come up with an elevator pitch that adequately covers everyone's particular view of ECM. AIIM, the ECM Association (www.aiim.org), describes ECM as follows:

> "*Enterprise Content Management (ECM) is the technologies used to capture, manage, store, preserve, and deliver content and documents related to organizational processes. ECM tools and strategies allow the management of an organization's unstructured information, wherever that information exists."*

A similar definition from Wikipedia states that ECM is:

> *"software technology that enables organizations to create/capture, manage/secure, store/retain/destroy, publish/distribute, search, personalize, and present/view/print digital content such as pictures/images, text, reports, video, audio, transactional data, catalog, code. ECM systems primarily focus on the capture, storage, retrieval, and dissemination of digital files for enterprise use and their life-cycle management."*

From our point of view, the important concept to take from those definitions is that ECM is not a single application but a set of tools and technologies that allows you to manage content to achieve your business goals. These tools and technologies should be capable of managing the complete lifecycle of content, from its entry into the digital world to its exit, as well as maximizing the value of your content to your business.

In SharePoint Server 2007 ECM tools and technologies primarily surface in four functional areas: document management, records management, Web content management and electronic forms management. And all these functional areas leverage common services, such as storage, workflow, security and search. Let's now look at some of these functional areas and highlight the ECM features that can help you manage your enterprise content appropriately.

5.1 Document Management

Document management refers to all the things you can do to the content you store in lists and libraries. In Chapter 2, Windows SharePoint Services V3.0, we covered many of the key technologies that allow you to manage any type of content that you choose to store in a site. *Content types,site columns, versioning, check-in/out, approval, security trimming* are all involved in managing content, as is integration with *workflow.* All of these are available in the base WSS 3.0, platform but MOSS extends the technologies to help you better monitor your content, to help users better understand how content should be used, to ensure your content meets specific requirements, and to protect content when it is taken out of SharePoint. It does this via auditing, information management policies and rights management integration. At a high level, all of these technologies help you meet compliance requirements, so let's take a general look at that first.

5.1.1 Compliance

Compliance—be it legal, regulatory or corporate—is a major concern for most businesses these days. You can read many horror stories of businesses being fined billions of dollars for not complying with a particular regulation such as Sarbanes-Oxley or HIPAA (the Health Insurance Portability and Accountability Act). Regardless of the actual regulation, the root cause is usually a failure to manage digital content appropriately.

Being compliant needs to be a core business tenet, not just a separate project that you embark upon and then deem complete. It needs to be engrained in your common business processes, something that everyone considers as they go about their day-to-day jobs. For example, users should be aware of your policies with regards to retaining or deleting e-mail and documents or know how to properly classify and secure information so that it is not subsequently used inappropriately. As a business you need to first decide what your policies are (some will be driven by regulations) and then help your employees understand and comply with these policies. Ultimately, these policies help ensure that information is handled appropriately, stored appropriately and with the right metadata to allow its subsequent discovery and retrieval.

SharePoint Server 2007 provides a collection of features that address common scenarios that can be used to help you support your specific compliance requirements. Whether these features are sufficient to meet your needs will depend upon individual circumstances, but be aware that SharePoint is a very flexible and extensible platform. Therefore, while it may not meet 100% of your compliance needs, it will more than likely go a long way toward achieving them. Understanding the available features is key in helping you map to whatever regulations are appropriate for your business. As an example of this, an excellent white paper on SharePoint's role in supporting regulatory compliance with the Sarbanes-Oxley Act of 2002 is available from Mindsharp at www.mindsharp.com. This paper looks at the sections of Sarbanes-Oxley that are most applicable to collaboration and lists the features of MOSS that can be used to address each section.

Also available is a white paper from Microsoft that talks about compliance across the whole of the 2007 Microsoft Office System, at www.microsoft.com/downloads/details.aspx?FamilyID=d64dfb49-aa29-4a4b-8f5a-32c922e850ca&displaylang=en.

5.1.2 Auditing

Defining granular permissions over content in lists and libraries is possible with WSS 3.0. Versioning can even be used to see what changes were made to an item and by whom. But these features don't easily tell you who has been accessing the information and what they have been doing to it over any extended period of time. And this information is something that many people require—especially when regulations dictate that they should track which users gain access to important content, what they do to it, and when.

WSS 3.0 provides the basic audit log, but other than being able to switch it on at a site collection level, there is not much more you can do with the audited information unless you write some code to do so (incidentally, there is an excellent white paper and sample code written by Ted Pattison, available at http://msdn2.microsoft.com/en-us/library/bb397403.aspx that shows exactly how to do this!). MOSS, on the other hand, provides an administrative user interface for specifying which events are audited and can generate reports from the audit logs. Furthermore, MOSS provides more granular control over what is actually audited. MOSS does not let end users view the audit for individual items, but the abovementioned sample code can also be used for this purpose.

As you can see in Figure 5.1, there are eight events that you can choose to audit. Note that you can even audit viewing of items and searches on the site's content. Also, some system auditing events will surface (such as workflow events) and custom events can be logged from within your own applications. You can set auditing at different levels: site collection, content type,

and library/list. If auditing has been set at multiple levels (for example at the site collection level and on an individual document library), then the audit events are amalgamated so you cannot, for example, enable the auditing of editing all items except those in a particular list. You can, however, enable the auditing of editing of items for only certain lists/libraries by not setting this at the site collection level and setting it at the individual list/library level instead. Additionally, note that auditing of site-wide events (the last three shown in Figure 5.1) is only available at the site collection level.

Setting audit events on content types and lists/libraries is done by applying information management policies, which we cover shortly. If you have set any auditing events on a content type at the site collection level, then they will apply everywhere within the site collection that the content type is used. You cannot set a different set of auditing events for that content type within an individual list or library.

Figure 5.1
Configuring auditing at the site collection level.

Once you have enabled the auditing of events, the system keeps track of what is going on and logs details into the AuditData table in the database. It keeps note of where the events are occurring, when they are occurring, who is doing the activity, and what they are doing. Specific data associated with particular events, such as keywords passed in a query and the search scope, are also stored.

Audit log reports are generated at the site collection level by an administrator. As you can see in Figure 5.2, four types of reports can be generated that gather together events of similar type. Therefore, there is a report that will show you all the events that modified content, or one that shows you all content that was viewed. These reports filter the gathered audit data, and you can run a custom report if you want to define your own filter. The content of the report is based on the XML language known as Excel Markup Language (ExcelML) and can be opened with Excel 2007. We will see in Chapters 8, Business Intelligence and Business Process, how you could perhaps render the display via Excel Services. When you open the report with Excel, you will see a summary sheet plus a sheet containing the raw data so that you can splice and dice it to suit your needs.

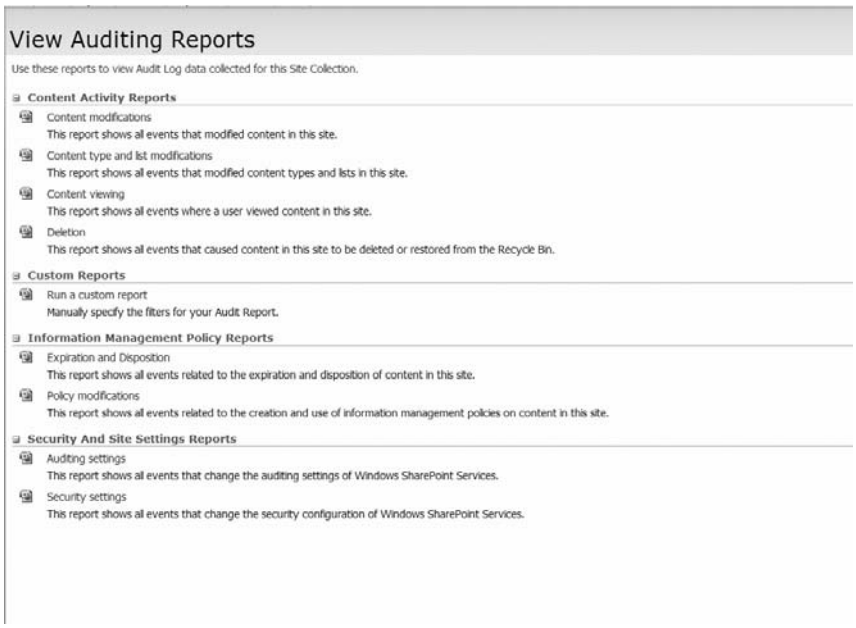


Figure 5.2
Audit reports.

At time of writing, the only way to clear the audit log would appear to be programmatically, using the DeleteEntries method of the SPAudit object.

5.1.3 Information Management Policies

Information management policies can be used to automate management of content and can therefore help your system be compliant. They are introduced into MOSS as part of the whole records management area, but they can be applied globally, not just to items that are in the records repository. Each policy defines a set of rules, and each rule enforces relevant processing for the task at hand. For example, you may have an expiration rule that defines how long content of a particular type should be kept within particular document libraries or lists.

Before a policy can be enforced, it has to be defined, and there are two ways in which this can be done. First, you can define policies at the site collection level that are then made globally available to any library or list within the site. Second, suitably authorized users can define their own policies and apply them directly to a library or list. The benefit of applying policies from the site collection gallery rather than custom policies is that any change made to the site collection policy will automatically be propagated to all the lists and libraries that are using that policy within the site. Whichever method you use to create a policy, you are presented with a page as shown in Figure 5.3. This figure demonstrates that, out of the box, there are up to four rules

you can enable within an individual policy. Now I say "out of the box" because the list of policy rules (AKA policy features) is extensible through custom code, and also, you can control which policy rules are available at the farm level using central administration operations. For example, there is another rule called Form Conversion for Archiving that can convert documents into a final form for read-only purposes. This rule will appear once document conversion is enabled on the farm. Disabling a rule at the farm level basically decommissions it, but any policy that has already been applied with this rule will continue to be active.

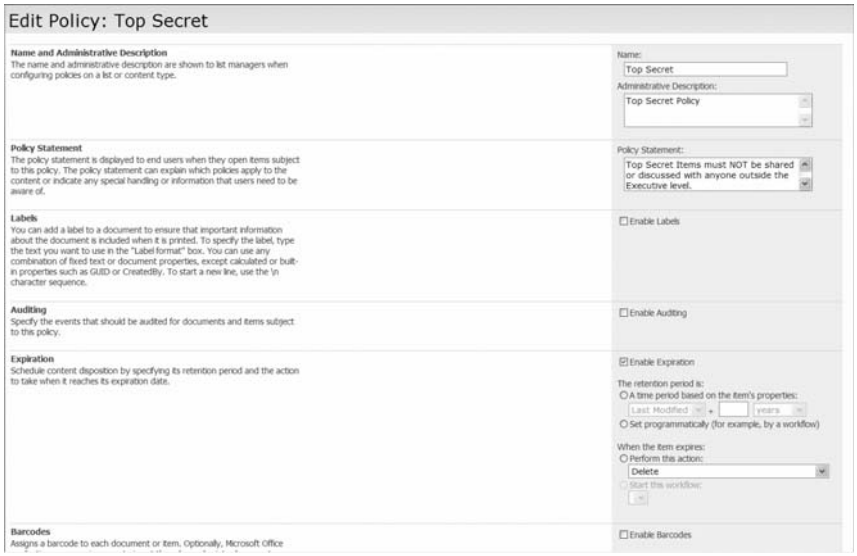


Figure 5.3
Creating a policy.

Remember that the goal of any compliance strategy should be to help your users understand your policies so they can more easily adhere to them. No one likes doing something that they don't understand! This is the primary purpose of the policy statement; you can think of it as handling instructions for the end user when dealing with content that is subject to the policy. Indeed, the type of content you are dealing with influences whether and where the policy statement is actually shown. For example, if you open an Office 2007 document that is subject to a policy, then the policy statement for that policy is shown in the information panel. However, if you open, say, a Notepad document, then the policy statement is not displayed anywhere. Similarly, if the policy applies to a list, then the statement does not surface anywhere.

Whether the out-of-the-box rules can actually function as fully intended depends on the type of content being considered. For example, the Labels and Barcodes rules create some properties on the item in the library or list.

The intention of these is to be able to track both the physical and electronic copy of the document; therefore, Word 2007 can automatically insert the barcode and/or label into the document when saved or printed. But this is only possible with Office 2007 documents. You can see the effect of setting a label and barcode in Figure 5.4. Note how the label can be constructed from other properties on the item. Also, note that some of the items do not have a barcode or label—why is that? These policy rules only *create* the label and barcode properties when the item is initially created, and editing an existing item is not sufficient to have the properties created.

The other policy rules—Auditing and Expiration—are content agnostic. Each rule also needs to be individually configured. The Auditing rule allows you to set those events that you want to audit, as we have previously discussed. Configuring an Expiration rule can be seen in Figure 5.3, which indicates how to set the expiration date—either a time period added to a property on the item, or an explicit date set programmatically. You can also indicate what happens to the item when the expiration date is reached either deleting the item or launching some form of workflow. The actions are extensible, and the out-of-the-box ones allow you to either automatically dispose of the item or have some form of human intervention through a workflow. Also, when an expiration policy is in effect, you can display the expiration date as a column and, by viewing an item's properties, indicate that an item is exempt from the policy.

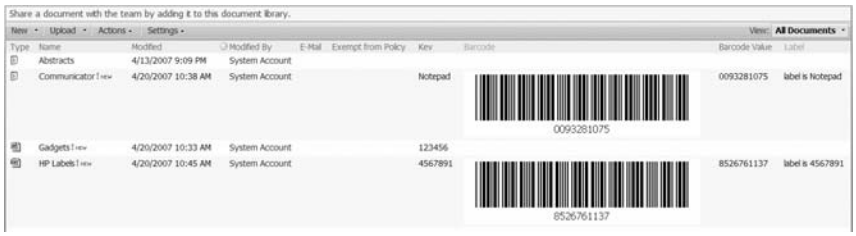


Figure 5.4
Labels and Barcodes

Once you have defined your policies, you have to apply them. Policies are ultimately associated with a content type, as you will probably want different management for different types of information. When you apply a policy to a library or list that does not accept multiple content types the policy is subjected to the default content type. If the list or library does support multiple content types, you need to choose the type to which the policy applies. You can also actually associate a policy directly with a site content type, so that wherever it is applied, its policy will follow it. Thus, in the end, it is really the individual items stored in lists and libraries that have a policy applied to them according to their content type.

Some policy rules, such as Labels and Barcodes, run synchronously as items are modified in the library. Others, such as Expiration, run on a sched-

ule via the SharePoint Timer Service. The expiration policy rule runs daily, and you can check its status via the Operations tab in Central Administration.

5.1.4 Rights Management

Information Rights Management (IRM) was introduced in the Office 2003 time frame and enables you to set rights on items, specifying who is allowed to do what with the item. IRM allows you greater control over digital information and can be used to protect users from inadvertently doing something inappropriate with content. For example, it is very easy to control who has access to repositories through access control lists, but once a user has read access to a document they can take copies of it, forward it outside the organization, print it, and so on. The user might be unaware that the content is not meant for further distribution; therefore, protecting it with IRM means that only intended people and intended operations can be performed on the content. IRM helps maintain the confidentiality and integrity of information, which is central to many regulations.

Windows Rights Management Services (RMS) allows you to set policies that enable better control over who is allowed to do what with documents. RMS works by encrypting the file and limiting the set of users and applications that are allowed to decrypt the file. It can also limit the rights of users who are allowed to read the file, so that they cannot print copies or copy text.

While RMS certainly works, a common issue with it is that *applying* the RMS policy to the information (be it a document, e-mail, etc.) is at the discretion of the end user. There is no easy way to ensure a user applies a policy, let alone the right policy.

SharePoint 2007 can integrate with RMS to provide a solution for this issue by essentially linking an IRM policy to items in the library (this also applies to attachments in list items). Note I said "linking," since the IRM policy is not physically on the documents when they are in the library. If this was this case the source documents would be encrypted and all sorts of problems would break out regarding indexing, search and so on. What actually happens is that when documents are exported from the library an appropriate IRM policy is applied. Also, when an IRM-protected document is uploaded, the policy is removed, leaving the document unencrypted. The IRM policy will be reapplied when the document is next exported. This seamless application ensures that users do not have to worry about whether the document is secured correctly once it has been taken out of the security system bounded by SharePoint. SharePoint security applies when the document is in the library, and an appropriate IRM policy secures it when it is outside SharePoint.

So what policy gets applied to a document that is in an IRM-secured library when it is exported? That depends on the users's rights to the source

document library. A mapping is done between SharePoint permissions and IRM rights, as shown in Table 5.1. Also, if the document was IRM-protected before it was imported into SharePoint, any IRM rights that it had initially are also reapplied.

Table 5.1 *SharePoint Rights Mapped to RMS Rights.*

SharePoint Rights	IRM Permissions
Manage Permissions Manage Web	Full control of the documents, as defined by the client application. This generally permits the user to read, edit, copy, save, and modify permissions of the document.
Edit List Items Manage List Add and Customize Pages	Edit, copy, and save permissions. The user can only print the document if the document library IRM settings have been configured to allow document printing.
View List Item	Read permissions. The user can read the document, but not copy or edit its content. The user can only print the document if the document library IRM settings have been configured to allow document printing.
All other ACL rights settings, such as Edit User Info	Not applicable; no corresponding IRM permissions.

SharePoint is the primary owner of the protected content, and the user who performs the download is added as a consumer of the document. This means that only SharePoint and the user have any rights to the file. Even a user who also has access to the document library where the document originated will not be able to access this file. Rather, they would have to go and fetch the document from the library themselves to get their own, properly protected, document. This is as it should be, since you want to ensure that each user only gets the rights to do what they are allowed to do in the source document library.

Configuring IRM involves registering the SharePoint farm with the RMS certification service and then subsequently enabling IRM on individual libraries and lists.

5.2 Records Management

We've explained some of the core features that aid compliance and can be used in collaborative team sites. But this is not enough when it comes to the wider needs of the organization. For example, perhaps the corporation has a

policy that states that all information related to a particular subject matter—say, financial reports—must be kept centrally for a particular period of time regardless of where that information originates. There are many reasons why companies want to do this type of thing, particularly if they feel that there is the possibility of an audit by a regulation authority or some similar thing. Therefore, records management should be a key component of any organization's compliance strategy.

With this in mind, MOSS offers a centralized records repository that leverages all the information management policies we have just described and makes it simple for end users to deposit items into the repository for filing according to a corporate file plan. Organizations are aware that they must retain certain information but, they cannot possibly keep everything. The goal of a records management system is essentially two-fold—ensure that we keep what we have to keep and ensure that we get rid of what we don't have to keep. You might think that you should just keep everything but the costs associated with doing so become very high.

And when we talk about keeping information, we do not mean that you might need it one day just to prove or disprove something in a lawsuit. We are talking about the intellectual capital of your organization. For example, as a user I create collateral as I work on projects, and at any point in time I will be working with some artifacts that are current—say a PowerPoint presentation for a conference. When I am finished with the conference I, as a user, eventually have no more interest in the presentation, as it is not current in my eyes, and I could just decide to delete it. But the organization as a whole will benefit if it is kept so where should it go? Records management can help encourage end users to follow behaviours that are good for the company as a whole (e.g., not deleting content that might be of use elsewhere!).

So records management is really all about storing business content for the long term in such a way that it is easy to file and easy to find. But what is a record? Actually, it can be anything, either electronic or physical (be it a printed copy of an electronic document or a physical asset such as hardware). Each organization needs to decide for itself what it needs to keep, and then implement a system accordingly. The considerations that will accompany this effort includes deciding how to *identify* content, how to *transfer* that content from where it is to a records repository, how to *file* that content into its appropriate resting place, and subsequently how to *manage* that content once it is there.

The MOSS implementation of a records repository leverages many of the core services, such as content types to identify and file content and information management policies to manage the content. A "records manager" controls the routing and filing by means of a file plan, and there are features available to identify and put content "on hold" should you need to do so for legal purposes. For example, you may be under litigation and therefore do

not want to dispose of any content related to the lawsuit, and as a result, you do not want any expiration policy to run on that content. Figure 5.5 shows a representation of the records repository. Let's take a look at how you create it and use it.

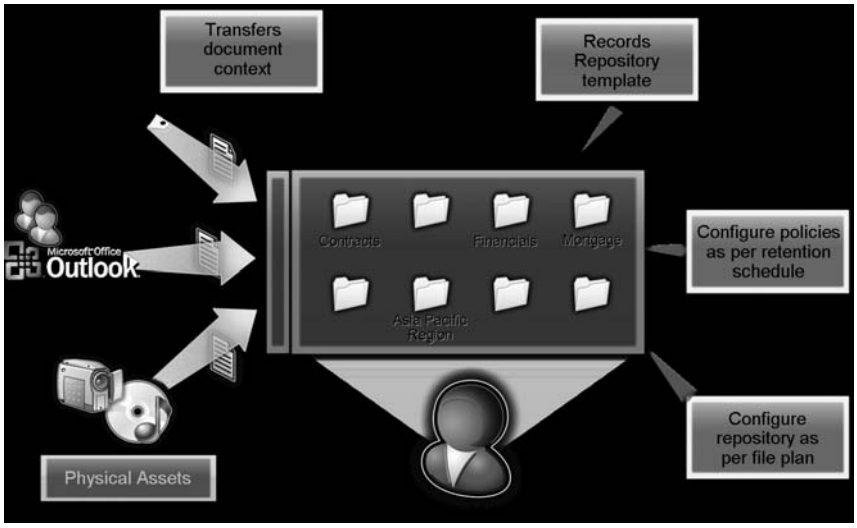


Figure 5.5 *Records repository.*

5.2.1 Records Repository Site

The first step in setting up your repository is to create a site collection using the OFFFILE template (the name comes from "official file" the name used for this feature during development). The resulting site looks pretty much like any other site and has a few libraries and lists pre-created. However, if you take a look at the names of the lists and libraries, you get a sense of the purpose of this site. For example, you will see libraries called Records Pending Submission, Unclassified Records, and Missing Properties, as well as lists called Record Routing and Submitted E-Mail Records.

Now that you have your site, begin setting up your file plan. Anyone involved in the records management space will be familiar with file plans; they basically tell you where to physically store something based on some attributes of that something. In SharePoint's case, you need to set up document libraries for the storage and configure your record routing. Routing occurs when anything hits the site, and the routing decision is based on the content type of the item. Thus, the content types you expect to receive will drive the creation of document libraries for storage. You can have any number of libraries and any number of content types defined in your file plan—and you can have multiple different content types being routed to a single document library, if your plan requires it.

Given that the routing is based on incoming content types, you can see why it is important to understand what content types are being used in your organization. If you are going to get the most out of your records center it's also important that you encourage their use! If no one uses content types, but everyone wants to send content to the records center, then everything will end up in one document library. Content types help classify information, but only if they have been planned carefully.

With that in mind, let's see how to define a filing rule in your file plan. The rules of your filing plan are stored in a list called Record Routing and there is one entry in this list by default. This entry is the catch-all for incoming items that do not match any other rule; it files the items in a library called Unclassified Records. The records manager would typically sort out any unclassified records manually, by manual filing or requesting more detail from the submitter to figure out where to file the item. This catch-all rule can be set to any rule you have defined in your filing plan.

Items are filed based on their content type, and it is very simple to set up the file plan. All you need to provide is the name of the content type, a description for administrative purposes, the library you want to file items into, and whether there are any aliases for this content type. Note that you enter the content type into the Title field within the Record Routing list, which doesn't seem all that intuitive at first glance. Also, the document library must already exist in the Records site itself, so you have to create the library first using the standard create document library option. Once a library is associated with a file plan, it is set so that it cannot be deleted. Aliases allow you to file multiple content types into the same library without having to maintain separate routing records for each one. There can be many reasons for doing this; one good example is different users who have used different content types to essentially describe the same content—perhaps for multi-lingual purposes, or because different departments use slightly different terminology. Given that an item itself can only belong to one content type, there can at most be one rule that will match the incoming type, so there is no issue with conflicts arising in the filing plan. It is possible to extend the processing that occurs during this routing stage and to call your own custom code. For example, once you have decided where the item is to be filed you may want to run some de-duplication code and do something special should a copy of the item already exist in the repository.

As a records manager, you set up your document libraries that will host items of different content types. As mentioned, you could have multiple content types heading for the same document library, which will require some thought in terms of the metadata that you want to capture since you cannot be sure whether the original items will contain the correct information. Content types are essentially defined at the site level, and it could be that the same named content type in one site has some mandatory properties already

on it, whereas another doesn't. What this boils down to is that each records document library has to be set up with the columns that need to have mandatory data in them according to the needs of your long-term repository. We'll now discuss the way content gets into the libraries and what happens if mandatory information is not in the original item.

5.2.2 Web Service and Populating the Repository

The primary mechanism for sending items to the repository is via the Send To menu, which appears when you hover over documents in libraries. WSS allows you to configure an option in the Send To menu for the purpose of sending the document to a records repository. But you need something to process the Send To command, and this is where MOSS comes into the picture with its official file Web service. Third parties could write their own Web service to process the Send To if required. The MOSS Web service is called officialfile.asmx, and it supports a method called SubmitFile, which is ultimately used to submit the document. Other methods allow you to gather information about the records repository in terms of its routing set up.

While you can have as many record repository sites as you want, each SharePoint site can only Send To a single repository, as the Send To option is scoped to the Web application. Thus every site within that Web application will Send To the same record repository. If you need to have more than one repository, you will need to have multiple Web Applications and partition the sites that need to submit content to specific repositories accordingly. It isn't ideal but then again the whole idea of a records repository is to have a centralized place for corporate item retention. Note that the Web application that hosts the repository can be different from that which holds the sites; indeed, this is a good practice, because it lets you more easily control the security aspect of the records repository without perhaps worrying about permissions being inherited or other such things.

You set up the Send To option in Central Administration at the Web application level and specify a descriptive name for the option plus a link to the Web service. This is shown in Figure 5.6, where we have created our records repository site called Records and then included a Send To the "Spy Vault" by specifying http://moss.spysrus.com/sites/records/_vit_bin/officialfile.asmx as our Send To link.

Now that you have the Send To link available and you've set up your record routing rules, it is simple for users to send a document to the repository. But what actually gets sent? First, all the current properties associated with the document and all its audit history are packaged up into two separate XML files, which are sent along with the document. In this way you always have a complete picture of what happened with the document before it was sent to the repository.

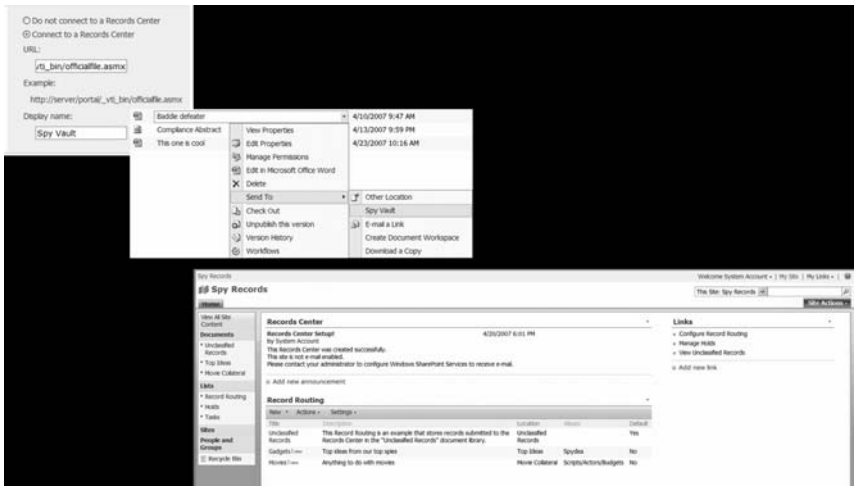


Figure 5.6
Setting up the Send To link.

When it hits the repository, the routing rules kick in and the destination library is determined. Any properties (columns, metadata) that have been defined on the destination library are now looked for in the list of properties on the source document. Any that match are promoted into the document library and are therefore available in views and so forth, like any normal property on a document library. If a property is defined on the destination library as mandatory and it is not on the source document, the user is prompted to provide the missing properties there and then. This is where the Missing Properties list and the Records Pending Submission library comes into play as a holding ground until the user provides the necessary details. We can see this in action in Figure 5.7, in which we have defined a mandatory field called Spy Name on our destination library for the filing of gadget content types. We have also defined the Idea field, which is automatically added to the destination library as it was provided in the source item. After supplying the missing properties (in this example, we put Ethan Hunt as the Spy Name) the document itself is placed into a system-generated folder within the destination library, and a unique identifier is appended to its file name. The properties and audit XML files are then stored in a subfolder with the same file name. If a document is sent more than once to the library, multiple copies will exist, but each will have a unique identifier appended to it.

Before we leave this discussion on filing items in the repository, remember that it is based on content type. You could, in theory, use the repository to record physical items as well as electronic items by ensuring that the properties on the item described the underlying physical item. Also information management features, such as generating labels and bar codes, could come in useful here for printing labels that can be attached to the physical item for subsequent identification purposes.

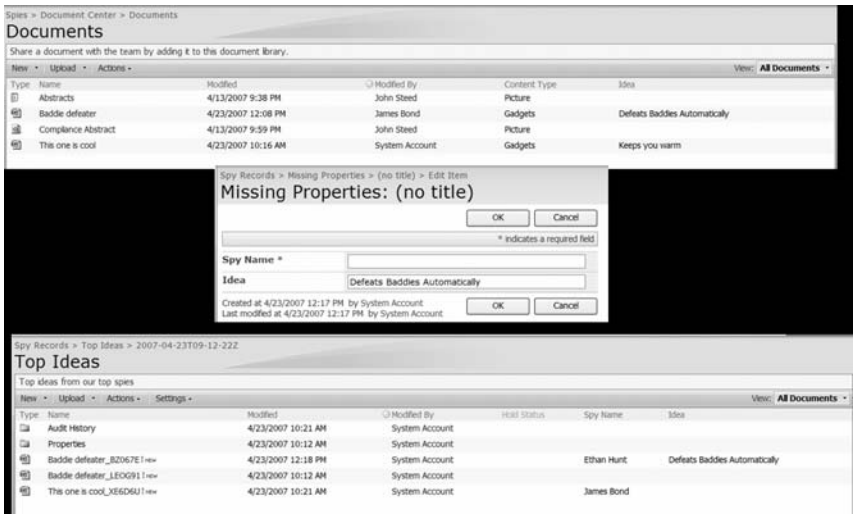


Figure 5.7
Missing Properties Dialog

5.2.3 E-mail and Records

The above explains what happens when documents from libraries are synchronously added to the records repository. E-mail is a common form of communication and is subject to the same compliance and regulation needs as documents. Therefore, your compliance strategy should cover e-mail as well as documents. Recall from Chapter 3, Collaboration that any list or library can be e-mail enabled and this, in conjunction with Exchange 2007 and Outlook 2007, will ensure that end users can contribute e-mail to the repository in such a way that it is filed appropriately based on your filing needs.

If incoming e-mail has been enabled at the farm level when you provision your records repository site, the Submitted E-Mail Records list is automatically mail enabled. If you need to mail enable this list after your site has been created, you only need to give it an alias of your choosing and ensure that attachments that are sent to the list are retained. You may also want to change the text of the Announcement item that is also provisioned, as it will say that the site is not mail-enabled!

There are two issues when filing e-mail. The first is that there is no notion of a content type being applied to e-mail in the programs where it is created; thus, e-mail that comes in to any SharePoint list or library is effectively unclassified. The second problem is that e-mail will be delivered to a list and library asynchronously, and there is no way to interact with the originator to ask them for the filing information as we could when using the Send To link for documents.

That said, there is support in Exchange 2007 for something called Managed e-mail folders and they serve a very similar purpose to information management policies that have been set on document libraries and lists. Managed e-mail folders are created by Exchange administrators and revealed to users through Outlook 2007 and OWA 2007. They are flexible in their design, enabling you to target different managed folders to different groups of users. From the end users' point of view, they behave just like any other folder except that they have handling instructions on them (akin to the Policy Statement in SharePoint) and can have retention policies applied to them. Additionally, these managed folders can be enabled with a journal rule which means that any item placed in such a folder has a journal report sent elsewhere—elsewhere being any SMTP address, which can be the address of your SharePoint Records Repository. An Exchange journal report is a message whose body contains details about the original message, such as who it was sent to; the original (and untouched) message is attached to the journal report.

This is where the link between Exchange 2007 managed e-mail folders and the records repository comes in. The repository will process these journal reports, detach the original message, and automatically file it according to the filing plan that has been set up. But what does it use to determine the content type of the e-mail? Each managed e-mail folder can have a label attached to it (configured by the administrator when the folder is created and is assigned the journal rule). This label is placed in the journal report from where the records repository can extract it and use it as the content type. So we now have a way of classifying e-mail based on the managed folder that the user files it under. The organization defines the classification top-level structure, and users have the freedom to create their own filing structures underneath. In this way, corporate retention policies can be implemented with the minimum of fuss on the end users' part.

The last part of this jigsaw is what happens if there are some mandatory columns on the destination document library—clearly, we can't ask for these synchronously from the user. Items that find their way into the submitted e-mail records list are processed daily by a SharePoint task called records center processing. When it runs, it extracts the labels from the body of the journal report, applies the routing rule, and then tries to move the item into the destination library. At this point, if there are missing properties, an e-mail message is sent back to the originator of the message. This e-mail provides a link to a Web page within the records repository that shows items with missing properties in a datagrid view. The user can then mass-update e-mail records that require more information so that they can be filed to their ultimate destination.

If you want to change the frequency of the records center processing task, you can do so via stsadm. Use the –o setrecordsrepositoryschedule option and add the schedule you'd like to the –schedule parameter. For

example, stsadm –o setrecordsrepositoryschedule –schedule "Every 5 minutes from 0 to 59". To see the valid syntax for schedules, do stsadm –help setcontentdeploymentjobschedule.

It should be clear now how managed e-mail folders in Exchange 2007 enable a corporate classification structure for e-mail and how end users can use that structure to correctly file e-mail in the records repository. Figures 5.8 and 5.9 shows the various steps involved in setting this all up. Note how we enter "Movie" as the label in the journaling rule associated with a managed e-mail folder. This is what eventually triggers the routing of the e-mail in the file plan. In closing, one thing I haven't yet made clear is that the journaling associated with managed e-mail folders can also be associated with default mail folders, such as Inbox, Calendar, and so on. With this capability you can have e-mail in these folders automatically sent to the repository without the user having to actually file anything into managed folders!

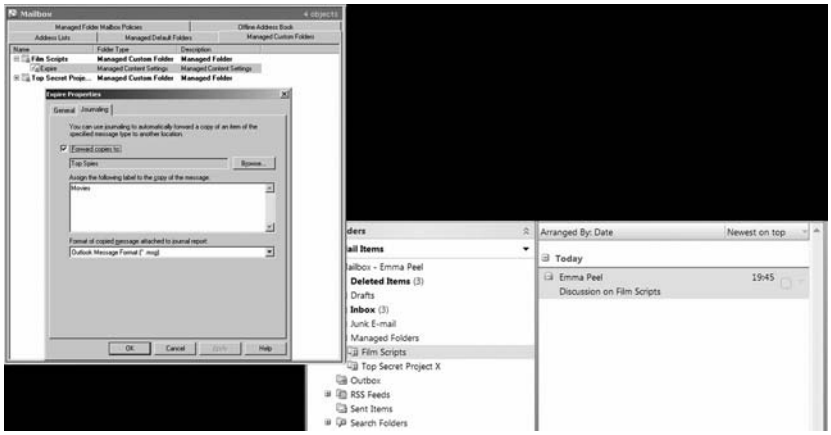


Figure 5.8
Setting up a managed folder.

5.2.4 Holds

The last feature for discussion in the Records Management space is the ability to put items on hold. An item that is on hold is not affected by any information management policies that may be in effect. Typically, items are put on hold if your organization is under discovery due to some litigation and you want to suspend the disposition of relevant information that may be needed during the lawsuit.

Placing items on hold involves setting up a hold and then identifying the items that need to be associated with it. When you view the controlling hold item, as seen in Figure 5.10, you can execute a search to do a mass discovery of items to add to the hold. You can also view the hold report and release the report from this interface.

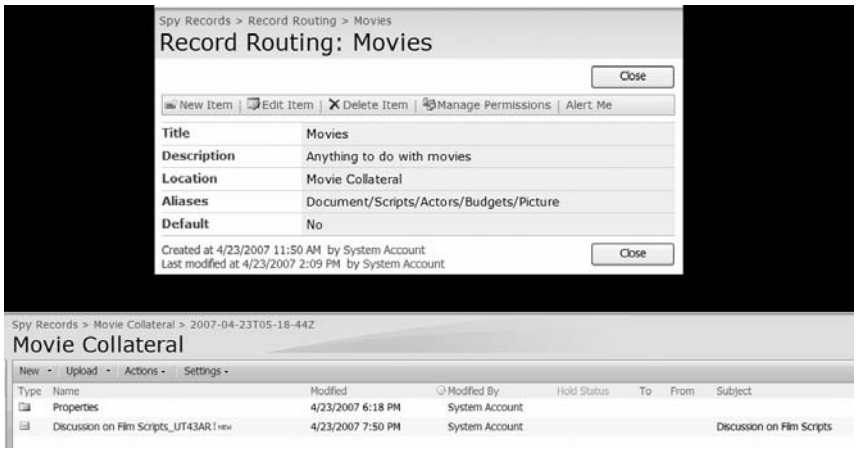


Figure 5.9
File plan for Movies.

Figure 5.10
Holds

5.3 Web Content Management

As mentioned in Chapter 1, 2007 Microsoft Office System, the Microsoft Content Management Server (CMS) is not being further evolved. Instead, its core features for creating, branding, publishing, and managing content-rich Web sites is included in SharePoint Server 2007—a feature generally referred to as Web Content Management (WCM). We believe this is a very sensible move for Microsoft, as it removes a common question about which product (CMS or SPS 2003) should be used for building Web sites. Now the answer is clear, since SharePoint Server 2007 has all the tools, features and resources

to build, deploy, and manage both internal- and external-facing Web sites, as well as sites designed for other purposes such as collaboration. Merging these two products also made sense from a technology point of view in that WSS provides many core services, such as check-in/check-out, item level security, and versioning, that are required as a foundation for content-rich Web sites.

So far in this book we have been mainly talking about the team-based, collaborative sites that SharePoint supports—sites that let users work with documents and lists and have these displayed in fairly simple Web Part Pages. When it comes to WCM, we are mainly talking about managing Web pages as opposed to documents and lists. By managing, we mean the physical design of such pages with all their supporting controls (style sheets, images, etc.), the creation and insertion of content into these Web pages, and the controlled deployment of these Web pages to a read-only user base. There are three major groups of people involved in this process: designers create Web page templates, content owners generate the content and site administrators eventually deploy the Web pages. The Web pages generally link together to form a Web site. A site that uses all the WCM features is known as a *publishing site,* a term we already talked about in Chapter 1 since the Collaboration Portal site has the publishing feature enabled.

In this section on ECM we are going to focus on managing the content and publishing of Web pages. Designing the Web pages is discussed in Chapter 14, Branding Your SharePoint 2007 Sites, where you will learn more detail about master pages, page layouts, and pages. For now, you need to understand a couple of basics in this area to be able to make sense of the rest of this section.

5.3.1 Publishing Site Templates

There are a few flavors of site templates that you can use as a starting point for your WCM needs. The Web site you see in Figure 5.11 was created via SharePoint Central Administration using the Publishing Portal template. It is only available at the root of a Web, but you can also create a blank publishing site, publishing site with workflow, and a news publishing site as a subsite inside any form of site collection. I think you will agree that it looks very different from a collaborative team site, and it certainly feels like a "Web site," but as you will discover, it uses the same WSS core features (document libraries, lists, content types, etc.) to achieve a different aim.

What you see here is the combination of master pages, page layouts, and pages with content that is physically stored in the galleries, lists, and libraries that are provisioned when the site is instantiated. Separation of content from presentation is key to having a flexible Web site with content

Figure 5.11
Publishing site.

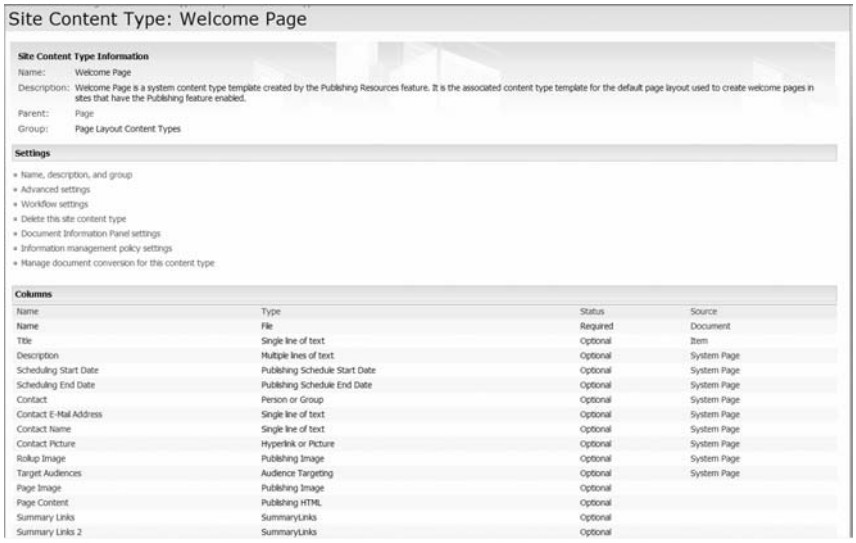
that can dynamically change but with a consistency in look and feel across the whole site.

Master pages let child pages inherit and reuse a common design and are used to implement the overall "visual blueprint" or "chrome" for the site. They typically contain the branding, navigation, header, and footer for the site. Page layouts allow a standard look and feel across Web pages that display different types of information. For example, Movie Press Release, Actor Biography, and Movie Synopsis display physically different information but all are made up of core data, such as title, description, and so on. You want the flexibility to be able to change the look and feel of such pages without having to change the core data. Page layouts are separate from the content they display, and designers specify *field controls* within the page layout that dictate where content can be added and edited.

Pages, which are ultimately shown to the end user, are actual items in a SharePoint library; the columns in such a library would typically hold the data that is rendered into the field controls in the page layout that is associated with the page. Therefore, core WSS features such as content types come into play here, as they control the actual data associated with specific pages in

the library. Also, given that these pages are in a library, in-built authoring controls and security allow pages to be edited and approved before being published for end-user consumption.

Thus, each publishing site has a Pages document library, and each item within the library is associated with a page layout. Each page layout is associated with a content type, and the field controls defined in the page layout are exposed through columns in the content type. Thus, populating the columns associated with the page layout ultimately results in the data being rendered into the page. The home page in Figure 5.11 (default.aspx) uses the welcome page content type, as you can see in Figure 5.12. You can see columns physically rendered onto the page, such as Page Image and Summary Links. Note also the different types of columns in this content type that are closely related to Web pages, such as publishing HTML and publishing image, which come from the Publishing and Page Layouts columns associated with the publishing feature.


Site Content Type: Welcome Page

Site Content Type Information

Name: Welcome Page

Description: Welcome Page is a system content type template created by the Publishing Resources feature. It is the associated content type template for the default page layout used to create welcome pages in sites that have the Publishing feature enabled.

Parent: Page

Group: Page Layout Content Types

Settings

- Name, description, and group
- Advanced settings
- Workflow settings
- Delete this site content type
- Document Information Panel settings
- Information management policy settings
- Manage document conversion for this content type

Columns

Name	Type	Status	Source
Name	File	Required	Document
Title	Single line of text	Optional	Item
Description	Multiple lines of text	Optional	System Page
Scheduling Start Date	Publishing Schedule Start Date	Optional	System Page
Scheduling End Date	Publishing Schedule End Date	Optional	System Page
Contact	Person or Group	Optional	System Page
Contact E-Mail Address	Single line of text	Optional	System Page
Contact Name	Single line of text	Optional	System Page
Contact Picture	Hyperlink or Picture	Optional	System Page
Rollup Image	Publishing Image	Optional	System Page
Target Audiences	Audience Targeting	Optional	System Page
Page Image	Publishing Image	Optional	
Page Content	Publishing HTML	Optional	
Summary Links	SummaryLinks	Optional	
Summary Links 2	SummaryLinks	Optional	

Figure 5.12
Welcome page content type.

5.3.2 Creating Pages and Modifying Content

When you create a page, you need to associate it with a page layout. This layout will determine what content you can add to the page and the content type that will be associated with your page. For example, some layouts are designed as article pages that just have a heading, image, and text, whereas others have Web Part zones into which you can place Web Parts. The net result will be the creation of an aspx page in the Pages library associated with the current site and marked with a suitable content type. To help you under-

stand the importance of associating a content type with a page, consider that you wanted to aggregate any press releases that had been published anywhere in your site collection and perhaps display the five most recent. In fact let's say that you only want press releases relating to a particular movie. The Content Query Web Part can do this for you and you configure it by indicating which content type you are interested in. You can then filter the items returned using any column from that content type—so you could easily filter on the column that contains the movie you are interested in. More detail on the Content Query Web Part is available in Chapter 13, Building and Deploying Web Parts.

Once you have created the page, you can provide or modify the content that will appear on it. There are a few ways to modify the content on a page; in Figure 5.13, we simply use the Edit Page option to do so through the browser. This checks out a copy of the page for our personal use and opens up those field controls within the page where we are allowed to add content. Note the editing toolbar at the top of the page which allows us to check in our changes, start publishing approval, and so on. There are lots of options on the toolbar to help authors, such as viewing the edited page in a new window, comparing previous versions of the page and many more. Note also how the field controls on our page match to that of the content type shown in Figure 5.12. And just to prove that modifying the content of a page is as simple as editing the properties of an item in a library, take a look at Figure 5.14, where I am doing just that instead of editing the item through the browser. Not all the properties are modifiable through this method, but you get the idea!

Many other features support page and content creation, such as *variations, reusable content,* and *smart client content authoring.* Variations are used to keep different versions of your Web site, perhaps to support different browsers or different language versions. Reusable content is a site collection list in which you can store content that can be inserted on pages throughout your site. Changing the reusable content can result in all pages using that content to change. Smart client authoring allows you to generate HTML representations of documents through the document convertor feature, so if you have a lot of content held in, say, Word documents, you can very quickly generate an HTML representation of this and include it in a Web page.

In the end, the vast majority of the content, the aspx pages that render the content and support images, style sheets, and so on are all stored in lists and libraries inside a site collection. The normal tools are available for managing this content, including information management policies, workflow approvals, item level permissions, and so on. Let's now see how you go about deploying this content and keeping it up to date as the source content changes.

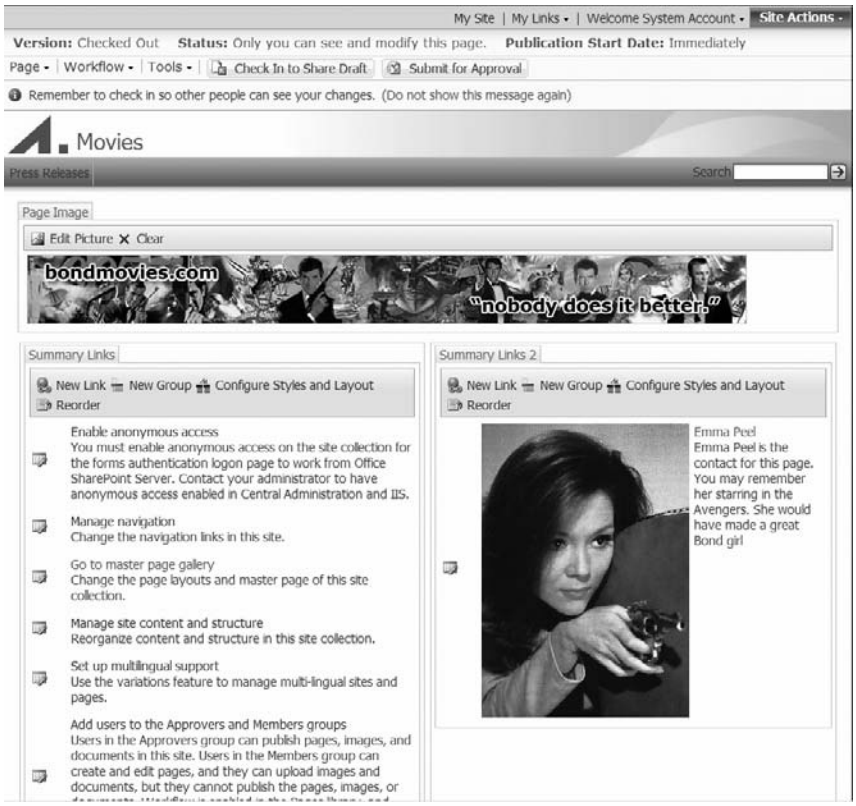


Figure 5.13
Editing a page.

5.3.3 Content Deployment

There may be many reasons to deploy content from one place to another—for example, taking a copy of a site locally for testing purposes, pushing a site to a remote geographical location on your intranet in a different farm, or performing an Internet deployment via an authoring -> staging -> production route. Ultimately, what you are doing is moving content from one site collection to another—and this can be for any type of site collection not just those with the publishing feature enabled!

There are three basics steps to content deployment: export from source, transport to destination, and import at destination. You first need to define a path that indicates the Web applications, site collections, and authentication information required to be able to connect between the source and destination sites. Second, define a job that does the actual export, transport, and import. Each job is associated with a path that indicates which sites to deploy and what schedule to do it on. In fact a job can specify only subsections of a site that need to be deployed. This allows you to, for example, update the

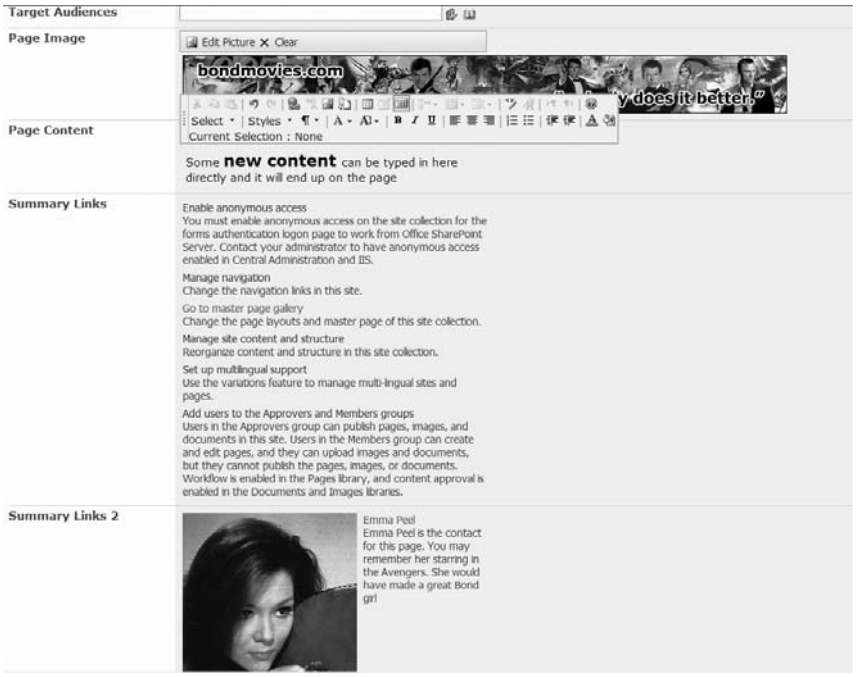


Figure 5.14
Editing properties of a page

Press Releases section of the Internet-facing read-only version of your Web site every hour, and perhaps only update other information on a weekly basis.

The actual content deployment job is smart enough to pick up dependencies that might have changed, even if the source of the dependency is not in the section of the site that is currently being deployed. So, for example, let's say my Press Release has a link to an image that exists in another part of my site collection. My hourly Press Release content job will scan for links and check to find whether that source content has changed. If so, it will be deployed along with the Press Release.

Central Administrators define paths and jobs at the farm level. For paths they provide details of the source and destination Web applications, the destination Central Administration Web site, and the source and destination site collections. The actual transport and import to destination is done via a Web service that runs in the Central Administration Web site called /_vti_adm/ContentDeploymentRemoteImport.asmx. The permission to execute such an import has to first be configured on the destination farm using the content deployment settings within the Operations tab. Defining a job involves associating it with a path and then defining the sites within the source collection that you want to deploy (all or specific sites). You then specify the schedule (and this can be a one-off deployment), whether you only want changed content to be deployed, and an e-mail address to send a report to. Addition-

ally, you can specify whether you want user information and roles and memberships to be deployed. Note that the source and destination site collections must have originally been created from the same site template, otherwise, the import will fail.

As mentioned, defining paths and jobs is a farm-wide activity and so is performed by a Central Administrator. But there may also be the need for site collection administrators to deploy content outside the defined job schedules, so what can we do here? Perhaps a critical press release needs to be deployed to your Internet-facing site immediately. A special deployment job called Quick Deploy is automatically created for any defined path for a site collection with the publishing feature enabled. Any site within the path that has the publishing feature enabled can then quickly deploy any page. The user who performs the quick deploy must be a member of the Quick Deploy group in the source site collection (created as part of the publishing feature); they can then initiate a deployment from the editing toolbar. The Quick Deploy job itself is scheduled to run every fifteen minutes by default, but can be changed by modifying the Quick Deploy job through Central Administration.

This page intentionally left blank

6

Search

There cannot be many people out there who have used a computer and not used a search engine such as Google or Windows Live Search. Searching for information is a very common task in both the commercial and business world, and trying to ensure that people actually find what they are searching for is a major challenge.

The challenges exist for many reasons—not least the fact that often, people do not know what it is they are trying to find! What I mean is that you might be doing some research into a particular subject area, but you have no idea what the corpus of information on that subject contains—so how can you possibly tell if you have found everything? Couple that with the common issue of people knowing what they are looking for but not looking in the right place, and you begin to understand the challenges that face people who *design* the search experience for others.

People tend to look for things using two primary mechanisms: browsing a hierarchical structure or searching via a full text search (in which the text also includes the metadata). A simplified explanation of these two methods is that browsing is used when you don't know exactly what it is you are looking for but you know roughly where it should be located. Full text search is used when you know what you want but have no clue where it is located. As I said this is a simplified (possibly even simplistic) explanation, since you can certainly use both methods regardless! But it serves to highlight a couple of major issues for people who design the search experience.

The first issue is how do you build *taxonomies* that will be useful to your consumers? How can you build a classification structure that will make sense to people browse that structure? This can be a very emotional, political and -drawn-out process! If you have a known clientele and a very narrow subject matter, you may be successful. But if you do not know your browsing clientele, how can you possibly know if they are equipped with the base knowledge required to make sense of *your* classification structure? Can you make your browsing hierarchy so obvious that people will navigate it seamlessly? Personally, I see this as an incredibly difficult task. I am not saying it is

unimportant, but the amount of effort you put into it should be weighed against the amount of benefit you think will result for your consumers.

The second issue is related: even if you do have classifications that everyone understands, how do you get people to apply the correct classification or ensure that the correct metadata is stamped on content? For me, the best person to describe what something is is the originator of that information—but will that person know what metadata they need to apply to the information at creation time so that it is subsequently useable down the line?

When it all boils down, people tend to both browse and perform full text searches for information, so let's take a look at how SharePoint Server 2007 supports all those things that can influence whether your corporate information is ultimately very useful or just gathering bit-dust on your expensive storage repositories.

6.1 A Common Search Engine

A common search engine has underpinned many Microsoft products throughout the years, but the end result for consumers was influenced by how much (or how little) the individual products exploited the capabilities in this engine. The best example of this at hand is WSS 2.0 and SPS 2003. WSS 2.0 leveraged SQL's full text implementation, which was fairly limited, whereas SPS 2003 actually enhanced the core engine and brought all its capabilities to the table.

WSS 3.0 now uses the same search technology as SharePoint Server 2007, although there are still some differences in the net features that are available to a pure WSS 3.0 team site as compared to a team site provisioned via SharePoint Server 2007. We will point out these differences as we address each search topic. In a WSS 3.0 deployment, search is delivered by the service called Windows SharePoint Services Search, but when you deploy SharePoint Server 2007 on top then the service called Office SharePoint Server Search takes over for all SharePoint content searches. With one exception (the SharePoint help system itself relies on the Windows SharePoint Services Search service) you should find both services running any SharePoint Server 2007. The executable for both services is the same—mssearch.exe—but there are two copies of it on the disk.

6.1.1 The Core Engine

The components of the core search engine have not changed dramatically since SPS 2003, but there have been significant changes in the way that it is consumed by calling applications. A full text index exists to support the efficient retrieval of relevant information from a collection of sources in response to user queries. If an index is to be truly useful, it must be able to support the

content sources that users are interested in, the type of data held within those content sources and rich retrieval capabilities. The core engine provides support to other products by creating full-text indexes on content and properties of structured and semi-structured data and by allowing fast linguistic searches on this data.

Although the engine supports the actual index, it is up to the consumers of this index to provide the content to be indexed, to format user queries, and to return the results of searches to the requesting clients. As such, the consumers effectively control the overall search experience, and different products (i.e., consumers) do much better in this space than others. As an example, consider that MOSS can provide content to be indexed from many sources, such as Exchange Public folders and Lotus Notes database, whereas WSS 3.0 can only provide content from team sites.

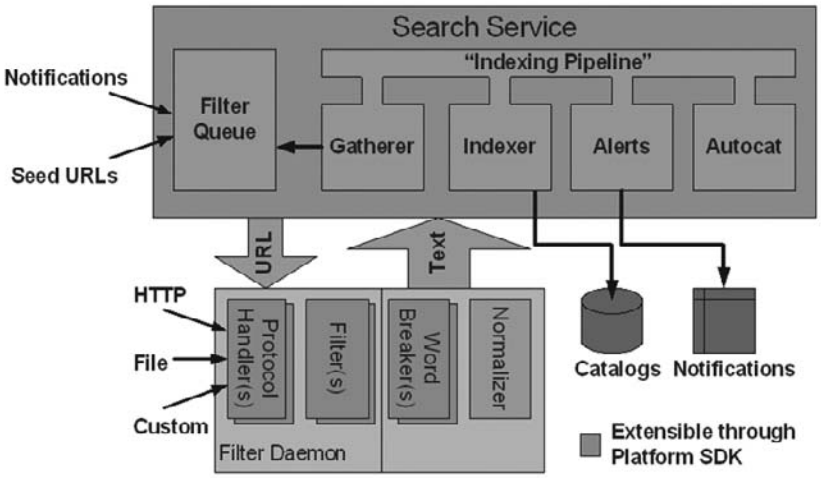


Figure 6.1
Core search engine.

Figure 6.1 shows the architecture of the core engine. Let's take a look at some of the major components:

- **Protocol handlers**. A protocol handler can access data over a particular protocol or from a particular store. Common protocol handlers include the file protocol, Hypertext Transfer Protocol (HTTP), Messaging Application Programming Interface (MAPI), and HTTP Distributed Authoring and Versioning (WebDAV). The protocol handler processes URLs passed to it by the gatherer. Protocol handlers know how to open up a particular type of data store and extract the individual items that will subsequently be indexed. Protocol handlers are extensible. For example, there are protocol handlers available for

D**o**umentum, and Exchange Server 2007 introduced a protocol handler for indexing messages in mailbox databases.

- **Gatherer**. The gatherer maintains the queue of URLs to be accessed across protocols. For example, a Web site crawl may include hundreds of pages and create network traffic by accessing each Web page one at a time. To be more efficient, the gatherer interleaves URLs from a remote Web location with URLs from other Web locations, or with access to file system documents or other stores. For each document accessed, the gatherer fetches the stream of content from the protocol handler and passes it on to the appropriate filter.
- **Filters.** Filters (also known as iFilters) extract textual information from a specific document format, such as Microsoft Word documents or text files. For example, Microsoft provides the Microsoft Office filter and the Office OpenXML format filter, which can extract terms from Microsoft Word, Microsoft Excel, Microsoft PowerPoint, and OneNote. Other filters work with HTML, XML, or e-mail messages. There are also third-party filters, such as the PDF filter provided by Adobe. The filter's task is to extract a stream of textual information from a document, discarding all nontextual and formatting information. The filter produces strings of text and property/value pairs, which are passed in turn to the indexing engine.
- **Word breakers and stemmers**. A word breaker is a component that determines where the word boundaries are in the stream of characters in the query or in the document being crawled. A stemmer extracts the root form of a given word. For example, "running," "ran," and "runner" are variants of the word "run." In some languages, a stemmer expands the root form of a word to alternate forms.
- **Indexing pipeline.** The pipeline is a series of functions that want to do something based on the text that eventually comes from the data source. The *indexer* prepares an inverted index of content. An inverse index is a data structure with a row for each term. In this row, there is information about the documents in which the term appears and the number of occurrences and relative position of the term within each document. The inverse index provides the ability to apply statistic and probabilistic formulas to quickly compute the relevance of documents.
- **Catalogs**. Full text indexes are stored in one or more catalogs within the file system on the server running the search service. Applications interact with the search service to maintain the indexes and the catalogs.

Note that the core engine is used not only by server side products but also on the Desktop. Windows Desktop Search, which is also used by Outlook 2007, leverages the same core engine.

6.1.2 Goals for the Search Experience

Even though the architecture of the core engine has not changed that much Microsoft has made many improvements in its administration and usage, with the net result being a better all around experience for the consumer.

Relevance is a key goal for SharePoint Server 2007. We are used to typing in a search term into Google and then watching the computer shudder as the avalanche of search results are displayed. Seldom does a search return few results, and you can never quite be confident that the results at the top of the list are actually the ones that are best suited to the query you posed. There are two reasons for this: first, as human beings, we are lazy and tend not to give the search engine too much input to go on! But second, it is very hard to judge which result is more relevant than another when searching across a wide corpus of unstructured and unrelated information repositories, which is typically what you are doing in a commercial search engine.

Many factors can be used to judge the relevance of a particular item according to the search terms, e.g., where the search terms appear in the item, how many times the terms appear compared to the size of the item, and so on. Relevance ranking has been in SharePoint since SPS 2001 (when the Okapi Probabilistic Ranking algorithm was introduced), and SharePoint Server 2007 makes it even better by including much more information about each and every document in the index. If certain metadata is missing (for example, the Title property of an Office document) then the indexer will create this data from the contents of the document.

Other new ingredients such as *click distance* help determine relevance. This is the distance a document is from a given authoritative site, such as the home page of your corporate intranet. If you think about it, links actually turn an unstructured repository into a structured one, allowing you to tie relevant pieces of information together. Thus, these links can be used to judge relevancy. *Anchor text,* the text of hyperlinks, is also considered when determining relevance. *URL depth* is another consideration for relevance; it looks at how buried a document is within a site. To ensure that rich metadata is available for all items in the index, certain standard properties are checked for. Should they be missing on the source item, they are automatically generated and placed in the index at indexing time. An example would be a text file that has just been dumped into a document library. Although it won't have the Title column set in the document library, the index will use the first sentence of the text file as its Title for display and relevance purposes.

A *familiar search experience* across all applications is also a goal for Microsoft (and the end user!), and this is being realized through the use of the common search engine. As mentioned previously, Windows Desktop Search, which is used by Windows XP and Vista, is also used by Outlook 2007. Exchange 2007 also harnesses the common search engine so that applications like Outlook Web Access and Exchange ActiveSync also provide a similar search experience to the end user.

Comprehensive indexing is also a goal, and the protocol handler and iFilter architecture help here. The BDC is also crucial here, as items held in back-end business applications can also be indexed and returned in search results.

Lastly a *flexible, scalable, manageable, and secure* search system is a key goal. SPS 2003 suffered here, particularly in larger enterprises where, for example, it was difficult to keep large indexes up to date on query servers due to the way index propagation from index servers functioned.

6.2 Configuring Search

Let's take a look at the major components that you need to think about when configuring search.

6.2.1 Managing Search

Search is a MOSS shared service and, as such, is managed via the SSP Administration option in Central Administration. With SPS 2003, the entry point into the management of the various components of search was a bit untidy and resulted in navigation frustration. In SharePoint Server 2007, the overall administration has been much improved, and a bird's-eye view of all administration tasks is available from a single page. This page, shown in Figure 6.2, groups the management tasks into three sections: crawl settings, scopes and authoritative pages. Note that for WSS there is no graphical user interface to search. It is enabled by default, and you have to use stsadm to make any changes to it. Figure 6.3 illustrates the use of stsadm to list the search configuration on a WSS farm.

The crawl settings section gives you access to the options that let you manage the content of the index: content sources, crawl rules and log files, file types, property mappings, and so on, as well as giving you an overall view of the status of your indexing operations. The scopes section allow you to manage what users see in the search scope pull-down on a standard search page, but the main end-user customizations are performed in the search results Web Part pages, as we will discuss shortly. Authoritative pages are used to define Web sites that are tagged as being authoritative or nonauthoritative; they ultimately affect the relevance ranking of search results.

As with SPS 2003, the physical full text index files are stored on NTFS, but there has been a change in the location of other parts of the search system. Details of much of the configuration, indexed properties, and other pertinent information such as anchor text from links are now stored in a SQL database associated with the search service. Note that the definition of content sources is still also stored in the registry on index servers (at HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office Server\12.0\Search\Applications\<appid>\Gather). Also, note that you have one index catalog per shared service provider. This again is a change from SPS 2003, but it simplifies the deployment and management of the index. Microsoft states that a single index can support up to 50 million documents, so a single index should suffice for most organizations.

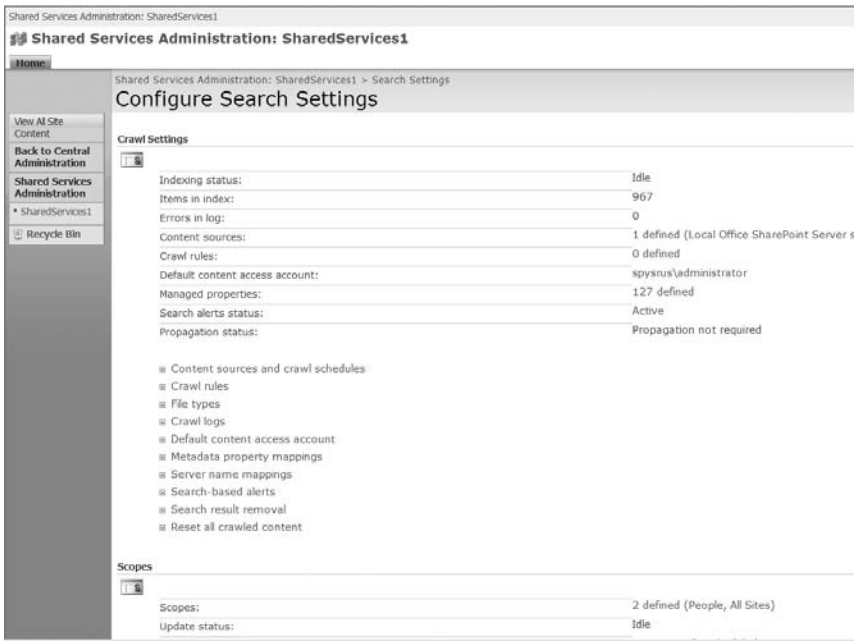


Figure 6.2 *MOSS search administration.*

A few areas of search have not changed that dramatically (or at all) since SPS 2003. This includes the thesaurus, which is used to augment or replace search query terms. The crawl log is easier to search in MOSS than it was in SPS 2003, and you still have control over the file types that the engine should attempt to index.

The ability to perform server name mappings—that is, the replacement of URLs in the result set that is passed back to calling clients still exists. This is typically used in situations where the content was crawled using an internal URL, but the clients need to access the results via an external URL.

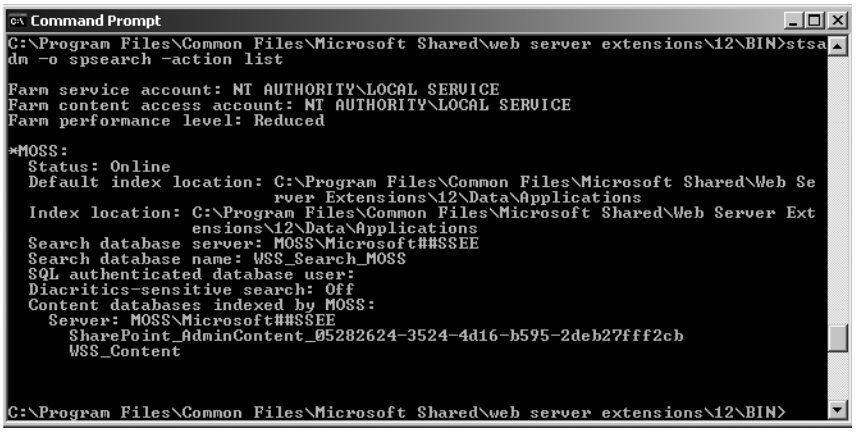


Figure 6.3
WSS search administration.

6.2.2 Content Sources, Crawl Rules, Site Hit Frequencies

Content sources are used to define the source content that is ultimately crawled by the index engine. You can think of content sources as providing the starting URLs to the gatherer; therefore, a content source is closely related to the protocol handlers that are available.

In a pure WSS 3.0 scenario, there is only one content source, and it is configured automatically for you when you enable search. It uses the SharePoint protocol handler to index the content of SharePoint sites on the WSS farm. On the other hand, SharePoint Server 2007 allows you to add as many content sources as you like and, out of the box, supports content source types of SharePoint sites, Web sites, file shares, Exchange Public Folders, Business Data (via the BDC) and also Lotus Notes. The latter requires that you install the Lotus Notes protocol handler, which is not done via a default installation. Third-party protocol handler installations will also extend the number of content source types available to you.

A single content source, called "Local Office SharePoint Server Sites," of type SharePoint sites is configured for you when you install SharePoint Server 2007. This will crawl all the sites on your farm, and you can control how often it does so by creating a full and/or incremental schedule. By default, no crawling is actually performed, so this is something you do need to configure. A change from SPS 2003 is that when an incremental crawl is performed, it uses the underlying WSS change log (an addition to WSS 3.0) rather than scanning every item in a site to see if it has changed since the last crawl. This significantly increases the speed at which new content is inserted into the index and makes the crawling process far more efficient.

The configuration of other content sources will be specific to their types, but in general you have to supply a name for the content source,

starting URLs and their scope, and a schedule for crawling the content. In SPS 2003, a content source only had one starting URL, but MOSS allows you to group more than one into the same content source. This gives you more flexibility in overall management. For example, if you have three non-SharePoint Web sites that you want to index, you can control the indexing through one content source rather than three. Of course there are other issues to consider if you do this; for example consider that all the content sources will be indexed at the same time since a schedule is now associated with each content source. Also, a start address can only be associated with one and only one content source.

Crawl rules are pretty much as they were in SPS 2003. They allow you to specifically include or exclude content based on a URL. So, for example, you may have a Web content source that points to the root of a Web site, but you do not want a particular subsection of that Web site to be indexed. In this case, you can specify an exclude crawl rule to omit the desired section. Crawl rules can also be used to specify different credentials to be used for different URLs that are being crawled and whether complex URLs (those with query parameters) are crawled. As with content sources crawl, rule definitions are stored in the registry on your index server at HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Office Server\12.0\Search\Applications\<appid>\Gather\Portal_Content\Sites*\Paths.

One welcome addition is search result removal, the ability to remove certain items from the index. You can never be sure exactly what links are going to ultimately end up in your index, and this gives you a quick way of removing any that are not wanted. As an administrator, you type in the URLs that you want to remove from the index, and behind the scenes this creates a crawl rule that will explicitly exclude that content, such that subsequent crawls do not reload these URLs.

Various other configuration settings that affect the indexing process are set at the farm level. You access them via the Manage Search Service option on the Application Management page of Central Administration. Crawler impact rules can be set here (how many pages to request at one time from a source and how long to wait between requests) and you can indicate a proxy server that the indexer should go through while accessing remote content.

6.2.3 Search Scopes

Search scopes reveal themselves in the user interface and allow searches to be scoped to a particular area of the index. For example, they can control whether a search is scoped to a list, site, site collection or type of item in the index (for example, only items that have something to do with people).

In SPS 2003, search scopes were very closely tied to content sources and were somewhat inflexible. In MOSS, this situation is vastly improved and

you can now build search scopes that span content regardless of which content source it may be in. In fact, you can now build search scopes based on properties of items so that, for example, you can have a scope that includes all documents of a particular content type regardless of which underlying site it resides in.

Scope rules define what is and isn't in a search scope and there are four different types of scope rules: Web address (URL, file, or hostname), property query, content source, and all content. You can mix and match rules within a scope, and there is also a "tie-breaker" configuration that you set when the matched items contradict each other. For example, you may have an item that matches one rule but not another, so should it be included or not included in the scope? We can see the creation of a property-based rule for a scope called Gadgets in Figure 6.4. Previous to this we indicated that the property called ContentType was a valid property for use in a scope rule. (Just a word of caution here if you are going to actually use content type in a scope. Remember that content types are created *per site collection*, and therefore you would need to ensure that the same content type, using the same *name* for the type, is physically created in all site collections). As you create scopes and rules at the SSP level, they must be compiled before they are useable. This is done automatically on a 15-minute schedule, but you can elect to manually kick off the compilation at any time.

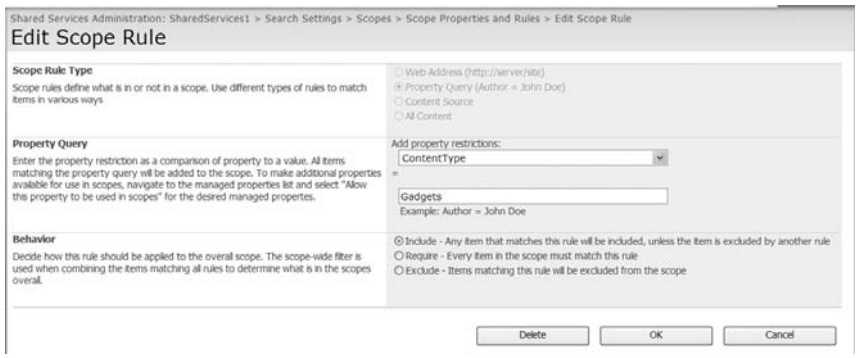


Figure 6.4 *Adding a scope rule.*

Now that we have a scope, it can appear in the scope drop-down in the standard Search Web Part. Two scopes are provided out of the box (All Sites and People) and we can see our new scope Gadgets, in Figure 6.5. We see two query result sets: one scoped to all content and one scoped to just items of content type Gadgets. It is also possible to use a separate search results page for each defined scope should you want to display the results in a special format. This is set on the search scope itself.

Search scopes can actually be defined at the SSP level or at the site collection level but will not show up in the search drop-down until they are

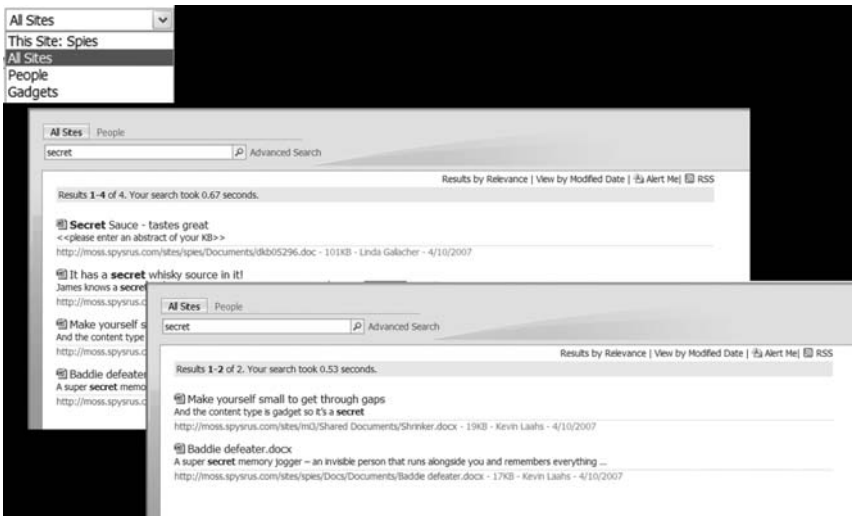


Figure 6.5
Search scope in action

applied at the site collection level. SSP-created scopes are known as *shared scopes* and can be applied to any site collection using that Shared Service provider. You apply scopes to site collections via Site Settings/Site Collection Administration/Search Scopes. Scopes are organized into groups, as can be seen in Figure 6.6, and site collection administrators can create new groups and scopes. Creating a new scope requires that the scope be compiled which, as mentioned previously, is done at the farm level on a schedule. Even though the scope is created at the site collection level, it is still effectively owned by the shared services provider and can still be deleted by a farm administrator. Furthermore, a farm administrator can chose to copy a site collection–specific scope, thus making it shared so that other sites can benefit from it.

As well as configuring the scopes for a site collection, you must also indicate to that site that it will use the scopes associated with a shared services provider. A good indication of a site not being associated with a shared services provider is the lack of scopes in the search drop-down or the ability to do an advanced search—which is usually the case with a standard WSS team site! Once this is done (via Site Settings/Site Collection Administration/Search Settings), you can then offer your users a search experience with scopes that are suited to the content held within the site collection. You associate individual scopes to display groups in a many-to-many relationship by clicking the display group itself and then indicating which scopes are in that display group. The site-specific display groups need to be configured by a site designer in the Search Box Web Part before they can be leveraged by end users.

Figure 6.6
Viewing site collection scopes.

6.2.4 Managing Properties

As you crawl content, your index will grow not only with the full text extract from the content but also by all the properties, or metadata, that describe the individual items in the content. Metadata can come from various places: columns in SharePoint lists, <META> tags in HTML content, or custom properties from Office documents. Properties can also be used to shape the search experience for end users. For example, you may want to define a search scope that uses a property set using a <META> tag on certain Web pages held in certain Web applications within your organization, or to allow users to select a specific property for an Advanced Search. It is also highly likely that a lot of properties extracted from content are completely irrelevant to your situation, and you may therefore want to remove them to keep things nice and tidy.

SharePoint Server 2007 lets you manage crawled properties using the Metadata Property Mapping option from the SSP Admin application. In an attempt to make this whole area more manageable, we see the introduction of something called a "managed property." A managed property is essentially a container object for one or more crawled properties for which the different properties contain the same information. Managed properties can then be revealed in search scopes and used in search queries.

As an example, consider something like First Name. There are many places within SharePoint where someone's first name will be stored—for example, as a property in a user profile, a column in a Contacts list, a custom column in a custom list, or even a property from a crawled custom source. It is usually too cumbersome and difficult for end users to know exactly where the data they are searching for resides, so a managed property can be used to group all the different places where First Name might be used together. We can see this in Figure 6.7. Out of the box, the FirstName managed property is associated with the First Name field from the profile and the "FirstName" and "First Name" columns in any list. We have added another crawled prop-

erty that we used in a custom list called "Also known as" to the managed property. The net result of this can be seen in Figure 6.8, where the user has issued a property-based query on the property called "firstname". The results, as you can see, come from the multiple locations where data representing someone's first name is stored.

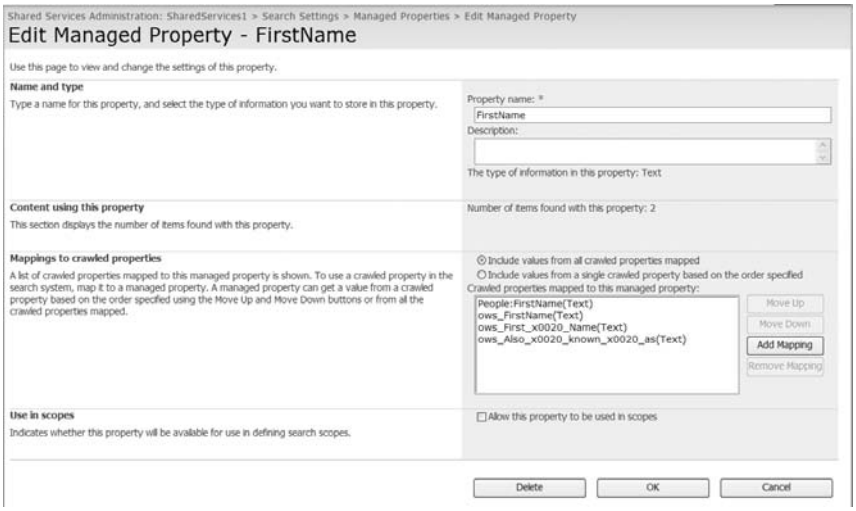


Figure 6.7
The FirstName managed property.

Figure 6.8
Search using managed property.

Managed properties can be configured to be available as a rule within a search scope. Crawled properties are grouped by categories (11 inbuilt and nonextensible), and you can also choose whether a crawled property is placed in the search index, as can be seen in Figure 6.9. The net effect of this is whether the value of the property will be matched in a search against the full text index or whether it is only matched when a property-based search is executed. The category with which a crawled property is associated is determined by the content source that it was crawled with and the type of the underlying item, and you cannot change which category a property belongs to.

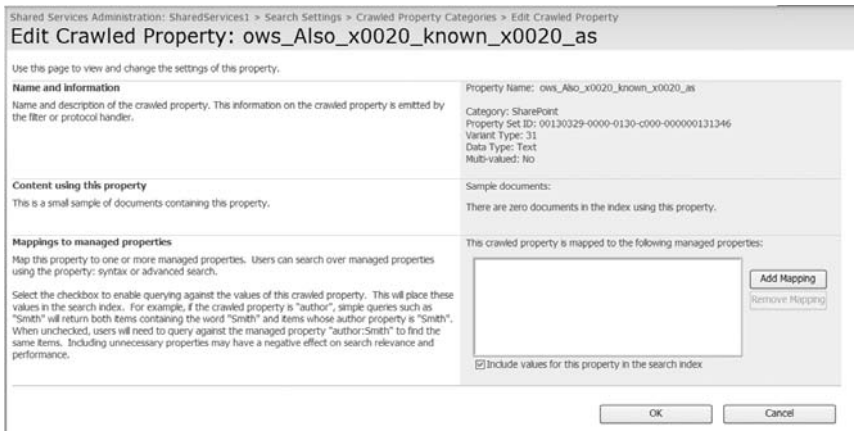


Figure 6.9 *Administering a crawled property.*

Just a word of caution before we move on from managed properties: remember that there is only one definition of a property as far as the index goes. Therefore, if you have two properties that have the same name but serve different purposes, then you have some thinking to do! A simple example will help explain here. Let's assume we have two site collections and each one contains a custom list with a column that the site administrator has created called "Band." The result of this is a single crawled property called ows_Band. Now, in the first site collection, Band may refer to a rock group, whereas in the second it might be being used for salary bands—so what do you do if one site collection wants to do a property-based search on band but the other doesn't?

There is no easy answer to this, especially since end users are empowered to create any metadata they like. The only suggestion we have is to think carefully about where crawled properties are actually being used before you attach them to a managed property.

6.3 Consuming Search

Search can be consumed from multiple places and in multiple ways—a search Web service and a full object model are available to allow you to exploit the index from within your own applications. We are now going to take a look at the various ways that you can consume search out of the box, starting with a look at the different ways you can formulate and execute a search query.

6.3.1 Search Query Syntax

There are three main ways in which you can formulate and execute search queries: via a keyword search, Structured Query Language (SQL) full-text extensions, or a URL search. The functionality that is ultimately delivered to end users depends on how these queries are formulated by the calling application. For example—in fact, for what is probably a very disappointing example to most folks—the search engine supports wildcard searches through the SQL syntax by using the CONTAINS predicate. But is there anywhere out of the box with SharePoint that actually formulates such a query? I'm afraid the answer is no, so either writing your own or finding a third-party (such as Ontolica) may be in the cards if you need this functionality. A very good community tool that can be used to generate queries is called the MOSS Query Tool and is available at www.wssdemo.com/Lists/Resources/DispForm.aspx?ID=893.

Keyword searches are the most straightforward and are used by the basic search Web Parts in SharePoint. As the name suggests, the query itself is a list of key words that should either be present or not be present in an item if it is deemed to have matched the search query. As well as constraining the result set based on key words, you can also add additional constraints based on managed property values. So, for example, you can execute a query that looks for certain words contained in the FirstName managed property, as we saw in Figure 6.8. Although I said earlier that wildcard searches do not surface in SharePoint, there is an exception here, as wildcards do work when you use a property filter constraint in your query. Thus entering "FirstName:Pa" would return all first names that begin with "Pa." There are also special inbuilt property filters that can be used, as we will see later with the use of the *duplicate* filter.

A keyword search supports multiple terms, which can either be a single word or a phrase (enclose the phrase in quotation marks: ""). A keyword search executes an implicit AND, which is different from that of SPS 2003, which executed an implicit OR. This means all the terms you specify must be in the item for it to be returned as a search result. But you can further constrain your keyword search by using + or - in front of a term. The + indicates that the term must exist in the content, and a - indicates that a term must not exist in the content.

Stemming—the ability to return content containing words that are closely related to the key word using inflectional variants—is supported, but has to be enabled in the Search Core Results Web Part. As an example, when stemming is enabled, a search for the word "Share" would also return hits on "Sharing" but not on "SharePoint."

SQL syntax queries give the most flexibility, with a rich syntax that allows you to perform very complex and specific searches including wildcards. You

would typically use the SQL syntax when you are building your own Web Parts that use the object model or applications that call the search Web service. The SQL search syntax extends the standard SQL-92 and SQL-99 database query syntax to leverage text-based queries and allows you to do multiple comparison and logical operations. The FREETEXT and CONTAINS predicates for the WHERE clause allow you to control the language that a query is executed in, which can be very useful in multilingual scenarios. As an example of SQL search syntax, the following is the near equivalent of the keyword search we issued to get the result shown in Figure 6.5. We say "near" equivalent, as we have only included a subset of the actual properties that are returned.

```
SELECT Title, Rank, Size, Description, Author FROM
portal..scope() WHERE CONTAINS ('"secret*"') AND ( ("SCOPE" =
'Gadgets') ) ORDER BY "Rank" DESC
```

URL searches are another way of formulating queries. They pass query strings to a search results page and enable a very simple way of providing links to canned searches from anywhere. Indeed, keyword searches ultimately call the search via a URL. So, for example, a call to http://moss.spysrus.com/sites/spies/searchcenter/Pages/results.aspx?k=secret&s=Gadgets would produce the same result as that shown in Figure 6.5. The page that you pass the query parameters to must have a search Web Part on it that can receive the parameters and execute the search—for example, the search core results Web part.

Table 6.1 *URL Parameters for a Search Results Page*

URL Parameter	Description	Example
k	Keywords. You can pass separate terms by separating them with %20	results.aspx?k=bond%20firstname:sean
s	Search scopes. You can provide multiple scopes by separating them with %20	results.aspx?k=bond&s=Gadgets%20Spies
v	The view to render the results in—can be either relevance or date	results.aspx?k=bond&v=date
start	The start page to render the view should the search results require more than one page	results.aspx?k=bond&start=3

The supported parameters you can pass to such a page are shown in Table 6.1.

6.3.2 Relevance, Best Bets, and Keywords

As discussed, there has been a great deal of work done in trying to ensure that lean and relevant results are returned to search queries. The SPS 2003 ranking calculation has been updated to include the following considerations:

- Click distance
- Hyperlink anchor text
- URL surf depth
- URL text matching
- Automated metadata extraction
- Automatic language detection
- File type relevancy biasing
- Enhanced text analysis

We've previously explained some of these and how they affect ranking, but most of them are effectively hard-coded unless you write your own search code (for example, you can use the object model to change the weight applied to a managed property at indexing time). Therefore, there isn't that much you can do out of the box to change the resulting rank given to a particular item, but you can configure authoritative and nonauthoritative pages in addition to editorial type ranking via best bets and keywords.

Authoritative pages affect the rank given to other pages linked to from the authoritative page itself. Therefore, you may want to promote a search result for a particular item if it is linked to from your home page of your corporate portal. Or perhaps you just want to keep a centralized page that is not viewed by end-users but has links on it to items within your corpus that you want to raise above others in search results. You can also set up *nonauthoritative sites* (AKA demoted sites) so that any item with a URL that belongs to such a site has its rank demoted (or decreased), thus pushing the items further down the search results. This can be very useful if you have some sites with stale content in them but, for whatever reason, have to keep them in your index. Administrators define authoritative pages and nonauthoritative sites at the SSP level.

If you want to execute editorial rights and highlight particular items in search results, you can use *keywords* and *best bets* to do so. Keywords are

words or phrases that you can define so that additional information is displayed in a search results page if the keyword was matched in the search query supplied by the user. When a search term matches a defined keyword, a definition of that keyword and a list of links (best bets) to other items that are relevant to that keyword are shown in the results page. Administrators define keywords and best bets at the site collection level using Site Actions/Site Settings/Search Keywords.

Associated with keywords are synonyms that the user might type instead, a contact for the keyword, and a start, end, and review date for the keyword. The start and end date can be very useful if you need to promote a certain topic for a particular period of time. For example, it may be that you have a limited time offer on a particular item that is for sale; specifying a keyword on that item will promote it into the best bets Web Part, which is on a typical search results page. Note that keywords are defined at the site collection level and, as you will recall, every site collection needs to be associated with a search center if you are to use the advanced search features that MOSS brings to the table from within that site collection. Keywords and best bets are such an advanced feature. The search center itself is a site that belongs to a site collection; therefore, the keywords and best bets that are shown in any search results page are those defined in the site collection to which the search results page belongs. Confused? No wonder!

What this ultimately means is that if you want to have different keywords for different site collections, you must create a search center site for each site collection. All the site collections will share the index and be able to use shared search scopes but will have the flexibility of having their own search result landing pages and keywords displayed on those pages.

A related feature to keywords is something called *definitions.* These are automatically extracted from content as it is crawled and can subsequently be displayed at the bottom of the search results Web Part. The idea here is that if you are searching for a particular term and there is some content in your corpus that has a definition of that term, you can be directed to that definition. We can see this in action in Figure 6.10, where we searched for the term EMS. There are multiple items in our repository that have this term in it but one individual document has some text in it that seems to be defining what EMS stands for. Thus, this particular document is highlighted in the "What people are saying about EMS" link, along with the definition that was extracted from that document.

6.3.3 Search Results Page and Search Web Parts

We've talked a lot about how the index gets built and how you manage it, but let's turn to what it all means to the enduser. Microsoft has done significant work in allowing the display of search results to be very flexible—this fits

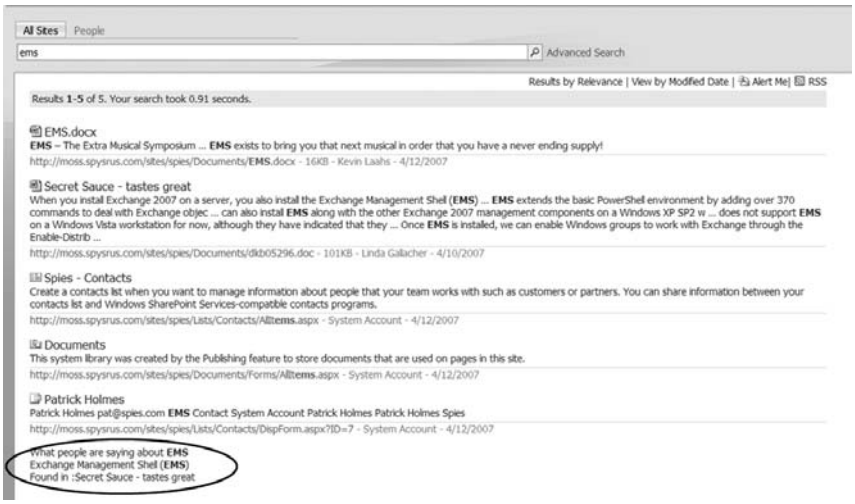


Figure 6.10 *Definition extraction.*

with the company's goal of leaner, more relevant search results and allows you to display search results in a way that is appropriate for the information that is being returned.

Any site in a site collection that has a search box Web Part on it needs to know where to display the results of a search, typically known as a search landing page. There are essentially three options that you can configure through normal site collection administration using Site Actions/Site Collection Administration/Search Settings. First, if you do not want to use MOSS search features, you can choose to offer standard WSS search, in which case the landing page is osssearchresults.aspx. This page limits search queries to the site and its children, is a nontabbed results page, and is noncustomizable.

The second (and third) option is to associate the site collection with a Search Center—this is a special type of site collection that can be instantiated from two different templates. The first template creates a Search Center with tabs and the other without tabs (we'll discuss tabs when we talk about searching the BDC). As we discussed earlier, the Collaboration Portal template automatically creates a tabbed Search Center and associates it with the site collection. Whether you need to create more Search Centers will be entirely up to your own particular situation and what you are trying to deliver in terms of search.

When a site collection is associated with a Search Center, the search box Web Part directs its results to a page called results.aspx in the Search Center site. Note that there is a slight issue here in that if your Search Center is a tabbed one, the results.aspx page lives in a document library called /pages, whereas if your Search Center is not a tabbed one, it resides in the top level of that site. So be careful when you set the link to your Search Center—it needs

to be a link to where the results.aspx page resides. The results.aspx for both types of Search Centers is a Web Part page that can be customized and contains many interesting Web Parts, as you can see in Figure 6.11—so let's take a look at them.

There is an anomaly with respect to the search landing page that is used when you are searching a contextual search scope rather than a custom search scope. We mentioned in the section on search scopes that you can associate a search landing page with each scope, and this is indeed the case. The anomaly is with contextual scopes, which are generated dynamically depending upon the current situation—for example, "This Site" and "This List." These scopes are treated as if they were associated with the standard WSS search landing page and therefore are not customizable out of the box. This is just something to be aware of, since users will notice the different result page when searching these scopes. This may or may not be an issue for you, but you can at least explain to users that these scopes, by their very nature, are much narrower in their focus and result sets and therefore do not require the extra features that you get on a Search Center results page.

Figure 6.11
Results.aspx.

Customizing the search experience in SPS 2003 was doable but certainly not simple and required writing custom code that would override the search results Web Part. With SharePoint Server 2007, it is now much easier to get the right result for your scenario by modifying the search results page and the Web Parts therein.

The search core results Web Part is the primary part, and the best bets and high confidence results Parts are merely derivatives of it. The difference between them is the result sets that they are passed from the search engine. The core results Web Part receives all the hits from the index, the best bets Web Part receives keyword matches, and the high confidence results Web

Part receives hits that are deemed to be perfect matches for the executed query. For example, if I enter "John Steed" as my search terms, and we have a user in our profile database called John Steed, his details will be displayed in the high confidence Web Part.

There are many tweaks you can make to the displayed results by editing the properties on the Web Parts. For example, on the core results Web Part, you can enable stemming and permit-noise word lookups (so that queries on terms such as "the," "in," etc. would actually return results). You may think configurations that would affect the actual query that is executed would be on the search box Web Part where the query terms are entered rather than the results Web Part that displays the results wouldn't you? Well, contrary to a normal thought process, it is not the search box Web Part that issues the actual search. The search query is actually submitted to the search component from a hidden object that is instantiated by the search core results Web Part on the results page. The hidden object then returns the search results from the search component to the results page for display via the various Web Parts. You can modify the search box Web Part to display scopes and to also enter default keywords that can be appended to the search. In fact, the search results Web Part can also be configured as a fixed query Web Part, which can be very useful if you want to always display the most recent contents in the index that relate to a particular set of keywords.

As well as controlling the number of displayed items per page and number of lines to display per item, you can also control the properties that are returned and supply your own XSL to transform the XML result set into any shape or form you like. This is where you can really change the search results to suit your own purposes, as you can retrieve any property that is in the index and display it in any way you like.

The list of properties that are returned are stored as a property on the Web Part as XML. Figure 6.12 shows the default XML, and you can see the standard properties that are returned.

Figure 6.12 *XML defining returned properties.*

You can add more properties to the returned list by editing the XML and then modifying the XSL to transform the returned property for display purposes. You can see the net result of this in Figure 6.13, where I have displayed the default property called rank, retrieved and displayed a new property called content type, and underlined and bolded the author's name. Note that the column names that you manipulate in the XSL must be lowercase, irrespective of how you define them in the XML!

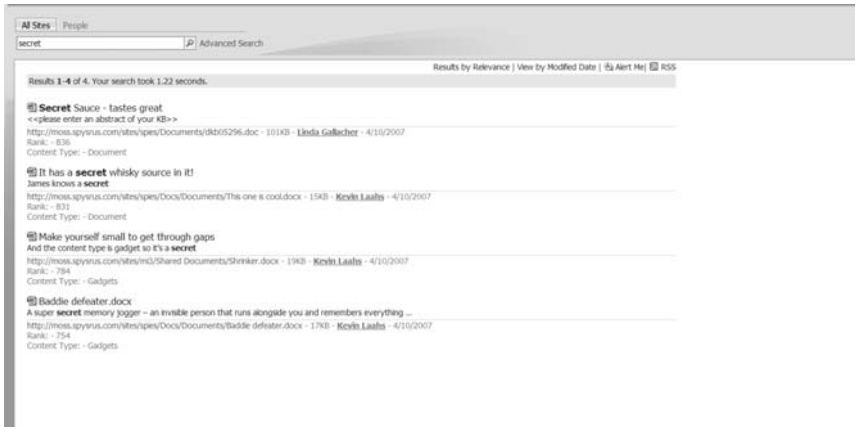


Figure 6.13
Modified search results.

The search summary, search statistics, and paging Web Parts are fairly self-explanatory, so this leaves the action links Web Part. It is also pretty self-explanatory, but it does have an interesting action that allows you to subscribe to an RSS feed for the search that generated the current results. This means that every time your RSS reader synchronizes with this feed, you will get an up-to-date search result set. This is a very powerful way of keeping track of information that has not yet even been published! Imagine, you have a favorite author, and you want to be told when he or she publishes anything new regardless of where it is or what it is? Now you can really search the future and be told the results right inside your favorite RSS reader.

Before we leave this section on managing search results, there are some other neat additions that make the search experience so much better than previous versions. First of all, we have a "Did you mean?" facility that will offer you suggested words and phrases that might return better search results. I'm sure you've all seen this in Google, but with MOSS the beauty is that it does not just offer you correctly spelled language specific suggestions—it actually mines your index and offers suggestions from there. If your index is filled with industry-specific terms, they can be offered in response to queries that are close to those terms.

Hit highlighting is available in search results and you can see how it works by analyzing the basic XSL in the results Web Part. It looks at the property

and, if it is a property that can support hit highlighting, it will search for the query terms within that returned property and bold them in the display. One interesting thing to note here is that the description property that is returned from the search engine and shown, by default, on the second line of the search result for an item is taken from parts of the item that match the search terms. For example, it could be sentences embedded deep within the document or a column in a list item. A good example of this can be seen in Figure 6.10. This again just shows that Microsoft has thought about how to make search results more meaningful.

The last item I want to mention is *duplicate detection*. It is not an uncommon practice for a document to be stored in multiple document libraries; indeed, it's a practice that is incredibly difficult to avoid, since people tend to want to hoard information in their own repositories! But it is intensely annoying when multiple search results from multiple team sites lead to the underlying document that is exactly the same, since you have no way of knowing this merely from the search results themselves. You tend to get confused as to which document is the one you want when, in fact, any of them would do. SharePoint can now detect duplicate documents and collapse them into a single item in the result set. In fact, it detects *near* duplicates so even if the items are only subtly different (either in the text or even the metadata), they can be grouped together as duplicates in the search results. Consider the situation where someone needs to create a presentation—what better place to start than another presentation that is very close to what they want. They only need to change it slightly and the result is that it is very nearly a duplicate of the original. How does SharePoint detect duplicates? It uses something that Microsoft terms "shingling," which is an algorithm that can tell when some things are around 95% similar. It does this using hashes as items are indexed, and these are stored in the search database—the hashes relate to the content of an item as well as its properties.

The user interface allows you to reveal all the duplicates with one click; this operation actually uses a keyword search using a special property filter called duplicates, which takes a URL as a parameter. You can see this in action in the two screen shots in Figure 6.14. There could well be more uses of this feature, such as duplicate detection and deletion if you need to clean up your information repositories!

6.3.4 Search Tabs and Searching the BDC

Out of the box, the search results landing page has two tabs: All Sites and People. What is the purpose of these multiple tabs, and where might you put them to good use? To explain this, I will take the opportunity to explain the role the BDC can play in enterprise search and how you can help people find what they are looking for by targeting searches to specific scopes with specific filters and by having a different landing page for such searches.

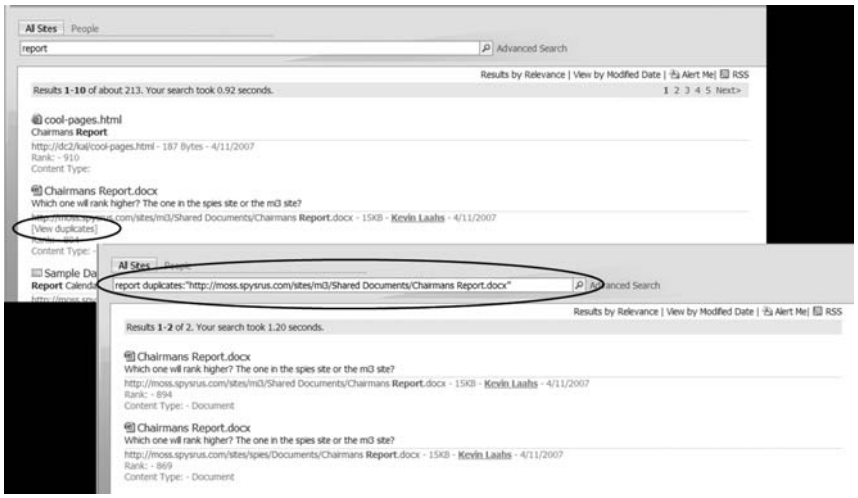


Figure 6.14
Near duplicate detection.

As we explained in Chapter 4, Microsoft Office SharePoint Server 2007, the BDC allows you to surface back-end business data and then use it within various places in SharePoint. The search system provides a BDC protocol handler which means that content in your LOB applications can be extracted, placed into the index and subsequently returned as search results whenever a search term matches. In order for this to work your application definition file must describe an enumerator method for the entities that you want to be indexed. Once this is done you can merely add a new content source to your index specifying business data as the source type. You are then offered the ability to crawl the full BDC or just a specific instance. For our example, I have crawled the Actor and Movie tables, and you can see the search result coming back from the tables in Figure 6.15.

In this example, the first item in the result set is from the BDC, but the end user would not know this. It, and other hits from the BDC, is intermingled with many other results from many different sources, so let's see how we can turn this into a much more targeted and focused search result by using tabs.

Tabs offer you the ability to quickly jump to a different search landing page so that you can modify this page to suit the needs of the specific information that the page is trying to convey. There are other ways to achieve this, such as associating a separate landing page for a search scope, but tabs are easier for the end user, since they do not need to know anything about which scope to search in. Check out the People tab and you will see that it uses a landing page called peopleresults.aspx. This page has a specific people results Web Part on it that displays the results in a way that makes sense, such as showing the results by social distance.

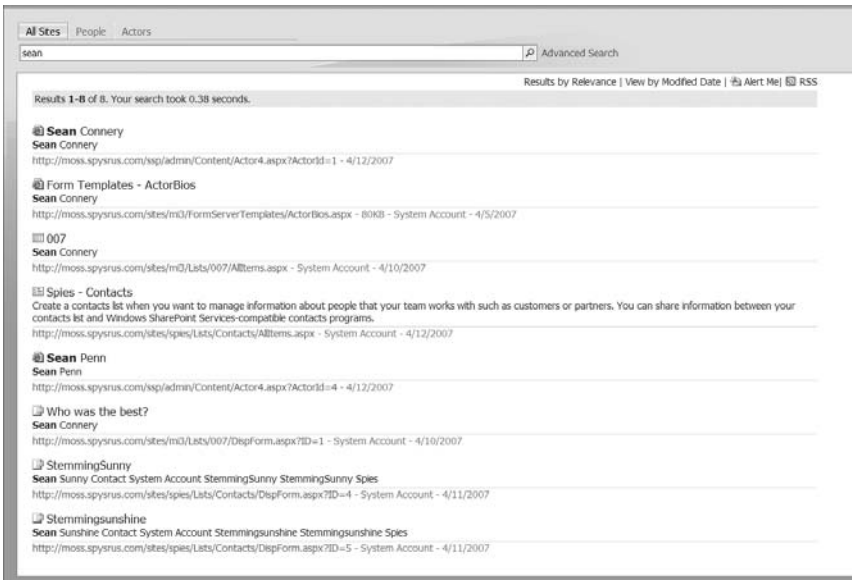


Figure 6.15 *Search results from BDC.*

So let's create a tab that will allow us to focus on Actors. We do this by first creating a couple of pages that will support the tab: for entering a query and one for displaying the results. When you create a page from a Search Center (Site Actions/Create Page), you are offered template pages that do both of the above, which is very handy! In addition, we can create a search scope that will limit any results to those coming from the BDC. This scope can then be configured into the search core results Web Part on our Actor results page, which means any search will immediately be targeted at the BDC without the user having to select any scope. Additionally, we tell the search box Web Part that it should use ActorResults.aspx as its search results page. You can see this in action by comparing Figures 6.15 and 6.16, the only thing the user had to do was to click the Actors tab and that immediately filtered out all the other noise results.

Figure 6.16 *Targeted BDC results in separate tab.*

Now that we have our own landing page for Actors, we can do whatever we like to it—for example, modifying the master page, page layout, Web Parts, and so on. For our example, we will tweak the XSL in the search results Web Part so that we can provide a pivot link that will search the Movie table, which has been published through the BDC. To make this happen, all we need do is to create a managed property (in our example we called it movie) that links to the actorid column in our Movie table, which has been made visible through the BDC. Then we can use a property-constrained keyword search to return all the movies in which that actor has appeared. The net result of this can be seen in Figure 6.17. The top graphic shows that we used our Actor tab to find all actors called Sean. From here we took the Movies link for Sean Connery, which called our search results page with a property constrained search for all movies whose actorid is 1, and you can see the result in the bottom graphic. Note how the actual search that was performed was movie:1.

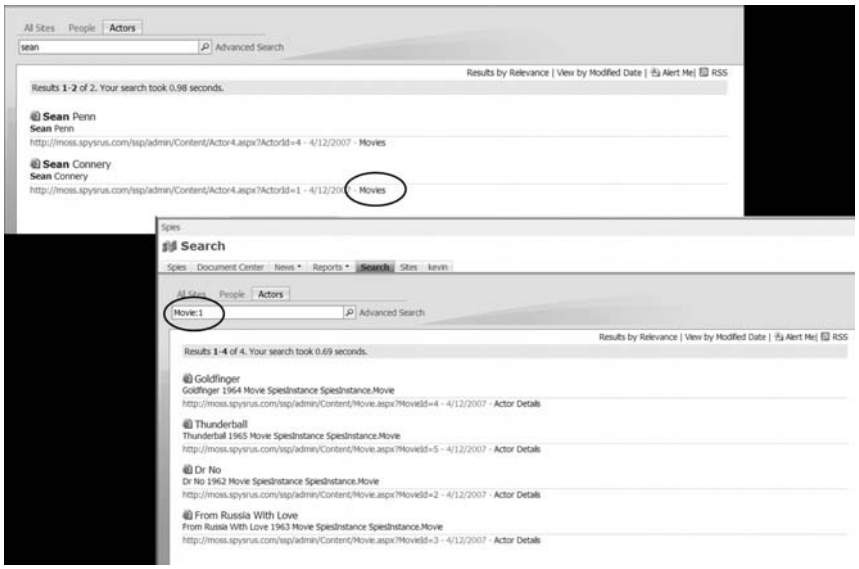


Figure 6.17 *Pivot searches—find movies from actor.*

I think you will agree that the power and flexibility of SharePoint's search functionality, coupled with the BDC, offers a very effective way of making sure your users find the information they are looking for, regardless of its location within your information repositories.

6.3.5 Making Sense of Searches through Search Reports

Understanding what searches are happening on your index can help you shape a better experience for your consumers. For example, if you know that

certain keywords are being used regularly in searches, you may want to create a keyword best bet for them. Alternatively, you might find that there are a lot of zero hit search results, in which case you may want to expose so more properties somewhere, so that these types of searches will succeed.

MOSS actually stores data about the searches that are being performed in the search database and then provides a set of canned reports that let you see what is going on. Search usage reports are accessible from SSP administration and include details like most popular searches over a scope and queries that returned zero hits.

MOSS captures the searches by calling the search Web service after every keyword search is executed. The search Web service, search.asmx, supports a method called RecordClick, which passes up the query terms and the search pages that executed them to the server. From here they are stored in the search database for subsequent analysis.

6.3.6 More than One Search!

In closing this chapter on search, we wanted to make you aware of the fact that Microsoft offers a separate product that can be used solely for search. It is called Office SharePoint Server 2007 for Search. It is more functional than WSS, but less so than a full-blown MOSS installation. Table 6.2 indicates the particular features that you get with each search product.

Table 6.2 *Search Features*

Feature	Search in WSS 3.0	Office SharePoint Server 2007 for Search	Enterprise Search in MOSS 2007
What can be indexed	Local SharePoint content	SharePoint sites, Microsoft Exchange Server content, file shares, Lotus Notes, custom content.	SharePoint sites, Microsoft Exchange Server content, file shares, Lotus Notes, custom content, line-of-business (LOB) content
Rich, relevant results	Yes	Yes	Yes
Alerts **RSS** **Did you mean?** **Collapsing of duplicates**	Yes to all	Yes to all	Yes to all
Best bets **Results removal, Query reports**	No to all	Yes to all	Yes to all

Table 6.2 *Search Features (continued)*

Feature	Search in WSS 3.0	Office SharePoint Server 2007 for Search	Enterprise Search in MOSS 2007
Search Center / Tabs	No	Search Center without Tabs	Search Center with Tabs
People search **Knowledge network**	No to all	Yes to all	Yes to all
Business Data Search	No	No	Yes
Query Web service	http://<site>/_vti_bin/spsearch.asmx	http://<site>/_vti_bin/search.asmx	http://<site>/_vti_bin/search.asmx
Security trimming of search results	Yes; supports default security trimming only	Yes; supports default security trimming only	Yes; supports default and custom security trimming

7

People

We stated in Chapter 1, 2007 Microsoft Office System, that SharePoint is focused on connecting people, process, and information across organizational boundaries. You may recall that in SPS 2003 we saw the introduction of people as a "first class object," which meant that there was specific processing for information related to people. We therefore saw the ability for users to have their own team site—known as a My Site—which had public and private views on it. We also saw the user profile, which could be indexed and details from it returned in search results, as well as target certain types of information toward users through something called Audiences.

All these features did indeed offer some specific support for people in an SPS 2003 portal, but there was not much in the way of connecting people together or exploiting the fact that we could indeed differentiate document-based information from people-based information. For example, if you issued a keyword based search query and got a hit on a person's name, that search result would be returned to you along with other hits, such as keywords in documents or list items. Furthermore, there would not be much clue about why a particular person was being returned to you in a search result—was it because the person has some knowledge about the search queries you entered, or because they happen to share the same surname as one of the keywords you entered? In essence, there was not much support in terms of helping you make sense of the result set and the relationships that you might have with the people information returned to you.

So, building upon the lessons learned with the SPS 2003 implementation, we see some great improvements in the way people information is harnessed, which ultimately results in better connecting people to other people and people to information. The basis for all things people is the user profile; other functional areas, such as the personal portal capability (My Site) and targeting information (Audiences), rely heavily on it. So let's first look at the profile and then zoom in on the other features that the profile makes possible.

7.1 The SharePoint User Profile

From a conceptual point of view, the user profile is somewhat similar to a standard SharePoint list that is globally accessible. Indeed, it is a shared service and can therefore be used by any site in any farm using the SSP. As with all lists, it is extensible in the columns it contains, it supports multi-valued fields, and the items in it can be indexed. But given that it is specifically designed for user information, it has some other features that normal lists don't have, such as the ability to populate certain columns from external data sources.

A common misconception is that the user profile is required and/or involved when authorizing access to a SharePoint site. This is not the case, and there is no requirement to implement the profile at all. When you grant access for a user to a SharePoint site, the user is validated against the underlying authentication provider you are using—by default, this is the AD and the profile is not involved at all for this operation. Indeed the user details for each site collection are stored in a table called UserInfo, and this information is modifiable by site collection administrators. End users can also see and modify their own site details by clicking on the My Settings option, which appears under the Welcome menu at the top right corner of a SharePoint site page (as shown in Figure 7.1). But this information about a user is minimal, is scoped to a site collection, and is not kept synchronized with the original user object. Therefore, this information is not that useful and can very quickly become stale.

If you do implement the profile, far richer information about people can be displayed when navigating to a person's details and, furthermore, it can also be kept up to date with the underlying directory service, which makes it far more appealing.

7.1.1 Populating the User Profile

So how does information get into the user profile? There are various options, from manually adding entries, to having entries dynamically created, to writing your own import, to importing from external data sources. The last of these is the one most likely to be used by the vast majority of people, and the most common external data source will be some form of directory service. Why? The user profile is very much like SharePoint's own people directory, and therefore the data it contains is very similar to that of any directory service.

It should come as no surprise that user profiles can be imported from the AD. This ability was there in SPS 2003, but now we also have the option of importing from any LDAP directory, or even through the BDC. The latter

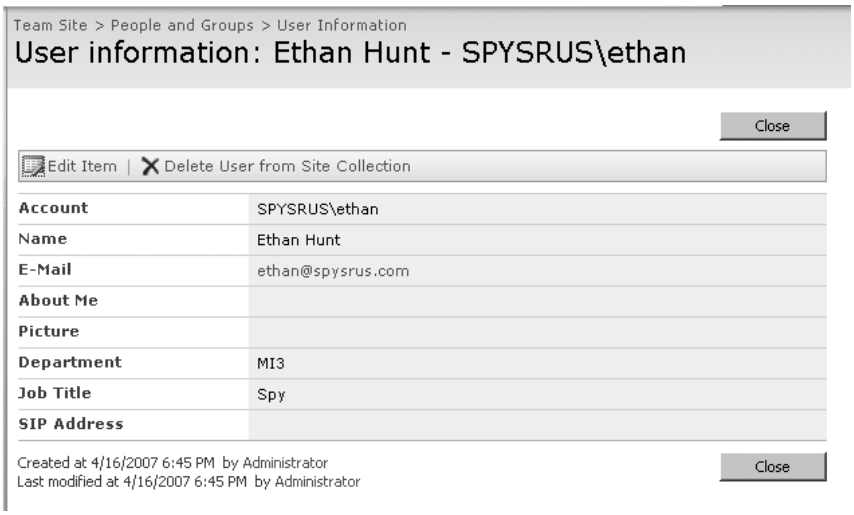


Figure 7.1
User settings from a WSS site

can be very useful if your master data source for people is perhaps controlled by a human resources–type business application, such as Seibel or PeopleSoft.

An SSP administrator will have to first configure the import, which involves specifying the import source and also setting up connections to the external data sources. Although the import itself can only come from one external source, you can set up individual profile properties that can be populated from different sources, assuming you have a property in the profile that can be used as a key into the external source. Setting up an AD source is the most straightforward, as it will look at the current forest that the SharePoint server itself is running in and automatically discover a suitable domain controller from which to import profiles. You can choose to import objects from the current domain or from the entire forest, should you be running in a multidomain forest. You also specify a schedule for a full and incremental import.

You then need to configure an import connection for the source. We can see the default connection created for importing from the current active directory domain in Figure 7.2.

There are three sections for configuring the connection: connections, search, and authentication. The connections settings allow you to specify where to connect to in terms of domain controller, port, SSL, and time out values, and the authentication settings allow you to specify which account to use to access the AD. This account must have read access to the objects that you want to import.

The search settings section allows you to formulate the LDAP query that will eventually be executed against the AD. This grants you granular control over where to start your search (search base), how deep to go (scope),

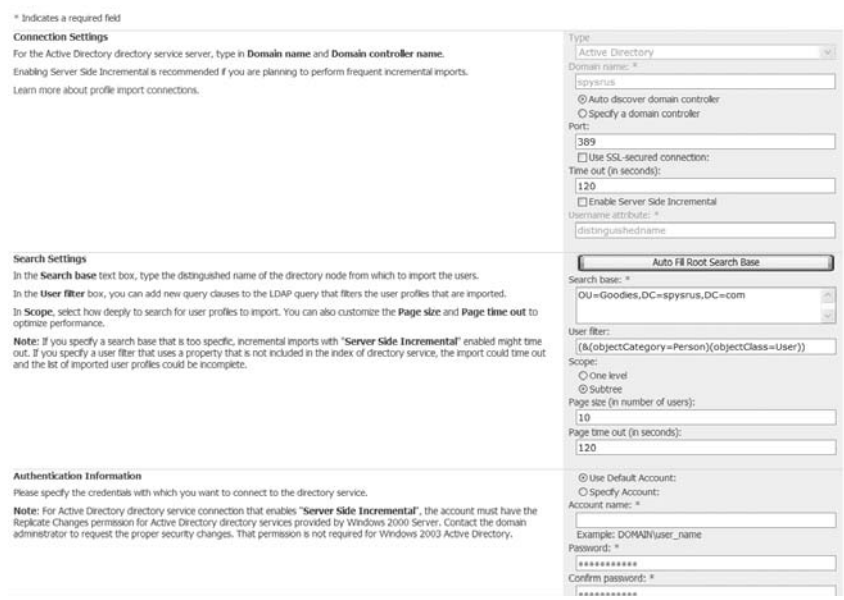


Figure 7.2
AD connection settings.

and the number of objects to return in a paged query (page size). Thus you can, for example, choose to only import user objects from a particular named organizational unit within your AD. It is important to realize that you need to be aware of how your AD is structured, so you will have to work with your directory people to ensure you pull in the right data for your needs.

The example show, in Figure 7.2 would execute an LDAP query as follows:

```
Starting node: OU=Goodies,DC=spysrus,DC=com
Filter:  ( &
(objectCategory=CN=Person,CN=Schema,CN=Configuration,DC=spysr
us,DC=com)  (objectClass=user) )
Search scope: subtree
```

This query would return all the objects with an objectClass of user, and then the import would enumerate each one and return various attributes, as we will see shortly.

Note that you have the option of specifying Server Side Incremental on the connection settings. This specification comes into play when an incremental import is being performed. If this is unchecked, the LDAP query effectively runs again and each object is checked to see if its modified date has changed. In large directories, this could take some time. Server Side Incremental means the LDAP query detects changes that have been made to

the AD which can make the incremental update perform much more quickly. You need to check with your AD as to whether this is supported in your environment, as it might require additional permissions to be granted to the access account.

7.1.2 User Profile Properties

When you view the user profile properties from the SSP admin, you will note that multiple types of properties are contained within sections. Furthermore certain properties are "mapped" properties, which means they take their data from an external source. If you change any of these properties from within SharePoint, they will be overwritten the next time the external source is imported.

By default, the following attributes from AD objects are returned from the LDAP query during an import:

```
displayName, givenName, sAMAccountName, wWWHomePage, mail,
proxyAddresses, telephoneNumber, distinguishedName, memberOf,
objectSid, physicalDeliveryOfficeName, department,
objectGUID, title, manager, sn
```

Most of these are written into the user profile. You can see some of these properties in the Mapped Attribute column in Figure 7.3, which is showing us all the user profile properties that are currently defined.

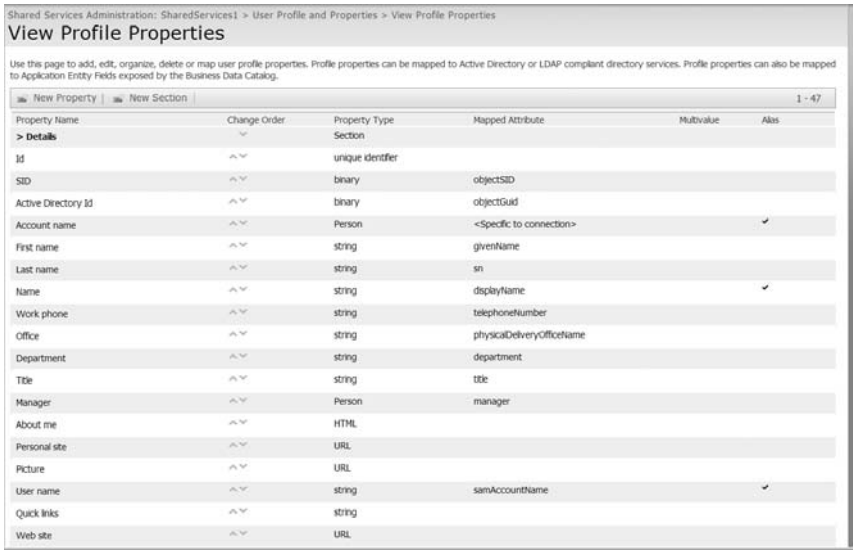


Figure 7.3 *User profile properties.*

The fact that the objectGuid is stored in the user profile ensures we have a unique way of matching the profile entry to the AD object. You will also note that the objectSID is stored as well, which gives us another way of finding the underlying object.

If you looked closely at the list of default attributes that are returned in the LDAP query you would have noticed the memberOf attribute. This attribute tells us which AD groups a user is a member of, and SharePoint keeps track of this information in a separate table in the SSP database. This information is subsequently used to show users what they have in common with other users. We will discuss this a bit more when we talk about the public view of a user's My Site.

You can modify the out-of-the-box properties and create your own as your needs dictate, but note that once a property is created, you can only modify its displayname. Therefore if you make a mistake or change your mind, you will need to delete and recreate the property. There are some important considerations to make when creating a property since they can affect many features down the line. Let's look at some of these settings now:

- ***Multilingual Considerations.*** User profile properties surface in many places and can be edited by end users should you so desire. Thus, you can associate language-specific text with the display name and description of a property, and this will be shown to users who are running SharePoint with the appropriate language pack enabled.
- ***Policy Settings.*** These settings allow you to apply a privacy policy to a property. User properties surface in many places—in search results and public views of a user's My Site, for example—and there may well be restrictions on who is allowed to see what properties. Policy settings are akin to Access Control Entries for individual properties, allowing you to specify who is allowed to see a property and whether end users are allowed to change the policy. There are five different scopes for privacy settings, and each one increases the number of people that can see the value of the property: Only Me, My Manager, My Workgroup, My Colleagues, and Everyone.
- ***Replicable.*** If a property is marked as replicable, then it is replicated into the user settings on individual SharePoint sites, thus ensuring that the same data is seen consistently across all sites. This overcomes the issue we mentioned earlier with site user information (held in the UserInfo table) being scoped to the site collection level. We can see by comparing Figures 7.1 and 7.4 what ultimately happens with site user information and replicable properties. The user details from the site have been matched with the details from the profile using the objectSID as the key. Thus, a single change in the AD permeates

across all SharePoint repositories. Replicable properties must have a privacy scope of Everyone. Properties that require replication are processed hourly by the profile synchronization job executed by the SharePoint Timer process. As well as updating properties on the user information at the site collection level, it also adds any new replicable properties to the user information list.

- ***Display and Edit Settings.*** These control whether the end user is allowed to edit this property's value and whether it is displayed on the user's public view of their My Site (and privacy settings also come into play here, so some users may see the property and others not). You can also indicate here whether the property is shown when its value changes in the colleague tracker Web Part (discussed later).
- ***Search Settings.*** Here you can indicate whether a property value is held in the full text index and therefore will be used to match query terms to people. You can also indicate whether a property is an alias, which means it will be treated as an equivalent to the user name and account name when searching for items authored by a user.
- ***Property Import Mapping.*** This is the section that allows you to map a property to something in an external data source—be it an attribute from the AD or from an LDAP directory or something that is described in the BDC. As an example, let's add the name and age from our Actor table that we used earlier as properties in the user profile. Before we can do the mapping, we need to define a connection to our BDC using a current user profile property as a lookup value for the entity in the BDC. In this case, we will create a new property (ActorId) of type Big Integer and use that as our key when defining the connection. When that is done, we can now choose the BDC connection when mapping to new properties. The net result is shown in Figure 7.5, in which we are viewing James Bond's profile after associating the primary ActorId of 1 with his profile.

7.1.3 Searching the Profile

We talked in Chapter 6, Search, about how you can display search results in an appropriate way for the type of search results you are seeking. Out of the box, you will find a Search Scope, Search Tab, and separate Search Results landing page for People. If you take a look at the Search Scope, you will see that it has a rule on it that matches those items in the index that have a content class of urn:content-class:SPSPeople. This is the content class that gets stamped on entries from the user profile as they are indexed via the Local Office SharePoint Server sites content source. Getting details from the profile to contribute to search results is a three stage process: define the property, populate the property, and index the property—so remember that you always

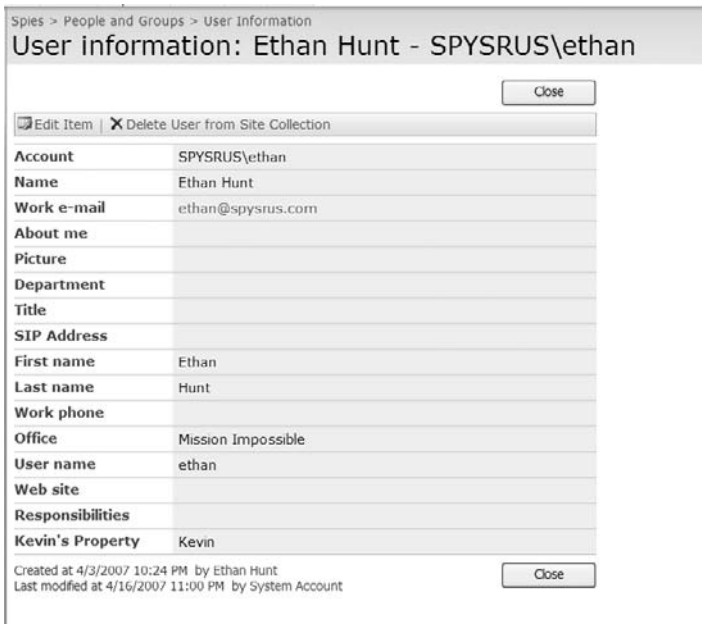


Figure 7.4
Property replication to site collection.

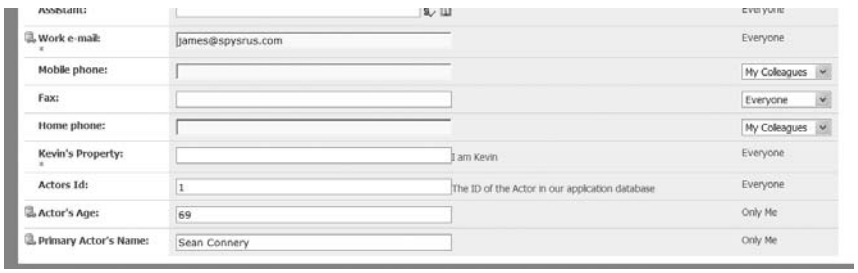


Figure 7.5
Mapping a property to the BDC.

have to have the search engine come along to index the profile after you have populated it! The peopleresults.aspx Web Part page has a core results Web Part on it that differs from the standard one in the columns that it returns and the XSL used to display the results. For example, it returns columns that are more applicable to people, such as skills and responsibilities and the picture associated with the person. The net result is that you can search for expertise instead of just searching for people by their names.

The landing page is called peopleresults.aspx and contains special Web Parts for displaying information from the user profile pertinent to the person doing the search. What do we mean by pertinent? The idea is to deliver the search results sorted by social distance to the user. By social distance, we mean how closely you know the people being returned in the result set. Your colleagues are listed first, followed by your colleagues' colleagues and then

everyone else. Let's assume that your query terms are searching for a particular expertise. All people in the profile that match this expertise will be returned, but your colleagues will be listed first. The reasoning is that you would normally seek out people you know first before casting your net more widely. Furthermore, if you do need to cast your net more widely, you would normally try and do this through someone that knows the person you actually need to talk to, rather than approaching them directly. Thus seeing the people whom your own direct colleagues know helps in this regard. In effect, the search results show you the social networks that exist in your organization, in an attempt to help you navigate and exploit them.

But how are these networks and the relationships determined? For a standard MOSS installation each user has a list of colleagues that can be maintained manually or semiautomatically. The latter means that the system will suggest people that you might consider your colleagues, and you can then accept or reject these suggestions. We will see later how these recommendations are determined but, as you can see, the social networks are built from the people that you know. There is an add-on product that was due to be released at the same time as MOSS called the Knowledge Network that extends the social networking capability, enabling you to find out who knows what rather than just who knows who. It does this by analyzing your e-mail patterns, e-mail contacts, and IM contacts to determine whom you communicate with internally, and externally, and the types of things you are discussing. From this information it builds an extended profile that hooks into the the index and then displays the results via an extended search landing page—for example, a separate tab for External People is added. At time of writing, the Knowledge Network is available as a technology preview, and more information can be found at http://blogs.msdn.com/kn/.

Turning back to the standard SharePoint capability, we can see an example of a search for expertise in Figure 7.6. Here, John Steed is searching for people that know something about cars, and the result set gives him no sense of any social connections since it is just a flat list. In Figure 7.7, we see the same result after we have set up some relationships in terms of colleagues. This is a much better results set since, for example, if Steed was particularly interested in Russian cars then he could ask Emma Peel, his direct colleague and one social "click" distance away from him, for an introduction to Illya Kuryakin who is currently two social clicks away since he is a direct colleague of Emma's.

Figure 7.6 *Flat people result set.*

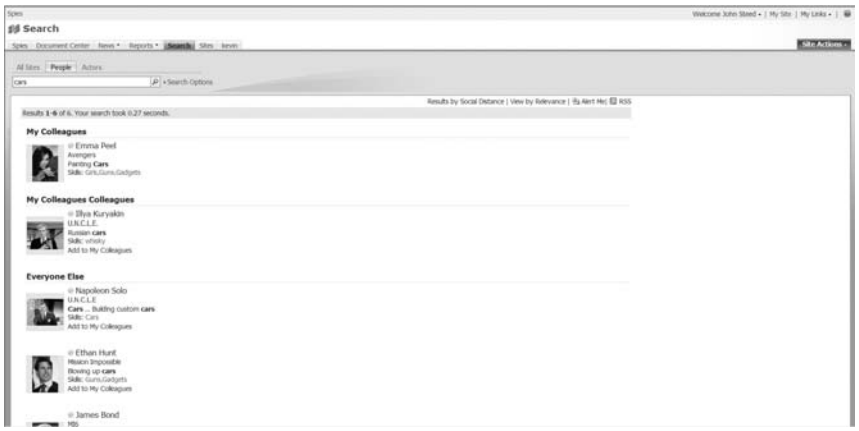


Figure 7.7 *Social structure to people result set.*

Now that we have discussed how to load up the profile and have it contribute to relevant search results, we will move on and discuss what the people features in SharePoint bring to the individual.

7.2 Personal Portal—A User's View

If SharePoint is to be the "place where people go," then it must offer support that enables the individual to process his or her own information with as much ease as they do other SharePoint information. From a personal portal point of view, there are three main areas to consider: By Me, About Me, and For Me. Users need their own personal place where they can work on private things and have the freedom to shape their own view into various information repositories, much like a My Yahoo or My MSN. This is the By Me area, and it is implemented via something called a My Site.

Next, users need to be able to make information about themselves available to others such that others can benefit. This is the About Me area, and it is implemented via a public view known as My Profile.

Lastly, there needs to be a way in which other people can push relevant information to the users. This is the For Me area, and it is mainly implemented through Audiences. Let's now take a closer look at these three areas and some key features that go along with them.

7.2.1 By Me—My Site

A My Site is a SharePoint site that is instantiated using the SPSPERS template. The site is created dynamically when suitably permissioned users click the My Site link on the Welcome navigation bar at the top right corner of a site. The site can also be pre-provisioned programmatically. When the site is instantiated, the URL for it is placed into the Personal Site property within the user profile.

My Site is a shared service, and therefore it is shared by all the Web applications that use the same SSP. What's more, a shared service provider is in itself a Web application that has its own set of permissions associated with it, and through these permissions you control who is and isn't allowed to see the My Site and My Links options. You can see this using the My Site Settings link from SSP administration. From here, there are three main settings pages that you can access from the left-hand quick launch bar. For controlling the visibility of My Site, you need to select the Personalization Services Permissions option. There are two rights related to My Site—Create Personal Site and Use Personal Features—and you will note that an out-of-the-box deployment grants these rights to Authenticated Users.

Use Personal Features enables the My Links option, and Create Personal Site enables the My Site option. Use Personal Features is a prerequisite for being able to create a My Site. The subtle difference between the two is that Use Personal Features lets you participate in all the personalization capabilities that SharePoint has to offer (such as showing you your colleagues and memberships), and Create My Site adds personal storage in terms of lists and libraries on top.

By default, there are two parts to your My Site: My Home and My Profile. The former is your actual personal site, and the latter is where you maintain your About Me details, which we discuss shortly. A default personal site can be seen in Figure 7.8. It is a regular SharePoint site collection with the owning user specified as having Full Control, but note that other permissions can be added automatically when the site is created. This is under the control of the SSP administrator (My Site Settings/My Site Reader Permissions). By default, the Authenticated Users group has read access in order to allow people to view the public profile. Note this does not give access to personal con-

tent, such as Personal Documents, since it has a unique permissions set. Also, standard security and rights trimming is effective in a personal site so that the owner is in complete control over who is allowed to see what.

Figure 7.8 *Default personal site.*

Given that the personal site is just a standard team site, the user can freely create more lists, libraries, and pages and exploit all the inbuilt SharePoint features. Search is enabled on the personal site, and SSP administrators can set the search results landing page that will be used. Just a warning here—search scopes are defaulted to All Sites and People, so if you want other scopes to be in the drop-down, you have to enable them through Site Actions/Search Settings within the personal site itself. Also, if you allow, users can choose the language for their My Site, which helps in multilingual organizations.

But there are some other interesting features and Web Parts that are specifically designed for a personal site.

First, there is the option to associate the personal site as your default Office My Site location—indeed, you are given this option when you visit your site for the first time. This does two things: it synchronizes your document libraries into Outlook 2007 and adds your My Site as a Save As location for Office applications. We will discuss the mechanics of this in Chapter 9.

Blogs are a good way of communicating your thoughts to anyone who wants to listen, and there is an option to have a personal blog created for you. Of course you could just do this through the normal clicks, but having it there as an option on your home page encourages more people to actually initiate it. Your blog is created as a subsite of your personal site collection. If you have created a blog other people will get to see your most recent postings when they visit your public profile.

My Links is a place where you can store your favorite sites for ease of navigation. As you navigate other SharePoint sites, you can add their URLs to your My Links and then group them according to your needs. You can add any type of link to your list, such as common searches and links to external information. You can group your links and also specify which links other people are allowed to see when they view your public profile.

7.2.2 SharePoint Sites Web Part

MOSS delivers some Web Parts that aggregate information from multiple locations. One such Web Part is My SharePoint Sites, and it appears by default on your personal site home page. The idea behind this Web Part is that most users will be members of many different sites but perhaps only actively participate in a handful at any one time. Therefore this Web Part surfaces the important content from the sites that you are currently interested in without you having to navigate to and from the various sites. More correctly, it surfaces any documents and tasks that are assigned to you from the target sites and offers you a quick way of accessing these items. During development of MOSS it was thought that this content aggregation of documents and tasks would be scoped to a site collection, but alas this is not the case and you only see the documents and tasks associated with the actual site. Pity that, since content roll-up across a site collection is something that most users are crying out for, and it would have been nice to see it in the core Web Parts that ship with the product. That said, the content by query Web Part can be used to aggregate content from anywhere within a site collection, so you could use it instead to deliver the required functionality.

The sites that you are currently interested in appear in a menu bar, which you can see in the My Site displayed in Figure 7.8. You can manually add sites to this bar and also choose to hide them from the bar if they are not currently foremost in your thoughts. Manually adding sites here is all very well, but sites where your user account has explicitly been added to the SharePoint group that represents the *members* of the site will appear automatically (you can specify which group holds the members, visitors, and owners for a site). Indeed, the same Profile Synchronization background task that replicates properties also keeps track of the members of individual sites, with the details ultimately being stored in various tables in the SSP database. This task runs hourly. A user must have an entry in the user profile to be linked to sites of which they are a member. The list of sites that you are determined to be a member of also surfaces in My Links in a special grouping, making it simple to link to different sites regardless of which site you might currently be visiting.

Another Web Part called simply SharePoint Sites gives you the same functionality as above, except it does not automatically populate the membership.

7.2.3 Colleague Tracker Web Part

SharePoint tries to build a picture of social networks and present the linkages to you when you browse people information. But it can only do this if it has some information about the working relationships between people, in terms of who their colleagues are. There are a few ways in which your list of colleagues can be built up and maintained, from manually adding contacts, to adding contacts from a search, to accepting suggestions from the system.

Colleagues for all users in all farms that share the same SSP are actually stored in a central list and not in a list in each user's personal site, as you might at first think. Remember that you do not need to actually have a My Site to use the personal features that SharePoint has on offer and this is one reason why the details are stored centrally. They are actually stored in a table called UserColleagues in the SSP database.

The colleague tracker Web Part (shown in Figure 7.8) lets you view your current colleagues, add new ones manually, and accept system suggested ones. We can see in Figure 7.9 how system-suggested colleagues are shown to you when you begin to add some colleagues. Although the help text suggests that the colleagues suggested to you are based upon your e-mail and IM patterns, this is only the case if the Knowledge Network is in the picture. MOSS itself does not look at your e-mail or IM details in any way! Rather, it looks at group membership in terms of SharePoint sites and AD groups to determine who might be likely colleagues of yours (recall that during a profile import, the AD groups that a user is a member of are analyzed). Ultimately, it is up to you, the end user, to opt in and accept these suggestions and you can always tell the system to stop suggesting various people to you if you think they are not relevant!

Privacy options come into play when other people are viewing your public profile. As with user profile properties, you can indicate who is allowed to view your colleagues using the scopes of only me, my manager, my workgroup, my colleagues, and everyone. Your workgroup contains a subset of your colleagues of your choosing and allows for finer granularity in determining who is allowed to see what.

The colleague tracker also shows you changes that have occurred to information about or by your colleagues. For example, it can tell you when one of your colleagues has added a new blog post or document, or when one of their user profile properties have been changed. This helps you keep up to date with what is going on with other people in your social network.

7.2.4 About Me—My Profile

All My Sites are actually linked to a central SharePoint site, and personalization items like colleagues, links, and memberships are accessed from this site.

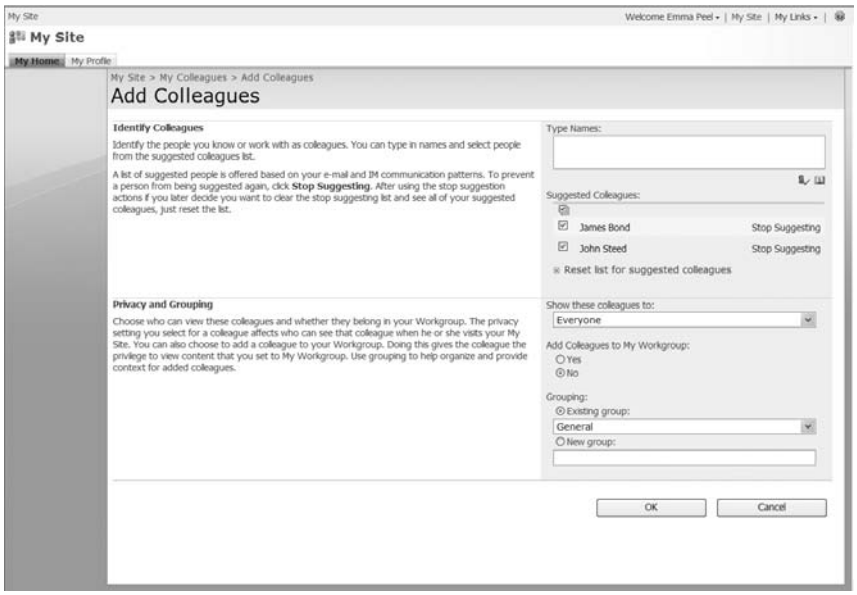


Figure 7.9
Adding colleagues.

The My Site Host is provisioned when you create the SSP from the site template called SPSMSITEHOST. The description of this directly from its template is "A site used for hosting personal sites (My Sites) and the public People Profile page. This template needs to be provisioned only once per Shared Service Provider, please consult the documentation for details."

The public People Profile page is what is shown to users that link to your name, and it controls what other users are allowed to see "About Me." The page itself is called person.aspx and its contents are determined by the actual person who is looking at the page and who the page is about. Person.aspx takes varying forms of query parameters to drive who the page is about—for example it can take Guids or account names—but ultimately, it will match an entry from the user profile.

Once this is done, it can then determine the relationship the requesting user has to the user whose details they are trying to view. This will categorize the requesting user as either being the actual user, the user's manager, in the user's workgroup, a colleague of the user, or none of the above. Incidentally, the user's manager is determined from the manager property in the user profile, which is mapped to the manager attribute in the AD.

This determination is important, since it controls who is allowed to see what on the various Web Parts that are on the About Me page. We have already seen how properties and colleagues can be in different privacy scopes, and there are other items on the About Me page that behave similarly. If you are accessing your own page, you are given the option to toggle the contents

of the page so that you can see what other people will see for each of the determined scopes. This is clearly very useful if you are concerned that you might be releasing information to people with whom you did not intend to share it.

Unlike standard Web Part pages, which have both a shared and personal view, the person.aspx only has a shared view and therefore cannot be personalized by end users in terms of adding and removing Web Parts. Only users who are granted suitable permissions on the My Site Host site can modify the shared view of the page and, when they do, they are modifying it for all users. Also, note that not all the information on the page is delivered via Web Parts, and you therefore may have to use other editing techniques, such as SharePoint Designer, if you want to radically alter the contents of the page.

Figure 7.10 shows the default public My Profile page. In this example, we are viewing the public profile of Emma Peel from the perspective of James Bond as the logged-on user. The details, membership, links, and colleagues that are displayed are all dictated by the privacy scope the person viewing the page is deemed to be in. The organization hierarchy is determined by looking at the manager property from the user profile, thus determining who reports to whom. Note the In Common With Web Part at the top right. This shows the relationships you share with the user whose details you are currently viewing.

Figure 7.10
Public profile view.

The team sites that are displayed on the documents Web Part come from those in the user's Membership list that you are allowed to view. By clicking on each team site, you can then see the documents that have been authored by the person you are viewing in those team sites.

7.2.5 For Me—Audiences and Personalization Sites

The last part of the people jigsaw is the For Me piece which targets relevant content to people through some editorial process. The major types of content that can be targeted at specific users are personalization site links, Web Parts, and list items.

Personalization site links appear in the navigation bar on a user's My Site. As we have seen, this normally contains two links, one to the personal site and one to the public profile but you can add more links dynamically by creating them through SSP administration. Furthermore, you can subsequently choose to only display the links for certain audiences thus allowing you to target personalized links. As an example, you may want to provide a link directly to the user's divisional portal. Although a link to any type of site can be added, the design of personalization site links is geared towards linking to actual personalization sites (thus their name!). What does this mean? The user context is passed in a personalization site link and if you therefore have user-aware Web Parts on the target site, the content can be specific to the user. Thus, you could have links to My HR, My Sales, and so forth where each user gets the HR or Sales information specific to them.

A *personalization site template*, called SPSMSITE, is shipped with MOSS with the following description: "A site for delivering personalized views, data, and navigation from this site collection into My Site. It includes personalization-specific Web Parts and navigation that is optimized for My Site sites." The magic here is that there are a couple of filter Web Parts precreated on the home page that allow you to pass details about the calling user to other Web Parts. These target Web Parts then filter the content for a personalized display. The success of personalization sites relies heavily on understanding the data that you want to show, knowing where it is going to come from (the BDC perhaps), and having Web Parts that understand and effectively use the filters that are passed to them.

As mentioned, you can use personalization site links to place links to other sites in the navigation bar that appears in a user's My Site. This same navigation bar is also used when you navigate to any site that has been instantiated from the personalization site template. The My Site navigation bar is different from that of other sites, and you may want to do all of your work from within your My Site space rather than linking off to different contexts. You can achieve this by using the ?MySiteView=1 parameter on the URL for the target site (and, incidentally, this parameter is automatically added to any personalization site links the SSP administrator may have set up for you). This subsequently allows the user to pin the link so that navigating to it stays with the My Site chrome. A good use of this is to provide a link to the main portal, as shown in Figure 7.11. This is a personalization site link that has been pinned to the My Site navigation bar.

Figure 7.11
Pinning navigation links to My Site.

Although personalization sites can be used to filter information based on the calling individual, *audiences* can achieve a similar result for rule-based groups of people. Once you have created and compiled audiences, you can then target personalization site links, Web Parts, and list items to users that are part of a particular audience. Audiences are a shared service and are therefore managed via SSP administration. Each audience consists of one or more rules, and you can configure whether one or all of the rules have to be met before a person is considered a part of the audience.

Each rule has an operand, an operator, and a value to check the result against. The operator can be either a user or a property from the user profile. If it is a user, then the operator is one of Reports Under or Member Of. The Reports Under matches against the manager property in the user profile. The Member Of matches against any group that is defined—be it an AD group that has been imported via the user profile, or the members group of individual team sites as described earlier. A property operand comes from any property defined in the user profile (even BDC properties), and the operator is determined by the type of the property, with all common operators being available.

Audiences must be compiled before they can be used, and once compilation has succeeded, you can view the members of the audience to verify that it is picking up the people you expect it to.

There are a couple of ways in which audiences can be put to good use when it comes to Web Parts. First you can control the actual Web Parts that appear on a page by setting the audiences in the advanced settings on the Web Part itself. Secondly, you can filter the contents that Web Parts display to therefore show different items from a list or library to different audiences. To achieve this, you have to first enable audience targeting on the list itself via the settings page. Doing so enables an audience targeting column in the

list. You then edit the properties of the items within the list that you want to target and set the appropriate audiences. Finally, you configure a Web Part that is audience aware and tell it to use the audiences that are set on the items as a filter for display purposes based on the user context. The content query Web Part is audience aware, and you can set its input to be any list or library in the site collection, so this can be put to very good use. For example, you may have a centralized list of help desk numbers but want to ensure that each number services a particular subset of your user population. A simple list whose contents are filtered by audiences could be used to achieve this.

7.3 People Picker and Cross Forest Operations

To close this chapter on people, let's take a quick look at the people picker. This is the control you will come across often as you add users to your sites, add colleagues, and so on. It allows you to check names and browse the directory service as well as the users currently within the site collection. One of the problems in SPS2003 was that the people picker could only browse the directory service in which the SharePoint Servers themselves were installed. Many customers implement a multiforest scenario with the user accounts in different forests that of SharePoint resources, with trusts set up to support interoperability between the forests. These scenarios were not supported in SPS 2003, but they are in SharePoint 2007, from both the people picker perspective and My Site, personalization, and so forth. For the latter case, SharePoint treats the user as having one identity, even though he or she may have multiple login accounts in different forests.

When you access the directory service through the people picker, LDAP queries are executed to retrieve and validate information about users and groups. For example, if we check names after entering the letters "BO" you can see the result in Figure 7.12. Why does "Aaron Asjar" appear in the result set? The actual executed LDAP query searches the whole forest using the following filter:

```
( |  ( &
(objectCategory=CN=Person,CN=Schema,CN=Configuration,DC=spysr
us,DC=com)  (objectClass=user) ( !  (userAccountControl&2) )
( |  (name=bo*)  (displayName=bo*)  (cn=bo*)  (mail=bo*)
(sn=bo*)  (sAMAccountName=bo*)  (proxyAddresses=SMTP:bo)
(proxyAddresses=sip:bo) ) )  ( &
(objectCategory=CN=Group,CN=Schema,CN=Configuration,DC=spysru
s,DC=com)  (groupType&-2147483648)  ( |  (name=bo*)
(displayName=bo*)  (cn=bo*)  (sAMAccountName=bo*) ) ) )
```

Thus, it attempts to find something that matches based on quite a few criteria. In our example, Aaron's email address was Bologne@spysrus.com, which is why he was selected.

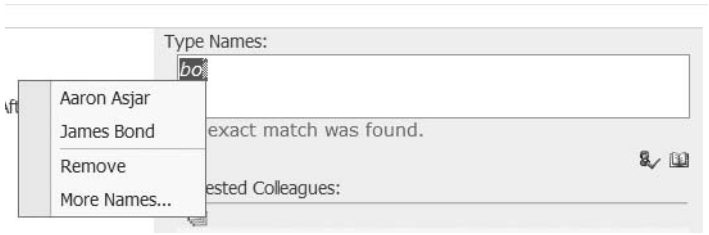


Figure 7.12
People picker.

You can somewhat control the LDAP query using stsadm, and this is also how you let the picker know that it can go cross-forest to resolve a name someone might type in. To set up cross-forest resolution, you need to set the property name called peoplepicker-searchadforests, whose value contains the names of the remote forests and a username and password for accessing them. For example:

```
stsadm.exe -o setproperty -url http://server:port -pn
"peoplepicker-searchadforests" -pv
"forest:foo.corp.com;domain:bar.foo.corp.com", LoginName,
P@ssword
```

You can also augment the LDAP query with your own custom query, which could allow you to only return objects that have been identified in some way as being applicable to SharePoint use. For example, you may set a custom attribute on such objects through your provisioning process. To set a custom query, you need to use stsadm and set the peoplepicker-searchadcustomquery property to a value that contains the custom part of the query. If this property is set to be (extensionAttribute1={0}*), then whatever is typed in the people picker would also be matched against the value in the extensionAttribute1 of the AD objects.

8

Business Intelligence

In the opening chapter of this book we talked about the core solution areas that MOSS addresses, one of which is Business Intelligence (BI). Business intelligence is all about making smarter use of your business information, and there are three core products in the Microsoft space that enable this: Microsoft SQL Server 2005 platform, Microsoft Office SharePoint Server 2007, and Microsoft Office Business Scorecard Manager 2005. There are also a couple of complementary products in this space: namely ProClarity Analytics 6 and Microsoft Office PerformancePoint Server 2007 (which builds on ScoreCard Manager). Within SQL Server, you have powerful analytics via the Analysis Services, which help businesses understand and analyze large volumes of rapidly changing data. The Business Scorecard Manager is typically used by C-level executives who need to track overall performance against company goals.

What we are about to discuss in this chapter are the BI features that SharePoint enables. We like to call this BI for the masses, since it really empowers teams of users to share and analyze any kind of business data without actually targeting a specific scenario (as, for example, ScoreCard Manager does). What is the primary target for the source of such business data? Excel. Yes, Excel—but not as you once knew it! As we will see, SharePoint addresses one of the primary issues with Excel—that being that the data contained within a workbook is seldom shared or reused outside of the confines of Excel. However, there is a wealth of business logic and intellectual property contained in Excel workbooks that should be readily consumable by others on your team or even by a wider audience. And although we said Excel is the primary target for the source of business data, remember that Excel itself can consume data from SQL Server and other databases, so targeting Excel really means targeting all the data sources that Excel can get at. So let's discuss something called Excel Services and the other BI features that SharePoint brings to the masses, such as key performance indicators (KPIs), dashboards, and reporting libraries.

8.1 Excel Services

Excel Services is a server technology that ships in SharePoint Server 2007 and enables the sharing, controlling, and reusing of Excel 2007 workbooks on centrally managed servers. Server-based workbooks are accessible via browsers and therefore consumers of these workbooks do not even need to have Excel 2007 installed locally. Note that the authoring of workbooks does require the Excel 2007 client, but the subsequent reuse of them does not.

Workbooks can be stored in UNC file shares, HTTP servers, or SharePoint libraries, and in the last case they benefit from all the inbuilt goodness of SharePoint, such as security, auditing, workflow, versioning, and so on. Once workbooks are stored on the server, the data within them can be exposed through multiple vehicles; for example, a single cell from within a workbook can be used as the data to measure a key performance indicator against, or a chart can be displayed within a dashboard page on a SharePoint site. This is where the real power of Excel Services comes into play: unlocking and unleashing business logic and business data so that others can use it to make informed business decisions in a controlled and safe environment.

Sharing Excel workbooks today usually involves sending out multiple copies attached to e-mail or placing a copy on a central file server. Each interested party then takes a private copy of the workbook and uses their local computing power to analyze the data—but in isolation from the other folks. Not only do you multiply the actual computing power required, you lose control over who has what and end up with multiple versions of the "truth." With Excel Services, a single calculation engine on the server computes all the functions and prepares the workbook for consumption. Each user then has his or her own view into the workbook for the purpose at hand. This is a much better way of ensuring that you have a single version of the truth, and makes it far more efficient use of computing power by having long-running calculations run on the server, rather than on each individual's PC.

There are three core parts to Excel Services: a calculation engine that runs in tier 2 of a SharePoint architecture, Excel Web Access, which enables the browser based access to workbooks, and Excel Web Services which allows other applications to consume workbooks. The last of these is very powerful, as you can control your business logic using Excel but present the data in any shape or form you like. Consider a simple mortgage calculation engine in an Excel workbook. If the interest rate changes, all you need do is to change it in the source workbook on the server, and every application using that workbook will reflect the change. No longer do you need to code the business logic into each and every application.

Additionally, there are some great changes in Excel 2007 itself for those who author workbooks in the first place. For example, much larger workbooks can be created, with up to one million rows and sixteen thousand col-

umns of data. Sorting and filtering have been improved, as has the ease of using PivotTables and PivotViews. Conditional formatting allows you to generate rich visual content using heat maps, colored icons, and arrows. Through data connections, you can easily link to data and metadata in SQL Server Analysis Services extracting information from your OLAP cubes, thereby providing context for the data being analyzed.

8.1.1 Publishing Excel Workbooks

Publishing workbooks from the Excel 2007 client is pretty straightforward for authors. The major decisions they need to make are based on how much of the workbook they want to expose to other users, and there are three levels of granularity here: the whole workbook, individual worksheets, and named cells. Note that regardless of what you choose to publish, the whole workbook is stored on the server; the granularity is only imposed at consumption time. This makes sense, because all data in a workbook may be required to generate the result in a single named item. Note that the level of granularity is per workbook, not per user, and it applies when viewing the workbook through the browser. Whether a user can access the full workbook or not (by downloading it, for example) depends upon their permissions on the underlying SharePoint library to which the workbook has been published. If the user has at least read permissions, he or she will be able to take a copy of the workbook into Excel 2007 and analyze it to his or her heart's content. But if the user only has View Item access, all he or she can take is a snapshot of the workbook—and this snapshot only contains the data that the author indicated should be published. This is a nice example of the difference between the View Item right and the Read Item right.

Publishing is done using the Publish/Excel Services option from the Office button in Excel 2007. Doing so reveals the dialogs in Figure 8.1. The points to note from here are that the workbook itself is saved to a SharePoint library—any library will do, but as we will see later, a Reports Library is optimized for workbooks and dashboards. On the traditional Save dialog, the author is given two extra options: a check box that can be used to display the workbook in a browser once it is published, and a button labeled "Excel Services Options…" that controls how much of the workbook will be displayed in the browser—the entire workbook, individual sheets, or individual items such as the chart we are showing in our golf scores workbook. The former is a good way of checking that the workbook will actually display correctly before releasing it for public use, as not all Excel client features are supported in Excel Services (for example, protected sheets are not supported). The latter is useful for displaying results without revealing any of your formulas or raw data. For example, in our golf workbook, the individual worksheets contain each individuals score, and the summary worksheet contains the overall scoreboard. We

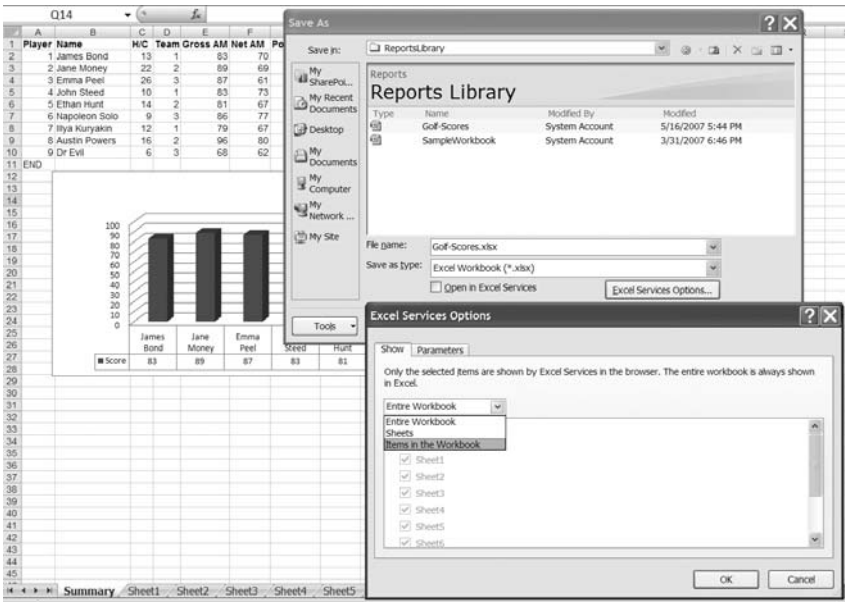


Figure 8.1
Publishing to Excel Web Services.

could choose to only publish the Summary worksheet, in which case no consumer can see the individual score cards via the browser.

8.1.2 Viewing Workbooks in the Browser

Even though authors can save workbooks to SharePoint libraries, the workbooks can only be viewed in a browser if a farm administrator has enabled the library as a trusted file location for workbooks and the Excel calculation service is running on the farm. So fear not—you have complete control over what workbooks can be rendered by Excel Services and also which data locations (for example which SQL database) those workbooks are allowed to access. Excel Services is a MOSS Shared Service, and you manage the trusted file and data locations from Shared Services Administration.

Once a workbook is in a library, you can view it using Excel Web Access and you'll find a very high-fidelity version of the source workbook. You can see this in Figure 8.2, which shows the browser version of the Workbook seen in Figure 8.1, as well as one of the individual sheets. Also note that Excel Web Access inserts menu options at the top left that allow you to take a snapshot of the data you are viewing into Excel or to launch the whole workbook (the latter option only appears if you have at least read access to the underlying workbook in the SharePoint library). Opening a workbook in Excel Web Access can be done by navigating to the SharePoint library that contains the workbook and using the View in Web Browser option from the drop-down

menu. There are no ActiveX control requirements with Excel Web Access—everything is done using DHTML (Dynamic HTML), and therefore the workbook should be displayable in many different browsers.

Figure 8.2
Viewing via Excel Web Access.

So we can easily view whole or parts of workbooks in a separate browser window, but the real power of Excel Services starts to be revealed when you display *portions* of workbooks within Web Parts. This is done using the Excel Web Access Web Part that ships with SharePoint Server 2007.

When you add this Web Part to a page, you customize it by specifying the URL to the workbook in a SharePoint library and by specifying the named items from workbook that you want displayed. Indeed, you can use the toolbar associated with this Web Part to browse through all the named items and display them at your leisure. You can see this in Figure 8.3, where I have chosen to only display the chart from our golf workbook into in a WSS team site. If an author updates the original workbook in the library the chart will also update upon refresh. This is a great example of how Excel Services allows you to unlock business intelligence and data that up until now has been very difficult to share and publish. Note that you can actually specify a URL to any workbook in any library in the same farm as the source workbook; however, if you use any of the browse capabilities when configuring the Web Part, it limits you to browsing the site collection into which you are adding the Web Part.

Figure 8.3
Excel Web Access Web Part.

8.1.3 Interacting with WorkBooks via Excel Web Access

So far, we have only shown how you can get read access to workbooks via Excel Web Access, but it is possible to allow the end user to filter and sort data as well as enter parameters that can be used to do "what-if" type analysis on the workbook. Excel filters are now type sensitive, so you can, for example, filter a table of numbers by values that are greater than or less than an entered value. Parameters can be used to control what is filtered in the various views within the workbook, and when we look at dashboards, we will see how other Web Parts can control the filtering. A good example of parameter use would be a mortgage calculation in which you let the enduser enter differing numbers of months to pay back the loan. In this case, the user is not modifying the source workbook at all; Excel Web Access is just feeding the data into a copy of the workbook that is being calculated on the server.

If you look back at Figure 8.1, you will notice a Parameters tab on the Excel Services Options dialog. This is where you can define cells whose values can be updated via the user entering parameters through Excel Web Access. Parameters must be named single cells that lend themselves well to things like percentages, quantities, and so on. For example, if we were to add a cell in our golf workbook to indicate the percentage of the user's handicap they were allocated, we would be able to do some what-if analysis on how that would affect net scores. We can see this in action in Figure 8.4—here we have republished the workbook, but only published the summary sheet. We have also enabled filtering and sorting on the scores table so the user can manually control what they want to see through the browser. In Figure 8.4 we have chosen to filter the view showing only those net scores that are less than 78 when players are only allocated half of their handicap. Oh, and remember—we are using a browser to display the workbook, not the Excel client!

The allocation named cell is actually in sheet1, but it is used in the calculation on the summary sheet. Therefore even though we have not pub-

lished sheet1, we can still control cells that live in it via parameters. Parameters are entered via a workpane that appears on the right-hand side of Excel Web Access (both the full browser and Web Part versions). You will note that we have also added the net scores to our chart.

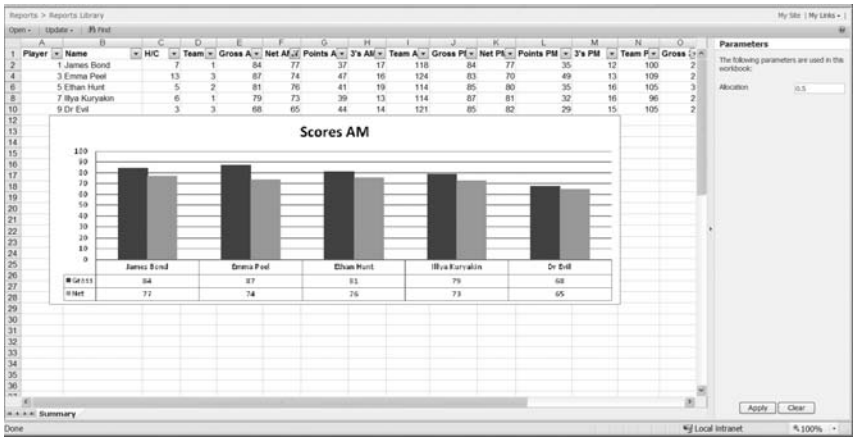


Figure 8.4
Parameters.

As you should hopefully now agree, Excel Services is a very powerful way of surfacing business data and the business logic that turns it into something useful. Business intelligence that has traditionally been locked away can now be used by the masses for many different purposes!

Let's now turn our attention to key performance indicators.

8.2 Key Performance Indicators (KPIs)

KPIs can be created in many applications, such as SQL Server and Business Scorecard Manager, and SharePoint Server 2007 also allows you to define KPIs and then display them in appropriate places. Again, this is all part of "BI for the masses," as the data that SharePoint KPIs use to make decisions can come from many different sources, even an individual cell in a workbook published via Excel Services. KPIs in SharePoint Server 2007 are very generic and flexible, and can be used for many different purposes. A KPI is generally used to give an aggregated view of data and to visually indicate how that data is performing against some goal. For example, you might want a KPI that warns you visually when the number of unanswered support calls goes over a certain threshold.

8.2.1 Creating SharePoint KPIs

The first step to creating a KPI is to create a KPI list in a SharePoint Server 2007 site. This is the list that will ultimately be used when displaying KPIs through Web Parts. This list supports multiple content types, and there are four types of KPIs that you can create. Each type gathers its data for calculating the performance of the indicator from different sources: data in a SharePoint list, manually entered data, data in an Excel workbook, and data in SQL Server 2005 Analysis Services.

Ultimately, each KPI is looking to gather a value, compare that value to some thresholds, and then display an icon (by default colored red, to yellow or green) to indicate the health of that value against those thresholds. Additionally, you can associate a details link with each KPI to display extra information, and this link could be to any aspx page you like. Thus, a very simple example with a manually entered KPI might be to display a red icon if the value is less than 50, yellow if it is between 50 and 75 and green if it is greater than 75.

Creating a KPI from data held in a SharePoint list gives you lots of flexibility in configuring what it is you are monitoring. For example, it could be as simple as the actual number of items in the list, the percentage of items in the list that have a particular column value greater than a particular number, or a calculation such as average, minimum, or maximum score. In our golf scenario, maybe we want to monitor the percentage of scores that are below par so we could do this with the net golf scores being entered into a custom SharePoint list. Although the UI does let you enter lists that are outside the current site collection, you will get an error when it tries to calculate the KPI; therefore, you are limited to gathering your data from a list within the current site collection.

Creating a KPI from an Excel workbook involves first specifying the URL to the workbook in your SharePoint library. You can then browse this workbook, and Excel Web Access allows you to navigate any of the sheets or named items that have been published from the workbook. From here, you can choose the cells that are to be used for the indicator value, indicator goal, and indicator warning. Thus you can use all the power of Excel to perform your calculations and then report on the performance of a particular aspect via a SharePoint KPI. Indeed, as a consumer of the Excel workbook, you needn't even concern yourself about the business logic behind the calculations—all you care about is the end result and how it is performing against certain goals. This is a very powerful feature, and its uses are effectively endless given that you have all the power of Excel to do your calculations.

Lastly, creating a KPI from data held in SQL Server 2005 Analysis Services involves specifying a data connection to the SQL Server. Once this is

done, any KPIs that are defined in SQL are surfaced and their values are used to determine how the SharePoint KPI is performing.

8.2.2 Displaying KPIs

You can navigate to your KPI list and view the items directly from there, but it is more likely that you want to surface individual KPIs in Web Part pages, along with other information pertinent to the task at hand. This is easily done using one of the two KPI Web Parts. The first will show you all of the KPIs in a list and allow you to choose different icons, such as traffic lights or checkmarks. You can also filter the list to only show KPIs that are deemed as problems. The second Web Part allows you to focus in on one KPI—this is the Web Part that would generally be used to put a big red or green icon on a page to focus the attention of the consumer, such as "your sales are down for the month!" You can see both types of Web Parts in use in Figure 8.5.

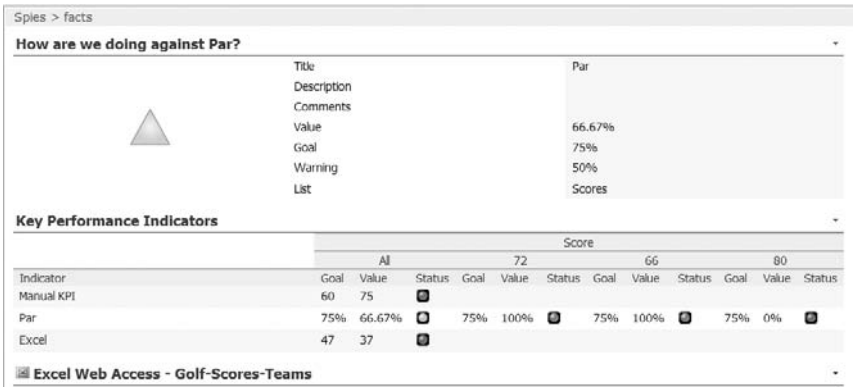


Figure 8.5
KPIs in action.

Now let's look at how we can use dashboards and reports to pull together pertinent information from multiple sources, ultimately displaying relevant information for the current user.

8.3 Dashboards and Report Center Templates

Dashboards are really just Web Part pages that are optimized for viewing filtered business data that resides in workbooks, KPIs, SQL Server 2005 Analysis Services, and so on.

SharePoint Server 2007 ships with a reports site template that is optimized for connecting to business data and displaying it in dashboards. It is also optimized to store dashboards and workbooks as reports, enabling features such as a report history so that you can look at trends. It contains predefined dashboards and supporting libraries, such as a data connection

library. The data connection library can be used to house definitions of how to connect to source data, and these can then be easily used in Excel workbooks and when connecting to SQL for consuming KPIs.

A key feature of a dashboard is the ability to filter the data being displayed based on some criteria. This is achieved using filter Web Parts which use Web Part connections to pass parameters to other Web Parts. The type and value of these parameters depend on the type of filter Web Part that is being used, and many of them ship with SharePoint—for example, you can have the current user's details passed as a filter, choose a value from something that has been exposed through the BDC, choose a value from a predefined list of values, or even use a query string that is passed to your aspx page as a filter. Of course the Web Part that receives the filter parameter must know what do with such a parameter in the first place and be able to act on the filter value that is passed to it. We have seen this in action already, haven't we? That's right—the parameter support in Excel Web Services allows us to filter the view of our workbook, and this support can be enhanced by connecting the Excel Web Access Web Parts to filter Web Parts. The best way to illustrate building a dashboard and filtering information is by example, so let's revert to our golfing scenario.

Each golfer is associated with a team, and we have two goals: to allow people to easily browse all individual scores and to just view those scores from a particular team. The first step in all of this is to create a PivotTable in our Excel workbook that allows us to filter on Team Number. In doing so, we name our Pivot Table page field 'teamid' so that we can then publish it as a parameter through Excel Web Services, as described previously. Since the source of our filter for our PivotTable originates from a Web Part in a dashboard, we have hidden the filter field in our workbook to avoid confusion for the end user. Thus, the only way the user can filter the view is via the dashboard. Once we have published our workbook with its available parameters, we can start to build our dashboard. We could do this by hand from scratch, but why go to the trouble when a Reports Library already contains some predefined dashboard layouts for us? In fact, when you create a dashboard you can also choose to have a KPI list associated with it, in which case KPI Web Parts are automatically added to your dashboard and linked with either a new or existing KPI list.

On our dashboard we place two Excel Web Access Web Parts and link them both to our published workbook. We also set the Web Part to only display the named item called PivotTable1. This achieves our first goal of showing all scores on one page. For the filtering, we have many choices—we could allow the user to type in a team number, select from a predefined list, or, as in our example, take the team name from a parameter that is passed to the URL for the dashboard. Doing so allows us to access the desired team anywhere we can enter a link—perhaps manually or by predefined links on our navigation

bar? The latter is what we have done, and you can see the result in Figure 8.6. The links to each team are on the vertical navigation bar and a single click will refresh the page, with the right-hand Web Part filtered using the Team query string. Achieving this filter is very straightforward; you just add a query string filter Web Part to the page and then connect it to the Excel Web Access Web Part. During this connection process, you can choose how the Excel part should use the filter—as a value to a parameter or even as a value to a named item within the workbook. In our case, we chose filter value and selected TeamId, which was one of the parameters we initially used when publishing our workbook.

Easy? Powerful? I hope you agree!

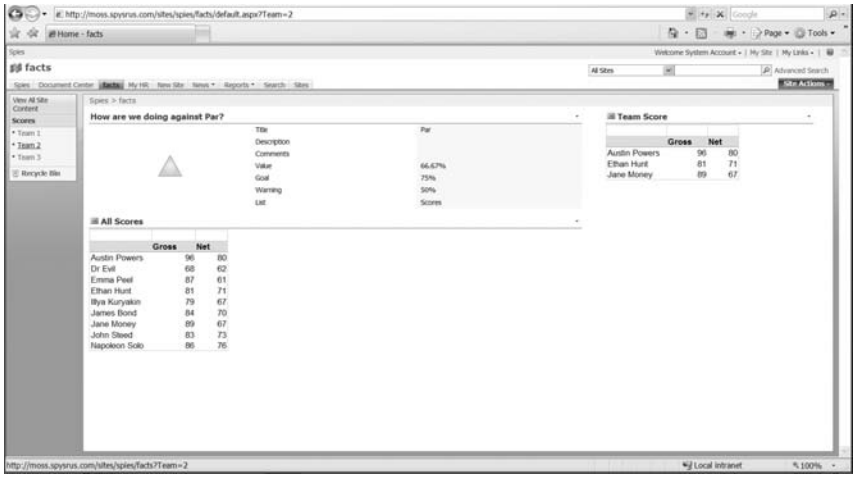


Figure 8.6
Filtering in action.

This page intentionally left blank

9

Office and Exchange Integration

Seldom will SharePoint be used on its own. Indeed, it is highly likely that the majority of installations will be within a Microsoft-centric communication and collaboration environment that includes Office and Exchange. In such an environment, Outlook will be the primary e-mail client, and Word, Excel, and PowerPoint will also be heavily used. Of course, SharePoint is part of the Office family, and it should come as no surprise that there are integration points between it and the client side of Office 2007. Indeed, Outlook 2007 is known as a smart client to SharePoint. And we have already seen how e-mail can be sent to SharePoint and how Exchange 2007's managed e-mail folders can be used to have e-mail filed in the records repository.

This chapter focuses on the integration features in the 2007 versions of Outlook, Word, Excel, PowerPoint, and InfoPath, as well as on how Outlook Web Access 2007 can provide a vehicle for accessing SharePoint document libraries. Note that the details provided here apply to the Professional and Enterprise editions of the Office 2007 suite. Other versions of Office (2000, XP, 2003) offer some level of integration, but the best experience is to be had with Office 2007; see http://www.microsoft.com/downloads/details.aspx?FamilyID=e0d05a69-f67b-4d37-961e-2db3c4065cb9&DisplayLang=en for a paper on the integration levels for these other Office versions and also some information on what you lose if you use Office 2007 Standard edition.

9.1 Outlook 2007

There are four main integration points between SharePoint and Outlook 2007: lists, libraries, alerts, and meeting workspaces. The latter two are very similar in function to Outlook 2003, so we will concentrate here on the former two, which extend the utilization of SharePoint through offline access to lists and libraries by synchronizing content into a special purpose PST (personal storage) file. In Outlook 2003, a read-only copy of SharePoint Event and Contacts lists can be taken offline into a special purpose PST file. With Outlook 2007, we see the welcome addition of read and write access

and support for other list types that significantly enhance the value of your SharePoint lists by making them accessible offline.

9.1.1 Synchronization Process

The first step in the synchronization process is to indicate that you want a list to link to Outlook. If the list type is supported for synchronization, and an ActiveX object called SharePoint.StssyncHandler (.2 or .3) can be created within the current browser session, then the Connect To Outlook option will appear in the Actions menu when viewing the list. The sync handler objects are installed by Office 2003 and Office 2007, so clearly either Outlook 2003 or Outlook 2007 must be installed on the device on which you are running your browser. The first time a user takes this option a check is made to see whether Outlook is enabled for SharePoint integration and, if so, a special kind of PST file is created. By default it is enabled, but this can be controlled via Group Policy. Indeed, a few options can be configured, including pre-populating lists that should be synchronized. The best way to see which options are available is to download the Office 12 Administration Templates from the Office 12 Resource Kit and load them into the group policy editor, as shown in Figure 9.1. As you can see you can control the synchronization frequency, switch on logging, and so on.

Figure 9.1 *Group policy settings for SharePoint/ Outlook integration.*

The PST file—by default called SharePoint Lists.pst in Outlook 2007 and SharePoint Folders.pst in Outlook 2003—is created in the user's Windows profile, and individual folders are created for each list that should be synchronized. Even though the data is held in a PST, the URLs to the lists are also held in the users' mailbox. Why? Outlook 2007 has introduced the capability to have your RSS, Internet calendar, and SharePoint list registrations roam with you, should you so wish. This means that if you have to create a new messaging profile (perhaps you get a new device or your mailbox

moves, etc.), your subscriptions to these various data sources will still work. The data associated with the subscriptions may have to be resynchronized, but at least you do not have to remember all the lists that you had subscribed to in the first place. You can control which subscriptions follow you around by modifying your profile via Control Panel/Mail.

Even though the PST appears to be like other PSTs, it behaves differently, because as users click on the folders (or automatically by default every 20 minutes) synchronization with the associated SharePoint list occurs. You can create other folders within the PST file, but only those associated with SharePoint lists will be synchronized. These can be identified via the double-pointed blue arrow on the folder icon. Similarly, you can move/copy such folders out of the PST into, say, your mailbox, but they will lose their ability to synchronize (even if you were to copy them back in again). There are certain attributes held on the folder that indicate it is a SharePoint based folder—such as the IsSharePointFolder attribute—and supporting items such as date and time of last synchronization and URL to the SharePoint list are held in the associated contents table in the root of the users mailbox.

Outlook initiates the synchronization, either on a scheduled basis or as the user navigates to the folder or modifies content within the folder. You can also include your SharePoint folders in a Send/Receive Group, just like other folders you synchronized, such as those in your cache, RSS, and Internet calendars. For list items, Outlook calls the lists.asmx Web service, using the GetListItemChangesSinceToken method to retrieve any updates that have been made to the list since the last time synchronization occurred. The same Web service is used to push changes that have been made to the Outlook item back to the SharePoint list, except the UpdateListItems method is used. Note that there is no form of conflict checking that occurs, and every time a user visits an Outlook folder, the most up-to-date changes are taken from the SharePoint List before any local changes are pushed back up. You need to be aware of this, especially when you are editing list items from within Outlook when you are truly offline. If any change has been made at all to the server's version of the item then the whole item, will take precedence over any of your changes—even if the individual attributes you have changed have not been touched on the server.

9.1.2 Lists

Not all list types can be synchronized, since Outlook has to be aware of the schema within the list in order to be able to present it to the end user. Therefore, only list types that can be represented by an Outlook item type are supported. Furthermore, custom columns on any list are not supported either as Outlook does not know how to process or display these columns. Table 9.1 shows the list types that are supported and the message class of the corresponding item in Outlook, along with some notes on each type. An impor-

tant point to note here is that once an item is in Outlook, it benefits from all the goodness of Outlook. For example calendar items can be displayed in overlay mode with your own personal calendar, and items can be dragged and dropped between different calendars, be they in your mailbox, in a public folder, or in a SharePoint list. Indeed, Outlook knows that the item is synchronized with a SharePoint item and knows its source URL within the SharePoint site; thus it offers you the option of viewing the item in a browser, giving you a quick way of getting back to the item in SharePoint.

Table 9.1 *List Types Available in Outlook*

SharePoint List	Outlook Message Class	Notes
Calendar	IPM.Appointment	Recurrence is supported. All dates are stored in UTC, and each client displays the date according to its timezone. Outlook uses the timezone of the current user to display the offset, and SharePoint uses the regional settings for the site. Therefore, confusion can reign if you are sharing/synchronizing a calendar between users in different timezones! Best practice is for calendar items to be scheduled with the time zone of the SharePoint site in mind, rather than the end users' time zones.
Contact	IPM.Contact	Only single e-mail, phone, and address fields are supported in SharePoint item.
Discussion Board	IPM.Post	Threaded views are supported.
Tasks	IPM.Task	Uses Custom Status and Custom Priority fields in Outlook item, but these are also linked to Status and Priority fields. Outlook view called Server Tasks shows the correct fields.
Project Tasks	IPM.Task	Same as Tasks.

Figure 9.2 shows the contents of a calendar list synchronized into Outlook. Note how the times are different between Outlook and SharePoint, due to the fact that the user's computer is set to a different time zone from that of the SharePoint site. These items were initially created in SharePoint, synchronized to Outlook, and then dragged and dropped in Outlook to change the start times.

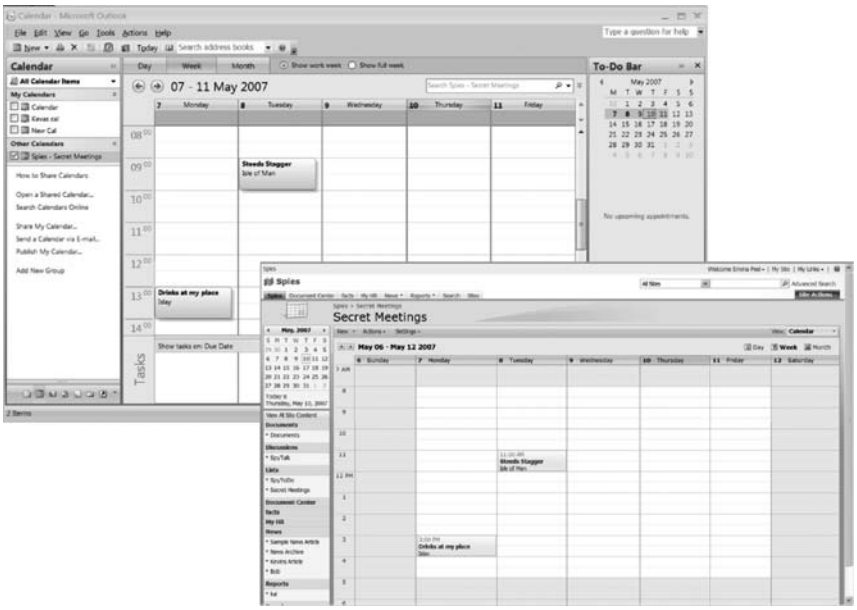


Figure 9.2
Synchronized calendar.

9.1.3 Libraries

The vast majority of library types—Documents, Slides, Forms, Reports, etc.—can all be connected to Outlook using the same mechanism as described above for lists (actually, if you browse to your master pages gallery, you will also see that it can be connected to Outlook—a bug, I suspect, since there isn't much you can do with the synchronized aspx pages from Outlook!). The major difference between lists and libraries in this regard is that libraries are read only from within Outlook. Yes, I hear you sighing with disappointment, but you shouldn't be, as there is support for modifying your offline content via the source application. So, for example, if you have synchronized a Word 2007 document into Outlook, then Word itself can allow you to modify it offline and ultimately have it synchronized back to SharePoint. If you think about this, it makes sense—after all, you cannot edit Word documents directly using Outlook today, can you? All documents in Outlook are essentially read-only, and the underlying application enables writing.

That said, the ability to take offline copies of documents into Outlook brings significant value, especially if you are a road warrior who is not always connected to the network. Outlook 2007 can preview documents without having to launch the underlying application, which adds value and gives you very quick access to your documents.

It is likely that you will synchronize many document libraries into Outlook; therefore, the first time you connect a library, a couple of Outlook

search folders are created for you in the SharePoint PST. One is called Offline Documents and it aggregates all your documents from all your connected libraries. One click and you have everything at your fingertips. The other search folder is called SharePoint Drafts, and it comes into play when you edit an Office document through its underlying application—such as when double-clicking a synchronized document from within Outlook or taking the Edit in Microsoft Word option from the drop-down menu while navigating a document library in a SharePoint site via your browser. So let's look at how Word 2007 deals with this.

As with lists, the lists Web service is used to receive a notification of the documents that have changed; a straightforward GET on those documents is then performed to pull them down into Outlook.

9.2 Word 2007

Word 2007, like other Office 2007 applications, is aware of SharePoint document libraries and knows whether the document that is currently being edited has any connection with a library. We say connection here because it is possible to take offline copies of source documents, edit them offline, and then copy them back to the library at a future point in time. Therefore, you could be editing an offline copy or directly editing the server copy. There are a couple of ways to create an offline copy, the first being through synchronizing the library with Outlook 2007. If you double-click such an item from within Outlook, a read-only copy of the document is opened in Word and an option is presented to you in the information panel for offline editing, as shown in Figure 9.3. In this instance, Word knows that it is not directly viewing the server copy and therefore offers you the option of offline editing.

Figure 9.3
Offline editing option.

If you click the Edit Offline button, a physical copy of the document is made from your SharePoint PST to your local SharePoint Drafts folder. This folder is created for you in your My Documents folder and is used to store offline copies for all documents that you want to modify offline, regardless of which SharePoint document library their source belongs in. Word itself allows you to control the offline editing capability and the location for your offline copies. Offline copies can also be placed here when you use the Check Out functionality available on all SharePoint document libraries. Thus, every item that finds its way into the SharePoint Drafts folder knows where it initially came from.

As you edit the offline copy, Word reminds you about the server copy and permits you to copy your version back to the server or, indeed, to overwrite your local copy with the version from the server in case it has been updated by someone else in the meantime. There are also options for merging changes with the server version. Outlook is also aware of those documents for which you have an offline copy. You will see a visual indication of this in Outlook's user interface, and Outlook will also present all your offline copies through the SharePoint Drafts search folder (this is only for documents that have found their way to the SharePoint Drafts folder via Outlook and not via any other mechanism).

Comparing versions

Word 2007 has the ability to compare documents and is aware of whether the document you are editing has come from a version-enabled document library. If so, you can quickly compare the current version to any previous version of the document. This is done using the Compare section, which appears on the Review section of the ribbon. You can also choose to compare any two files by manually specifying their location. Combining the files together is also possible. You can see this in action in Figure 9.4—the document I am currently editing is in a version-enabled library and I saved a version before I typed this sentence!

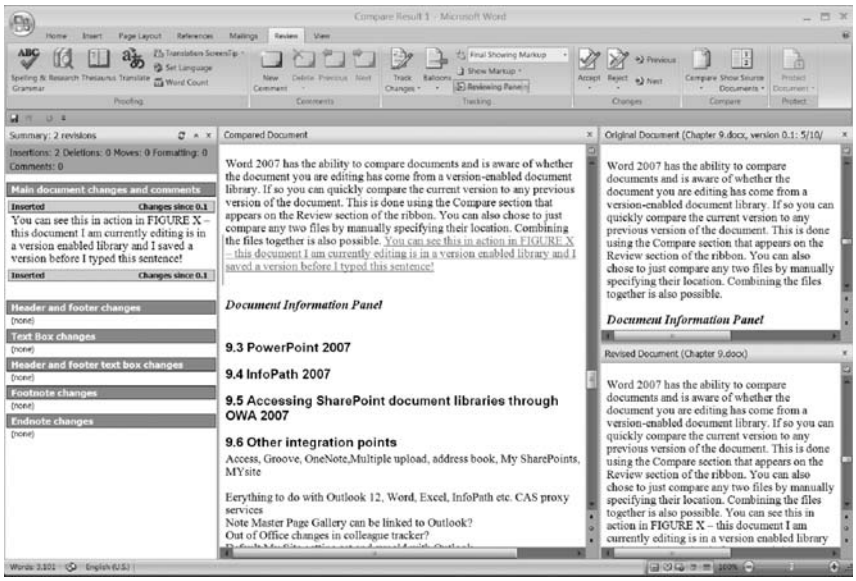


Figure 9.4 *Comparing versions of documents.*

Document information panel

The document information panel reveals itself in Word, Excel, and PowerPoint, and is the place where properties (metadata) associated with the item are maintained. This is where custom columns associated with SharePoint document libraries appear; the metadata exists both in SharePoint and in the document itself. Any restrictions associated with the metadata (such as mandatory fields, choice fields, etc.) are enforced via the panel. You reveal the information panel using the Prepare/Properties option from the Office button (that's the big round button at the top left hand corner of the window in case you don't know).

The document information panel is customizable on a per-content-type basis, and you can use InfoPath to design a form to be displayed. Additionally, you can indicate per content type whether the panel should be displayed upon open and initial save of the document.

9.3 PowerPoint 2007

PowerPoint documents can be manipulated in the same way as Word documents when it comes to synchronizing with Outlook, offline editing, and the document information panel. PowerPoint is also aware of SharePoint's slide library and can be used to build up a collection of individual slides that can then be used as the basis for new presentations. In this way, you can build corporate slide libraries to ensure consistency across presentations.

From PowerPoint, you can load a presentation as individual slides into a library by selecting the Publish/Publish Slides from the Office button. From here, you specify a SharePoint Slide Library and the individual slides are uploaded.

To insert slides into a new or existing presentation, you "reuse" slides from a library via the New Slide menu option. This allows you to browse for slide libraries and to select the slides you want from those libraries. Additionally, you can indicate that you would like to be told if the slide from the library changes so that you can update the presentation accordingly. When you open a presentation, the updated slides are shown to you, giving you the choice whether to replace the slides or not. Of course, you can also initiate a new presentation from the slide library itself by taking the option to insert slides into a presentation.

9.4 Excel 2007

We already discussed in Chapter 8 how the Excel 2007 client integrates with Excel Services, so here we will briefly discuss other generic integration capabilities that do not require Excel Services.

If you used all of the SharePoint integration capabilities in Excel 2003, the first thing you are going to note is the removal of the ability to do a two-way synchronization between Excel and a SharePoint list. Microsoft has stated that Access 2007 should be the vehicle for this type of operation, and whether you agree with this or not, it is what it is. Further discussion on this can be seen on the Excel teams blog at http://blogs.msdn.com/excel/archive/2006/08/24/718786.aspx Importing from a spreadsheet to a list is still possible, as is exporting to Excel for analysis.

You can synchronize spreadsheets from a library to Outlook in the same way as you can Word and PowerPoint documents, and you can work offline and interact with the document information panel.

9.5 Access 2007

A lot of work has been done to integrate Access with SharePoint, and you can now choose to link Access data to items in a list (termed upsizing) and also store entire Access databases as items in a library.

The Open with Access option on the Actions menu of a list allows you to modify the list items directly from Access or to export a copy of the data into a table in a database. When you are editing linked SharePoint data, Access allows you to work offline with a simple click at the bottom right hand corner of the Access window. You are also given a visual indication of which rows in your table have not yet been updated in the SharePoint list. When you next work online, changes from both the local Access table and the SharePoint list are synchronized using various methods of the list Web service such as GetListItemChangesSinceToken, UpdateListItems, and GetListAndView.

You can create a new database from Access and link its tables to SharePoint immediately. After supplying details to the SharePoint site, new lists are created to host the contents of each table. Furthermore, the database itself is published in its entirety to a library, and you have the option of having shortcuts published to the database's views and reports. What does this mean? Any view that is associated with the underlying Access table is automatically added to the list of available views in the SharePoint list. When a user takes one of these views from SharePoint, Access 2007 is launched to display the view.

9.6 InfoPath 2007

InfoPath 2007 integrates with the InfoPath Forms Service, but from a pure client point of view, you can publish InfoPath forms to a SharePoint forms library, just as you could with InfoPath 2003. Once you have designed your form you indicate which SharePoint site you want to publish it to. From here, you can create a new form library or publish into an existing one. In either case, the form itself is set as a template on the library, and any number of fields from the form can be promoted as columns in the form library itself. This allows for data held inside the form to be viewed by users without having to launch InfoPath itself.

9.7 My SharePoints—Navigating Sites, Opening, and Saving items

To date, there have been a few ways of opening, saving, and navigating items held in SharePoint libraries; for example, map network drives or type in URLs to sites via the Open/Save dialogs in Office applications. Not all applications are SharePoint-aware, so the ease with which you could do this was really dependent on the underlying application (albeit mapping a network drive would mask the fact that it was SharePoint in the background and thus any application could open and save directly to SharePoint). Applications that are SharePoint-aware allow you to interact with metadata and browse the content of sites via a WebView integrated into the Open/Save dialogs.

One of the problems users have faced is that they typically have access to many SharePoint sites and therefore lose track of all the different places in which they could open or save items. This situation is somewhat relieved in SharePoint 2007 with the introduction of the My SharePoint Sites location, which surfaces in the Open/Save dialogs of Office 2007 applications, as shown in Figure 9.5. This feature allows you to quickly navigate the sites and libraries that you have access to, making it simpler to open items held in SharePoint libraries. Not only does the dialog show you the libraries and lists within a site, but you can also see the subsites and drill down to them.

You will also note a reference to My Site, which lets you navigate your personal SharePoint site. How are these locations populated? Although these locations are ultimately held in the registry, they are maintained by the My Site feature provided by SharePoint Server 2007. The My Site location is set when you create your My Site (and can be subsequently changed from your My Site if its URL differs from that stored in the registry; in such cases, you are offered the option to Set as Default My Site). Ultimately this is held in the HKEY_CURRENT_USER\Software\Microsoft\Office\12.0\Common\Open Find\Places\UserDefinedPlaces\PersonalSite portion of the registry.

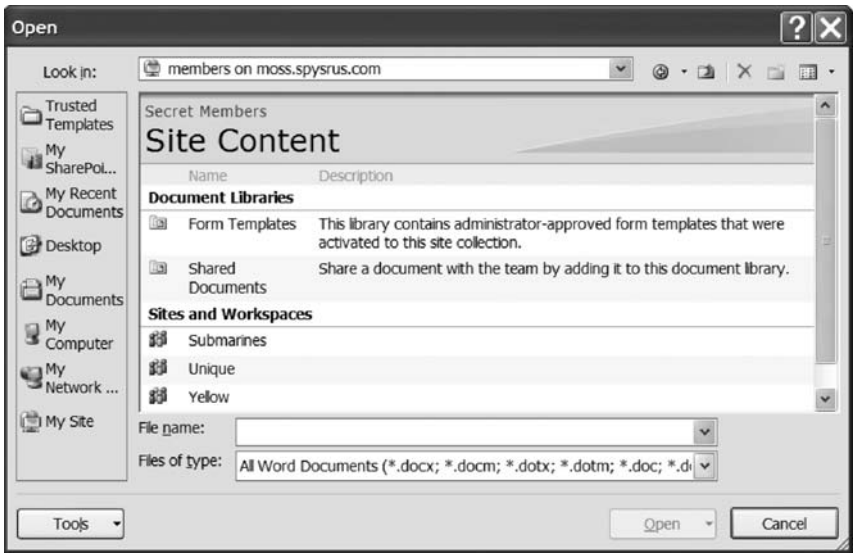


Figure 9.5
Navigating My SharePoint Sites.

Also, when you do set your My Site location, the personal library within your site can be automatically added to Outlook for synchronization.

The sites you see in the My SharePoint Sites location are the same sites available to you from your My Links and SharePoint Sites Web Part, which we discussed in Chapter 7. The Office File Open/Save dialogs read the details from the user's Windows profile at …\Local Settings\Application Data\Microsoft\Office\My SharePoint Sites, and these details are updated as you navigate to your SharePoint My Site, assuming any new sites have been discovered by SharePoint. You can also choose to manually add other locations to this folder; they will appear in the Open/Save dialogs.

Lastly, you can publish links to this location via the SSP. A user's My Site is associated with an SSP, and links associated with that SSP can be targeted at particular audiences. Thus, SSP administrators can create and target specific links for specific audiences that will ultimately show up in the Office Open/Save dialogs.

9.8 Accessing SharePoint Document Libraries through OWA 2007

The last part of this chapter covers the ability to access SharePoint document libraries through Outlook Web Access 2007. This ability ties in with Microsoft's design goal of Anywhere Access, which permeates all of their offerings. The primary scenario that this functionality targets when using

OWA outside your corporate network. This isn't the only scenario in which it works, but it is certainly the situation where its value can be most realized.

An example will help explain what I mean here. Consider how often you have been sent links in e-mail messages to *internal* documents in SharePoint sites. Users browse libraries, copy shortcuts, and then paste them into an e-mail to draw your attention to something. The URL they paste is typically the internal URL that they were using to browse the library in the first place. Now this is all very well if you read the e-mail from inside your network and can resolve and access the internal URL. But what if you are on the road and using your OWA infrastructure published to the Internet? That internal URL will not be resolvable from the outside and ends up in user frustration. It also results in bad working habits in which people attach whole documents to cater for such scenarios, rather than specifying a link to the source document.

OWA 2007 helps tremendously here by resolving and retrieving those internal links on your behalf. In other words, OWA can proxy requests to the internal SharePoint sites as if the actual users were accessing these URLs directly from inside your network. Thus, the above situation is resolved; no matter where you may be physically using OWA, you can still access back-end SharePoint libraries and the items held within.

This same feature can work with file shares. Before you start panicking about any security issues here, OWA cannot access anything that the user does not have at least read access to, and OWA will only allow the user read access to the source item. OWA creates a Kerberos token on behalf of the user and uses this to access the back-end SharePoint sites and file shares. That said, if you happen to publish your SharePoint sites via ISA 2006, then ISA itself will insert a link into the OWA page, enabling the user to launch the full SharePoint site in all its glory and therefore allowing read/write access. Remember, this is only possible if you have taken the conscious decision to publish your SharePoint sites in the first place. Given that OWA allows you to keep a list of your favorite SharePoint locations in your mailbox, no matter where you are, OWA can be your vehicle into all your information sources. Exchange administrators can control whether this functionality is available and also control which SharePoint sites and file shares are available. In addition, you can also use Exchange 2007's WebReady Document Viewing Capability, which can ensure that only HTML versions of attachments are available to the end user and mitigates another common security concern of leaving copies of attachments on unknown devices.

As you can see in Figure 9.6, as users navigate to a SharePoint library via OWA, they can use the Documents option to view all the documents in that library along with details of the user last modified the document; they can also easily look up that user's details in the Global Address List.

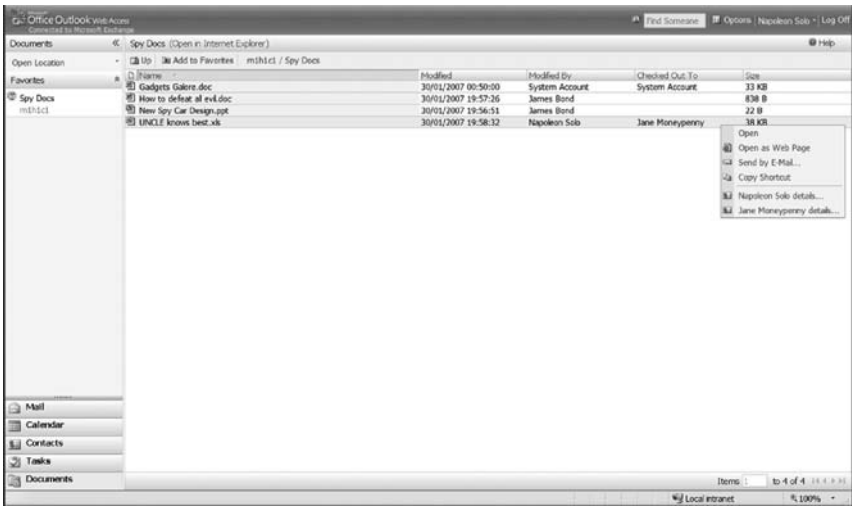


Figure 9.6 *Navigating SharePoint libraries through OWA 2007.*

This page intentionally left blank

10

Planning the SharePoint Deployment

Whether you're planning to deploy WSS 3.0 or MOSS 2007, thoroughly planning the solution and its architecture is key to a successful implementation for your particular business requirements. A well thought-out architecture plan ensures a smooth implementation, and planning the features to be used from the framework ensures that you hit the ground running after deployment. Deeper knowledge of the planned architecture and used feature set also gives you the tools necessary to more accurately size required hardware for the deployment.

In this chapter, we cover the most important areas that you need to consider when planning your SharePoint deployment so that you can implement your environment with confidence.

10.1 Getting to Know the Requirements

Before even thinking about SharePoint configuration or hardware specifications, you should spend time to gain a thorough understanding the usage scenario. What is the environment going to be used for, and how is it going to be used? These questions simple enough, right? But often as not, the answers are not well thought-out in advance of any deployment!

As we said in our previous book, when you buy a second car for your drive to the office every morning, you might decide on that sporty two-seater. However, after discussing it over with your partner and realizing that the car also needs to be used for weekend drives to the countryside with the family, your shiny new two-seater has less chance of ever pulling into the driveway. What you planned to use the car for was affected by how the family expected it to be used, thus tipping the scales on your choice to go for the comfortable people carrier instead.

The same rules apply to a SharePoint deployment, and with SharePoint 2007, there are a lot more features at your disposal. Thus, although the environment might primarily be designed for use as an enterprise search portal,

end users after discovering the collaborative features of WSS 3.0, might use the environment mainly for team collaboration. We learned this lesson in the HP Services Knowledge Management Team early on, when we simply "turned on" SharePoint Team Services (the previous version of Windows SharePoint Services) thinking no one would really use it. You can imagine our surprise one year later, when we discovered that country business units depended on the service being available 24/7, with estimated downtime costs of $100,000 per hour for the business.

Spend time finding out the answers to "what" and "how" by talking not only to the deployment champions and the company management, but also to the future end users. Often you will find the end-user expectations to be quite different from management vision. Managing the end users expectations and balancing them with management's priorities will help adoption and user acceptance once the system goes into production. Talking to users is especially important if you are planning to migrate from an existing legacy system, as users often have novel ideas on how to apply the technology to their particular business problems.

10.2 Planning SharePoint Features

As you have undoubtedly come to realize by this stage of the book, Windows SharePoint Services 3.0 provides a rich and extensible framework for collaboration. Add on top the portal, search, business intelligence, and content publishing features of Microsoft Office SharePoint Server 2007, and we're sure you can imagine endless options for all manner of implementation scenarios across an enterprise.

While we have already covered all the major features earlier in the book, it's good to highlight a few of the key areas you should consider early on in your planning cycle. While some features are easy to turn on/off and reconfigure throughout the deployment process, others, such as the Authentication Provider, can cause considerable work to change at later stages of a deployment process, since they typically involve supporting infrastructure.

10.2.1 Authentication Providers

Previous versions of SharePoint were tied to security principals that had to translate to a Windows identity. While it was logical for SharePoint products to do so, given their background and focus, this requirement did limit organizations that were using third-party user directory servers, such as Netscape and Novell directory servers. While it was technically possible to utilize third-party plug-ins that translated non-Microsoft directory user accounts to domain identities, they came with performance, support, and

maintenance costs, that discouraged their use in any but a few of the deployment scenarios.

With the new focus on extranet collaboration and content publishing, Microsoft listened to customer feedback and delivered an open framework for authentication in SharePoint 2007. Thus, SharePoint 2007 is the first version of SharePoint that supports methods of authenticating users against other directories than Windows domain or AD infrastructure, opening a wide array of architectural, options for both extranet as well as intranet SharePoint deployments.

SharePoint authentication providers are built upon the ASP.NET 2.0 pluggable authentication provider model, are providing a solid foundation for creating custom authentication providers and access to numerous existing ones. While WSS 3.0 does not ship with any authentication provider modules out of the box, ASP.NET 2.0 itself ships a module for SQL-based authentication. There are numerous pluggable authentication modules examples already available on the Web, such as a nifty little module for WSS list-based authentication for smaller, external-facing collaboration sites. MOSS 2007, on the other hand ships with a supported LDAP v3 provider, which will provide basic functionality for most organizations with modern directory servers that support LDAP v3.

While in a typical intranet SharePoint deployment scenario you will still use the Windows domain for authentication, the ability to choose different authentication providers really comes into play when planning an extranet deployment. In most extranet scenarios, you typically want a group of authorized business users to be able to create user accounts for the external parties in an ad hoc manner and manage the life cycle of these accounts without direct IT involvement. Thus, the ability to choose a lighter user management platform or use an existing directory server you have in the company product portfolio is a very compelling option.

10.2.2 Enterprise Search Scoping

In large corporations with disparate knowledge systems, the only way to present a cohesive front end to the knowledge workers is through an enterprise search facility, which integrates all the knowledge silos together into a single discoverable index. Microsoft SharePoint Portal Server 2003 provides an extensive search framework to allow enterprises to break down knowledge silos and facilitate effective knowledge reuse within the corporation. However, without effective planning, an enterprise search portal can quickly become unusable as aging content and uninteresting content are included in the search results. It is extremely important to prevent information avalanches and to carefully plan the content sources to include in the search catalogs. With selective use of Search Scopes, organizations can drive user

behavior to be more exclusive when searching for content, thus returning more relevant search results.

Out of the box, the SharePoint Indexing service is able to generate full-text indexes of Microsoft Office Word, Excel, PowerPoint, Visio, Publisher, HTML, TIFF image, and MIME content and text files. As you can quickly see from the list, at least one major content type most organizations want to include is missing: Adobe PDF.

Since the SharePoint Search service uses the standard IFilter interface to index document content, there are a host of filters available for content types ranging from MP3s to Adobe PDF files. Some of the IFilters, such as Adobe PDF, are available free of charge, but some, such as the ZFilter for ZIP files, come with a cost attached.

Since each IFilter must process the files into a form the word breaker is able to process, certain IFilters will apply more load on the index system than others. For example, TIFF image Optical Character Recognition (OCR) is far more process-intensive than indexing Microsoft Word files. If you plan to index large amounts of TIFF, PDF, or ZIP files, you should consider moving processing of these content types to a dedicated index server.

10.2.3 User Profiles and My Sites

Microsoft Office SharePoint Server 2007 builds upon the foundation created for user personalization in SharePoint Portal Server 2003. We could probably count the number of organizations that actually put thought and effort in making sure SPS2003 My Sites were used in an effective manner on one hand. The problem with My Sites to us has always been the lack of immediate end-user productivity features that would keet driving them to come back after the initial cool factor had worn off. In HP we, for example, simply put it out there for users, without much thought on what they were supposed to do with it. As such, we probably got 200 to 400 users who actually went in and updated their User Profile beyond the standard AD imported fields. These sites eventually withered and were in fact cut from the standard platform offering, due to lack of use and mainly lack of direction for the service.

There's a lesson to be learned there, because when it is properly planned and delivered, My Sites (with the right balance of user profile information) can be a true asset to any organization. The key in my experience is to ensure that you have something on the personal page of My Site that keeps driving users to the page. They will eventually discover the rest of the functionality and start using add-on functions such as shared links and blogs, and actually take time to fill profile fields that cannot be imported from most company directories, such as skills and interests.

User profile fields are fully customizable, allowing you to define exactly what data you wish to store as part of a user profile. Each field definition allows you to define whether the field should be hidden from users' or displayed on users My Site information page, and whether or not users are able to edit the field values. You can import existing user information from AD and map AD fields to user profile fields, in case your organization has extended the schema, for example. Realizing that in many organizations AD might not be the people directory of choice, MOSS 2007 allows you to synchronize people profiles from other sources, such as an LDAP directory.

When planning user profiles you should first get clarity regarding which profile fields your organization wants to collect and which ones should be visible to everybody via the public view of My Sites. You can also define fields that the users can edit themselves, although they will not be synchronized back to the company directory unless you create a custom write-back synchronization tool. If possible, you should consult your organization's privacy officer to verify that data protection regulations are adhered to with respect to the personal data stored in the user profiles. Since regulations differ drastically from country to country, data protection considerations are especially important when planning a global deployment. Once all the stakeholders agree on the data fields to be stored, you should map where, if anywhere, the information should be synchronized from, while keeping in mind that synchronization will always override user editable fields.

10.2.4 RSS Feeds

RSS, or real simple syndication, is an industry-standard way of delivering frequently updating information in standardized format. RSS grew popular in Web blogging environments because it aggregated hundreds of blogs to a single reader easily. It was quickly adopted by all major news networks as well as by Web sites such as MSDN to deliver the latest articles to end users.

It's not surprising, therefore, that RSS is a key part of underlying functionality in SharePoint 2007. SharePoint lists and document libraries, after all, are frequently used to deliver on-time latest versions of news or files to corporate end users. As we covered earlier in the book, RSS feeds must first be enabled at the Web application level for your SharePoint deployment before end users are allowed to create feeds for their lists.

Before diving in and enabling RSS feeds on every SharePoint Web application you're going to deploy, you should carefully look at the usage scenario and performance impact on your deployment. For example, enabling RSS in a large collaboration environment with projected 30,000+ team sites might not be the best idea, unless you account for the added load each RSS enabled list and potential reader will add, as the recommended time a reader should wait before requesting for updates is determined at the site collection level. In

high volume collaboration deployments, you should at least plan to proactively educate your users to only RSS-enable lists that deliver frequently changing information should be delivered as soon as possible to end users' desktop RSS readers, such as Microsoft Outlook. Of course, it helps to educate the users to set their recommended update schedule to be something other than five minutes.

If you plan on using RSS feeds widely in your SharePoint deployment, you should take time to estimate the number of feeds a user is likely to subscribe to (with the recommended update schedule) and use this information in deployment sizing calculations, as every RSS reader request for updates will count as an operation in the capacity planning calculations.

10.2.5 Impact of Web Parts

All applications that execute code to accomplish a task have a resource cost, and since Web Parts are essentially small applications, they too require processing power and computer resources. It is very important in an enterprise deployment that has thousands of users accessing the environment every day to properly test each new component, ensuring that it does not exhaust resources and thus risk affecting the end users' perceived performance.

You can get very sophisticated when developing Web Part test procedures using various tools and performance analyzers, but one should balance the effort it takes to test Web Parts with the effort it takes to create and modify them. Although you could run load tests on each new component with sophisticated performance analysis tools such as Segue SilkPerformer (http://www.segue.com/products/load-stress-performance-testing/silkperformer.asp) and ensure every Web Part complies with .NET 2.0 code best practices recommendations, the effort expended is usually double the time it takes to create and debug a Web Part in the first place.

A middle ground can be found with a robust performance test tool called Microsoft Application Center Test (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/act/htm/actml_main.asp), which ships with Microsoft Visual Studio.NET. The Application Center Test (ACT) is perfect for lightweight tests on Web-based applications overall throughput and can be directly called from within the Visual Studio UI when developing new components. Even without setting up complicated variable-based tests and realistic models, you can detect significant performance issues with Web Parts by using ACT.

Upfront performance testing during the development and testing phases can ensure that no surprises are hidden in the new whiz-bang Web Part ready for production. However, often enterprise deployments are managed by dedicated operations and support teams who are in charge of day-to-day management and responses to end-user queries. Thus, proper install

documentation, installation package, and functional description will prepare the operations and support groups for deployment and end-user queries, and these should therefore be required for all add-on Web Parts in an enterprise deployment.

In a larger deployment with multiple servers running SharePoint, version control on all aspects of the deployment (including Web Parts) prevents servers going out of sync and thus risking issues with incompatible software versions or bug fixes that have only been installed on part of the overall deployment. Perhaps the main challenge with add-on Web Parts is support—more precisely, the question "who do I call in case of problems." Contracted custom development might not include a support agreement, which could possibly put the operations team in a difficult position when troubleshooting issues that might have something to do with custom Web Parts.

Even if the source code was provided as part of the submission, debugging someone else's code without knowing exactly what to look for is a nightmare for any developer. It is imperative that a clear support and change management responsibility is established before going into production with add-on functionality. When possible for in-house or contract-developed code, a business group or another entity should be assigned as the primary owner instead of the original developer, since people come and go and Web Parts might therefore be left orphaned.

10.3 Overview of Deployment Sizing

No doubt, one of the most important parts of planning a SharePoint deployment is getting nitty-gritty with the physical hardware. The days of simply getting the biggest box out there in hopes of having it run the solution somewhat smoothly are long gone, and nowadays estimating the application workload, turning it into meaningful numbers, and selecting a hardware architecture that will perform the job to specifications with room for growth has turned into an art form. Power consumption, cooling requirements, and data center real estate are foremost on the minds of most architects when they size applications for deployment; this is a change from the past, when such things never even made it to the equation.

In this section, we look into the art form of server sizing and give you the tools you require to make the best possible choice for your deployment. While we touch on the add-on services and products of the Microsoft Office System, it should be noted that the content is focused on Windows SharePoint Services and SharePoint Server 2007 capacity planning.

In traditional client-server applications, such as Microsoft Exchange, emphasis for server sizing is on the number of users and type of clients they are using. Typically, the types of actions users are able to perform are limited

and thus the performance impact of each user can be precisely estimated. Often the resource cost of each client connection, which is maintained throughout the session is the limiting factor due to memory usage per connection. In a high availability Exchange deployment, there are well defined numbers of users a specific hardware configuration can support. Although sizing an Exchange environment is still an art form due to various storage options available today, planning typically centers on the number of users the environment should be able to handle.

Since SharePoint is at its heart a Web Application, there are no constant connections between the users and the server. A browser requests a page, the server responds and the connection is closed regardless of how long the user spends looking at the returned information. In addition the performance impact of each request varies depending on the type of action performed. Since each action type has a different resource cost, to be able to accurately devise the overall resource requirements you must be able to predict the way the environment will be used. When sizing a SharePoint deployment your emphasis should be on trying to understand the type of actions your users are likely to perform and how often they will perform them.

If after reading this chapter you want to delve further into the capacity and hardware planning for SharePoint, you can find some Sizing Tools and Performance WhitePapers on the HP ActiveAnswers Web site (http://www.hp.com/solutions/activeanswers) as well as the Microsoft Web Site.

10.3.1 Performance Testing Methodology

As you've undoubtedly found out by now, MOSS 2007 features a broad range of functionality that is so extensive that it is nearly impossible to come up with a model that replicates all the possibile tasks that an information worker could do in the course of a typical day. Thus, generalizations must be made when performance testing by combining key functions such as browse, search, and document upload into batches called workloads. The workload defines the types of operations and the frequency of their occurrence for a simulated user. To simulate real life conditions slightly more accurately, typically a randomize function is added to each type of action, because users do not perform the same sequence of actions on every visit.

Every platform has its limits on the number of actions it can simultaneously perform, which is called system throughput. The selected Performance Testing suite applies the workloads on the application at an ever-increasing volume until the throughput levels off or starts to decline, which indicate that one of the system components is not able to keep up with the pace. This is called a bottleneck, since the combined result of the system hinges on the component's performance. Think of running the water at full force in your bathroom sink;

eventually the drain hole is not able to keep up, thus becoming a bottleneck, and the water will overflow.

SharePoint offers two ways to combat the ever-increasing throughput: scale-up and scale-out. Scale-up means adding more processors, memory, or disks to the server you have deployed, but obviously you can only upgrade so far before you have to upgrade to a whole new server platform. Alternately, you can scale-out by adding more servers to the deployment and distributing the load between them. In performance testing, the goal is to find a way to predict system performance in both scale-up and scale-out scenarios, so it's typical to test a good number of different configurations to try to find a formula that can be applied to various scenarios.

With SharePoint the throughput is measured in the number of requests per second (RPS) the system is able to perform. If you're familiar with traditional Web application performance measurement, term "requests per second" might be a bit confusing used in this context. RPS is the favored term in Microsoft documentation, however, and in Web application sizing terms translates to operations per second (ops/sec), since each request underneath the scene actually generates several HTTP requests to resource files. So, from here on, when we talk about estimated and actual server throughput, we will use the slightly marketing-like term RPS.

A request (operation in traditional Web application context) is a sequence of physical events, such as opening a Web site homepage, which appears to the end users as a single request but in reality consists of numerous transactions behind the scenes, such as fetching an included JavaScript or StyleSheet file. Each Web Part adds to the complexity of the request, since the server must do more work to be able to provide the end result to the user.

Requests per second (RPS) doesn't equate to simultaneous connections in terms of, for example, Exchange, since the connection is stateless; the number of operations per second is typically considerably higher than the number of users currently visiting the SharePoint deployment. It's not uncommon to see thousands of users simultaneously visiting a portal, for example, and only generating less than 10 RPS. Don't confuse the simultaneous users with the actual number of users in the company either; they are drastically different, unless you plan to integrate a line of business application that everyone in the company must have open on their desktops at the same time. Typically, the number of active users per day is around 40 percent in a very active corporation; we've typically found most organizations to be around 15 to 20 percent, even for a corporate portal.

10.3.2 Estimating Required Throughput for SharePoint

Why are we talking about all of these weird and perhaps uninteresting terms? Later in the chapter, when we discuss sizing hardware, the estimated

throughput requirements for your deployment will be derived from all the things we've just mentioned in this section of the chapter.

Microsoft has devised a formula that you can use to calculate the estimated throughput volume based on generalizations and estimations. The formula has proven to be quite accurate, but it requires you to estimate the number of users, the percentage of those users who are active on a given day, the estimated number of operations per active user, and the number of hours in a workday in order to calculate the average operations per second requirements. Obviously, the accuracy of the average requests per second depends on the accuracy of the provided estimates.

In organizations without previous implementations, determining accurate figures to feed to the formula is often an educated guess at best. However, existing environments such as eRoom or Lotus Notes can provide detailed logs for you to analyze usage figures and patterns, such as average users and operations per day. Even analyzing existing File Shares might give you some idea on typical active users/operations per day in your enterprise.

Table 10.1 details the formula used to calculate required requests per second (RPS) for a SharePoint Product and Technologies deployment, which you can use to estimate the required hardware topology for your enterprise.

Table 10.1 *Calculating SharePoint Ops/Sec Requirements*

number of users	*X*	*percent of active users per day*	*X*	*number of operations per active user per day*	*X*	*peak factor*
360,000			*X*	*number of hours per day*		

- *Number of users* is the possible number of users who might connect to the system at any given time; if you're planning to deploy SharePoint across the enterprise, it is obviously the number of NT or AD accounts that exist and, as such, usually covers all the employees and contractors. If the environment is going to be accessible by external parties, such as partners or customers, be sure to include them in the figure.
- *Percent of active users per day* requires you to estimate how many of the potential users will actually visit the deployment on any given day. As I noted earlier, 40 percent is a safe bet in a typical usage scenario; however, for an enterprise portal with applications that information workers are expected to use as part of their daily work, I have used figures up to 60 percent on projects.
- *Number of operations per active user per day* might be even harder to predict, as it requires you to consider how the environment will be

used. Again, a safe bet is 10, which has been determined as the average for a typical enterprise. Obviously, there will be power users and content authors who will perform considerably more operations, but they are typically balanced out by the significantly larger reader population who might only visit the Web site homepage and download a couple of the latest files.

- *Peak factor* is a bit of a dark horse; it's a multiplier, typically between 1 and 10, that is used to indicate the peak hour extra throughput requirements. For example, consider a factory which opens at 9:00am and closes at 5:00pm. Between 8:45am and 9:00am, when the employees arrive, the coffee vending machines are typically under significantly increased throughput demands. Thus, to be able to meet requirements, the vending machines have likely been sized for this peak capacity. Similarly with SharePoint it's only natural that there will be peaks in the usage pattern, although in a global or cross-regional deployment these peaks will be averaged by the time differences. A safe bet is to go with 5to 7 in a typical enterprise deployment, but if you know that there will be a considerable peak during specific hours of the day, you should apply a higher peak factor.
- *Number of hours per day* is the time span during which most of the users will use the environment—typically, the average working day for the user population the environment is targeted for. However, as with the peak factor, cross-regional and global deployments will have to count for the time zone differences. The hours per day is multiplied by 360,000, which is derived from: *100 (for percent conversion) times 60 (number of minutes in an hour) times 60 (number of seconds in a minute)*.

As an example, let's take an enterprise portal deployment for a 40,000-user organization across United States. Since the portal is targeted at every employee, there will be 40,000 potential users, with a reasonable estimate that 40 percent of them will visit the environment on a given day. A typical operations-per-day sum of 40 is used, with a relatively low peak factor of 5 given the time zone impact across the US. We'll use a 12-hour working day, again given the cross-country time zone impact. As you can see in Table 10.2, the end result is an average throughput requirement of 74 RPS. We'll talk about what that means in hardware terms in just a while.

Table 10.2 *Example Requests per Second (RPS) Estimate Calculation*

40,000	X	40	X	40	X	5	**= 74 RPS**
360,000			X	12			

Not only are the formulas generalized, but they are also based on a number of assumptions on how the environment will be used by the end users. Next we'll look at some of those assumptions, since regularly exceeding the recommended application limits will result in unpredictable performance characteristics.

10.3.3 Recommended Application Limits

Every product has practical and technical reasons to recommend certain limits on how the application should be used. Microsoft has made several recommendations for both WSS and MOSS 2007 feature utilization, which gives you guidance as to how you should structure your deployment and information architecture. While hardly any of these recommended limits are enforced in the application, and as such referred to as *hard limits*, typically the recommendations have been made for a good reason. In SharePoint's case you will find that exceeding the limits results in decreased performance, with longer response times for end users.

Table 10.3 details the recommended acceptable performance guidelines for Windows SharePoint Services 3.0, and as such applies to WSS sites hosted from MOSS as well.

Table 10.3 *Recommended Limits for Windows SharePoint Services 3.0 Components*

Feature	Recommended Limit	Notes
Site collections	50,000	Although not a hard limit, Microsoft recommends splitting site collections across virtual servers if you plan to create more than 50,000 site collections, since performance has been shown to decrease steadily after 10,000 site collections.
Web sites per site collection	250,000	Several sites can be created within a site collection by utilizing subWeb hierarchies.
SubWebs per Web site	2,000	Due to limitations with subWeb enumeration routines, you should limit subWebs to under 1,000 per Web site.
Security principals per Web site	2,000	Personally I recommend that users utilize Windows Security Groups (AD or NT4) to grant access to a Web site if there are more than 200 users. Technically, though, you could add up to 2,000 security principals before the enumeration and rendering starts to get too slow to become useful.

Table 10.3 *Recommended Limits for Windows SharePoint Services 3.0 Components*

Users in groups	2,000,000	It is possible to create extremely large user groups to allow access to a partial set of company users, for example, with the upper limit set at 2 million users.
Lists per Web site	2,000	Once again, I suggest that usability limitations will face you far before the technical rendering-related issues catch up. Presenting 2,000 lists without a hierarchical navigation to end users will undoubtedly result in a lot of confusion.
Documents per library	5,000,000	WSS allows a significant amount of files to be stored in a document library, as long as you observe the below per-folder recommendation.
Items per view (for example, documents)	2,000	Like few of the previous items, enumeration and rendering is a likely reason for recommending a 2,000 document limit in a given view. Once again through, usability and the ability to find the information in a large flat structure is a likely limitation well before the technical limit is reached. Thus this point is likely moot, since most users are familiar with creating information structures with subfolders allowing considerable number of files to be stored in a WSS library.
Field type per list	256	While some custom lists are likely to have a good number of field types, we have certainly never ran into one that went anywhere near the current limit of 256.
Columns	2,000 for doclib 4,096 for list	Just as we mentioned with field types, we've yet to counter a real-world scenario that required anywhere near the recommended limits for document libraries and custom lists. However, it's good to know the limit is certainly higher than the usage cases we can imagine.
Maximum document size	50MB (2GB Max)	The index process has issues breaking up large Office files, thus you should seriously consider limiting the file upload size to about 50MB. If your users must be able to store large files, such as CD images or video files, it is possible to raise the limit to 2 gigabytes.

Table 10.3 *Recommended Limits for Windows SharePoint Services 3.0 Components*

Web parts per page	50	Once again, a usability will likely be a limiting factor, as no one is interested in sifting through lots and lots of information presented on a single page. Thus, the 100 Web Part recommended limit is unlikely to be approached in a typical usage scenario.
User profile	5,000,000	With the new target set firmly for extranet publishing, all aspects of the product have been extended to reach well beyond the million-user portal scenario, including the supported number of user profiles.

Table 10.4 lists architecture limit recommendations for all SharePoint farms.

Table 10.4 *Recommended Limits for SharePoint Logical Architecture Objects*

Feature	Recommended Limit
Shared Service Provider (SSP)	3 (per farm)
Zone	5
IIS application pool	8 (per Web server)
Site collection	50,000 (per Web application/database)
Content database	100 (per Web application)
Web application	99 (per SSP)

Table 10.5 lists recommended limits for SharePoint Search, and is such mainly related to MOSS 2007 deployment.

Table 10.5 *Recommended limits for SharePoint Search objects*

Feature	Recommended Limit	Notes
Indexed documents	50,000,000 per Search Index	SharePoint Search has been updated, along with all the other components, to support ever larger enterprise requirements including a considerable amount of documents that need to be indexed.

Table 10.5 *Recommended limits for SharePoint Search objects*

Search Indexes	1 (per SSP)	With the continuous index propagation, it is no longer required to split the search corpus into multiple content indexes, thus really bringing the number of search indexes down to a single one per SSP.
Content Sources	500	When crawling significant amount of external content sources, the 500 recommended content sources might seem like a limiting factor. However, there are many ways around this issue if it really becomes an issue, including, for example, a Web page that simply links to all of the thousands of content sources you'd like to limit. As long as you set the site hops above 1, you'll be able to index various smaller content sources.
Search Scopes	200 (per site)	From a usability point of view, search scopes should be kept to as few as possible, as otherwise users might have difficulties picking the right one from the extensive list. Consider keeping the number below 25 for usability's sake.
Alerts	1,000,000 (per SSP)	One million alerts might sound like a considerable amount, but you should take steps to ensure that you will not go over the limit in an enterprise deployment. For example, 10 alerts per person is not unheard of, and in a 100,000 user organization that's all that users would be allowed to create. Consider limiting alerts per person in a larger deployment to a number that ensures you won't go above the recommended limit.

While the recommended limits detailed in the three tables are not surprising, it's good to remember a few of them for future reference for example, the recommended limit for List items which you are likely to see an end user exceed.

10.4 Sizing SharePoint Server 2007

We've found, some people have preconceptions about Windows SharePoint Services performance, given that the product is offered by Microsoft free of charge while Microsoft Office SharePoint Server 2007 is not. Perhaps the fact that WSS is a free product makes people think that it is only suitable for small deployments and that you must pay for the enterprise scale product, namely MOSS 2007. Undoubtedly, if you've made it this far in the book you

know that such misconceptions couldn't be further from the truth. Rest assured WSS scales, and it scales very, very well, as you're about to find out.

Microsoft Office SharePoint Server 2007 on the other hand, having been built on the foundation provided by Windows SharePoint Services, adds a number of services to the equation, such as content publishing, business intelligence services, and enterprise search, each adding to the complexity of accurately predicting performance requirements. Since MOSS 2007 is built firmly on WSS 3.0, all of the sizing guidelines discussed earlier also apply to MOSS 2007, with slight variances and a few added services which we will cover shortly.

It might come as a surprise to most people already familiar with the previous versions of Windows SharePoint Server or SharePoint Portal Server that strictly comparatively speaking, performance has reduced by 40 percent in the new version of SharePoint. This is natural, due to the new functionality added to the product, which has significantly increased the underlying platform operations required to compile the requested page. For example, security trimming the UI, fine-grained security item security, and the recycle bin are a few features that require extra CPU cycles to compile the end result sent to users. Those of us familiar with the old version are quick to vouch that the drop in comparative performance is well worth the added features.

The drop in performance on the same hardware between the old and the new version should not be a cause for concern unless you are upgrading a SharePoint Portal Server 2003 deployment that is already running at high CPU load, and you were hoping to reuse the same hardware topology.

Of course, Microsoft will be quick to point out that while the product has lost in comparative performance on the same hardware platform, the hardware platforms themselves have gained up the difference by increasing CPU speeds and especially the number of CPU cores on die. It seems that the old saying, "What Intel gives, Microsoft takes away," is still true.

10.4.1 Web Front-End Scalability

We'll dive into the results in just a second, but first it's good to take a quick look at each of the components that make up a SharePoint deployment and how each of them perform before looking at the full picture.

At the time of writing, the argument for 64 bit is usually the topic of the day. Microsoft has gone as far as stating that they categorically recommend 64-bit everything for SharePoint 2007 while still stating that they will continue to support the 32-bit platforms. As you will see later in the chapter, this is an interesting point, since in raw RPS-64 bit servers in fact categorically perform 10 to 15 percent below the 32-bit counterparts. So why recommend 64 bit? In a word: memory.

HP's performance testing of SharePoint front ends has shown that a modern dual socket, dual core server platform is able to perform a maximum of 100 to 120 RPS with a typical mixed read/write operations workload. This number drops to approximately 85 RPS if you run the query service on the same frontend server. If you plan on a single server installation with all services as well as SQL Server running on the same physical server, HP's tests indicate expected throughput to reach 50-75 RPS.

SharePoint Web front end servers are still without a doubt CPU bound, at least with today's typical memory configurations in a datacenter network environment. This means that in theory, you can increase performance by adding additional CPUs or by adding additional Web servers. I'm happy to confirm that the theory holds true, and WSS has shown to be an excellent example of both scale-up and scale-out performance. For example, HP tests have shown that a 4-socket dual-core server performs 1.8 times the throughput of a 2-socket dual-core server.

Scale-out, as expected, doubles the effective throughput performance on any given platform, and both HP's and Microsoft's testing has shown that the trend will continue as long as the database server can keep up the pace. In fact, with modern CPU and memory architectures, large-scale deployments are likely to run into network scalability issues rather than anything else.

10.4.2 SQL Server Impact

Since all of the SharePoint data apart from the search index is stored on the SQL Server, it plays a crucial role in scalability.

While you can install SQL Server and SharePoint on the same physical server, for full scale-out options now or in the future we highly recommend separating the SQL Server role from the Web server role. Even if you are not going to deploy a full-fledged farm immediately, future transition should you ever need to do so is considerably more straightforward when you don't have to worry about SQL Server. It seems that most Database Administrators (DBAs) are sworn to never allow other applications to be installed on their precious database servers, though personally I have not seen any operational reasons to do so. Still, best practices are formed over time, and I tend to recommend running SQL on a separate server to put DBAs at ease and make the operational model slightly more straightforward.

HP's testing has shown that with a separate SQL Server running on a 2-socket dual-core X64 2.66Ghz Intel server, the previously mentioned Web front ends were only using 34 percent of CPU capacity on the SQL Server. This indicates that you could scale-out the Web front ends up to 6 WFEs (4 with mixed read/write workload) before you would need to add capacity on the SQL side. I have to note, though, that the 6:1 ratio is highly dependent on the number of write operations being performed as part of the workload.

We therefore recommended that you adopt a much more conservative 3:1 ratio for production deployments, or scale up the cores on the SQL Server.

10.4.3 Choosing Your Deployment Hardware Topology

With the talk of all of these operations per second figures, you might be slightly confused as to which hardware topology you should choose for your deployment of Windows SharePoint Services. First, let's remind ourselves with the formula to estimate the required throughput requirements, shown in Table 10.6.

Table 10.6 *Formula to Estimate Required Throughput Requirements for WSS*

number of users	X	percent of active users per day	X	number of operations per active user per day	X	peak factor
360,000			X	number of hours per day		

Perform a quick calculation for your environment, and first of all, let's look at the Web Front End (WFE) performance figures for different modern CPU platforms in Table 10.7. All tests were conducted using dual-core 2.66Ghz Intel X64 SQL Server 2005 as a back-end platform, with the exception of the first result, in which SQL server was installed on the same server as WFE to illustrate a single server installation.

Now let's refresh our memories with the earlier example of the 40,000 user organization with a typical user load. We won't quiz you on the topic, but if you recall, the estimated throughput requirement for that scenario was 74 RPS, which will hopefully illustrate that even with the new features adding CPU load, SharePoint performs extremely well.

Table 10.7 *Example Throughput Results from HP's Performance Testing Lab*

Server Topology	Estimated Server Throughput	SQL CPU	Estimated Client Network Traffic
1 WFE 2-socket dual-core 2.6Ghz AMD 285 X64 with SQL server	85 RPS	12%	6000 KB/s
1 WFE 2-socket dual-core 3.0Ghz X86 + SQL Server	126 RPS	34%	9000 KB/s
1 WFE 2-socket dual-core 3.0Ghz X64 + SQL Server	120 RPS	34%	9000 KB/s

Table 10.7 *Example Throughput Results from HP's Performance Testing Lab*

1 WFE 2-socket dual-core 2.6Ghz AMD 285 X86 + SQL Server	112 RPS	31%	8200 KB/s
1 WFE 2-socket dual-core 2.6Ghz AMD 285 X64 + SQL Server	105 RPS	30%	7700 KB/s

For those truly interested in the topic, the results do reveal some interesting observations for X86 and X64 as well as AMD vs. Intel platforms. It appears that at the time of writing the book, Intel still held the performance lead in both X86 and X64 hardware platforms. However, the difference is too close to call for any kind of review on your chosen CPU vendor.

We have recommended the dual-core 3.0Ghz server platform to several large corporations with more than 10,000 estimated users, becase without a doubt, the platform will be able to meet the throughput requirements for some time to come. It might be slightly odd coming from employees of a well-known server vendor, but in our opinion there is absolutely nothing wrong with starting up small and growing the environment as the requirements and the usage grow. The only recommendation we do suggest is to choose to separate SQL Server from the WSS server for easy future growth and a move to a highly available platform.

So why would you even consider selecting a topology with more than one front-end server? The answer can be summed up with one word: high-availability.

10.4.4 High Availability Considerations

Without a doubt, for most organizations the only reason to even consider a farm deployment is not throughput requirements; it's because the environment must be highly available. When that single server is down due to software, hardware, or administrator failure, the environment is down, and it will stay that way until the problem has been addressed.

SharePoint supports load balanced Web front ends, which not only provide increased throughput but, also fault tolerance in case one of the front ends is no longer responding. Both the hardware and the software-based network load balancing service automatically direct all traffic to a working server in case one of the servers is not responding within a predefined time period. While network load balancing determines the status of the nodes participating in the cluster via heart beats, hardware-based solutions can be configured to examine the HTTP result codes for lower level fault tolerance.

In addition to load-balanced and fault-tolerant Web front ends, SharePoint also supports SQL clusters for back end database fault tolerance. You

can install SQL Server in single- and multi-instance modes, which previously used to be referred to as active/passive and active/active configurations. Essentially, in a multi-instance SQL cluster both of the nodes run separate SQL Server instances, which can have any number of databases configured, thus effectively allowing you to manually load-balance the SQL Server load. Some SQL DBAs do not like to run SQL in multiple instance mode, because it gives you a false sense of capacity. When one of the SQL cluster nodes needs to fail over for one reason or another, the second node must have enough capacity to take on the load of the failed over instance. For example, if both nodes were running at 60 percent capacity, there would suddenly be 120 percent capacity requirements put on the single cluster node. Thus, most people prefer to run single instance SQL, as at least you can be sure the second node can take on the load of the server failing over. SQL Server also supports N+1 clusters, which are effectively multi-instance SQL clusters with an extra standby node ready to take over the load of a failing instance. N+1 clusters are a good way to reduce cost for larger SQL deployments, since the standby server can be shared between two active SQL Servers. Of course, if two servers fail in an N+1 cluster…

10.4.5 Network Impact

All the figures we've noted before have been tested in datacenter network conditions, i.e., 1,000Mbit switched LAN. HP's workload, utilized on the 2-socket dual-core 3.0Ghz platform, an average of 9,000 Kilobits per second total network traffic for the simulated users on full 126 RPS throughput. This throughput translates to an average of approximately 71.5 Kilobits per operation. As you can see, the network throughput requirements exceed that of the typical effective speed of a 10Mbit network, thus definitely requiring at least the 100Mbit back end network connection. It needs to be pointed out, though, that the bandwidth requirements vary based on file sizes and frequency of updates. The figures here are generalizations, which have been found to be accurate in one particular deployment scenario but might not fit in every environment.

You should work with your network team closely to ensure that adequate network capacity is at hand for your estimated throughput requirements and collaborate with them in an ongoing basis to adapt the estimates based on your deployment workload characteristics.

Even though the tested network capacity was not close to saturating the 1000Mbit datacenter network, separate network cards were utilized for back-end connectivity. It's not uncommon for today's servers to include at least two embedded NICs, thus allowing you to dedicate one of them to the back-end SQL- and network load balancing-related traffic, not to mention backups.

HP's observations during testing at about 100 requests/second throughput rate showed the following average network traffic characteristics:

- Traffic to clients: 9000KB/sec
- WFE traffic (read/write): 11000KB/sec receive, 20000KB/sec send
- WFE traffic (read only): 5000KB/sec receive, 20000KB/sec send
- SQL traffic (read/write): 4000KB/sec receive, 10,000KB/sec send
- SQL traffic (read only): 2000KB/sec receive, 10000KB/sec send

If you are going to deploy SharePoint in a datacenter with a 100MBit backbone, we recommend you to create two virtual networks in order to separate client traffic from the internal SQL and search traffic to ensure no network throughput bottlenecks occur.

10.5 Disk Subsystem Planning

SharePoint 2007 disk subsystem planning is slightly trickier than most applications, due to the database disaster recovery options and various content indexes generated outside the database. Especially in a farm configuration, you should carefully plan the I/O capacity requirements ensure ample space for future growth. Luckily with today's copious storage technology options, which have brought down the per-gigabyte cost of storage, you have options when deciding upon your storage platform.

Shared Storage Area Networks (SANs) are pretty much the norm in many enterprise datacenters, providing cost savings and easy upgrade paths. Since SQL fully supports SAN storage, you can take advantage of this technology across your SharePoint deployment. Still, even with a SAN you have to define a slice from the shared storage pie that should be allocated to you in the beginning. If you're choosing to use directly attached storage instead, it's even more important to consider all future requirements beforehand, since adding capacity might not be as straightforward on a directly attached storage device.

The hard part is trying to come up with an accurate estimate of the "content stored" part for an enterprise with thousands of users. Personally, I've been bitten by this in the past and it appears I did not learn my lesson. Thus, from bitter experience I recommend that you think of the worst possible scenario and **double** it. That just might get you through the first 12 months and establish more accurate storage growth patterns for your enterprise.

10.5.1 Database Sizing

First and foremost, database sizing should be your primary concern. Although it's easy to add additional data files to SQL Server, you should try to anticipate usage well in advance, since sometimes the order process might take a while. With directly attached storage devices, you might also need to plan for system downtime unless the devices support adding capacity online, which SANs at least typically do.

Aside from the standard 1.5GB per farm configuration database, you need to estimate the initial size of data moving to the SharePoint deployment and count in the estimated number of new sites, keeping in mind the site quota limits, to get a grasp on the required disk space for your deployments SQL databases. Once you come up with the magic figure, you need to multiple the total content storage size by 1.3 to get the item storage size in SQL databases, due to additional item properties and associated overhead. In addition, it is generally recommended to forecast a healthy growth plan of at least 50 percent of the initial projected storage so that you do not run out of disk space right after go-live. We've seen many organizations discount the almost viral-like growth team collaboration sites seem to exhibit in most organizations, at the cost of possibly painful storage upgrades. For example, in HP we had to build a whole new farm to facilitate SharePoint 2003 storage growth, because the original projected growth was far surpassed by the massive upsurge in team collaboration site usage. At the time, the SQL team recommended against SAN storage and insisted on directly attached storage enclosures instead, which have a limited growth span. Safe to say, we learned our lesson in time for the SharePoint 2007 deployment.

The projected content storage requirements will help you plan and implement your content database structure, since Microsoft best practices recommend a 100GB limit for content databases. With HP's SharePoint deployment, for example, closing in on the 6TB mark at the time of writing, planning for creating and maintaining these content databases was a top priority at design phase. For each of the 100GB content databases, you should also project at least 35GB for the transaction log.

For further details on the art of physical database storage design, you should refer to the Microsoft TechNet article, located at http://www.microsoft.com/technet/prodtechnol/sql/2005/physdbstor.mspx.

10.5.2 Search Index Sizing

In our experience you should expect index to be about 30 percent of the total indexed content size. However, the exact size of the index corpus is largely dependent on the content types you are going to be indexing. However, it pays to be extra safe, as the content index databases cannot be spread across

volumes. Every content index is its own database, stored in a location that can be changed with command line tools, but as such is slightly tricky to execute. You should instead follow a generous guideline and reserve about 50 percent of the content indexed for the index corpus.

If you're used to sizing index servers for the previous version of SharePoint, you'll be glad to know that SharePoint index propagation now operates on a continuous basis, as we've mentioned earlier on in the book. Therefore, you no longer need to reserve double the size of the corpus on the search servers.

10.6 Global Deployment Considerations

We live in an exceedingly globalized world, so it's not unusual for most of us to work in distributed work environments, emphasizing the need for good collaboration. For example, I'm writing this chapter in Ireland, while Kevin is going to review it in Scotland and pass it to Emer in California. To work efficiently, we must use the collaboration tools to work across time zones and great distances.

It's not surprising, therefore, that most SharePoint deployments must consider how to best serve the collaboration requirements of the global workforce. This might include simply hosting the services in a centralized network hub and ensuring that the latency and bandwidth meet the needs for the remote users. However, exceedingly today we must also consider the language requirements a global deployment dictates.

10.6.1 Addressing Multiple Languages

If you are considering hosting services for multiple regions from a centralized deployment, you will likely have to provide regionalized services for different countries. Regionalizing SharePoint deployment ranges from time zone and character set modifications all the way to fully translated site templates.

WSS allows any number of the languages supported by Office to be installed on the same server with downloadable language packs, which must installed on each of the WFE servers in your deployment. In addition to the language packs that come with the necessary translated SharePoint templates, you need to configure the underlying operating system with the necessary language files, such as keyboard files, fonts, and Input Method Editors (IMEs).

MOSS 2007 multilingual support has been vastly improved from the previous (frankly dismal) story in SharePoint Portal Server 2003, which only supported one language per installation. Now MOSS actually adds, on top of all the supported multilingual features of WSS, extra features and templates.

Given the merge of content management server features into the SharePoint Server family, content publishing on SharePoint has been enhanced with Variations support, allows users to create multilanguage versions of a master page source, not to mention build different display templates, for example, targeted at mobile device readers. It should be noted that Variations is only available in publishing sites and therefore does not replace language packs for collaboration sites.

10.6.2 Distributed or Regionalized Server Deployment

In a global deployment, one of the fundamental questions is always how to best serve the regional users. From an IT management and cost perspective, a centralized deployment is often the favored option, since you can deploy one large farm to be able to scale to any enterprise requirements. The placing of that one central farm, though, often becomes a company political issue, especially if there is a strong presence in multiple continents. Placing the hardware near the largest user base might not always be wise if most of the content readers, for example, are based on another continent. Of course, deploying hardware at each of the major sites would resolve any disputes and provide users with fast connectivity all around. However, if high availability is a requirement for these environments, it quickly becomes prohibitively expensive to put up farms in multiple sites.

There is no silver bullet for regional deployment server placement planning, but we can offer two major approaches we've been involved with and the pros and cons of both.

A valid approach, both technically and from an information architecture perspective, is to go with a centralized deployment of SharePoint, especially now that SharePoint features extensive support for multilingual deployments. With a centralized deployment, you don't have to worry about replicating content or users going to the wrong URL; you have one place to bookmark and you can find everything from that one central location. The IT staff doesn't have to worry about supporting multiple platforms and the infrastructure they rely on.

Unfortunately, the problem with centralized deployments is speed for remote users. WSS typically requires good network connectivity, since by the nature of the application you are actively working on information documents and presentations, which today can be rather large in file size. While, as a Web application, SharePoint is not as dependent on network latency, there's not a lot one can do about uploading a 4Mb file to a team site. Document management is slow over a low bandwidth WAN link, and there's not much you can do about that. Actually, that's not exactly true. If you have the ability to do so, you can request remote links to be upgraded to a certain standard latency and bandwidth, thus making the centralized server deployment a

serious architecture contender. Obviously, another issue with centralized deployments is that you're effectively "placing all your eggs in one basket," thus placing demands on your business continuity planning. Business continuity planning should detail the plan to continue operations in the shortest available time when a disaster strikes and the primary server farm cannot be quickly brought back online. Data center fires are not unheard of, and other natural and manmade disasters are also possible. If you must be able to continue operations in these exceptional circumstances, you must plan for a disaster recovery environment in another datacenter.

The second option is to deploy regional instances in the user population centers across the enterprise. While technically this is an easy fix, it does cause headaches for the knowledge management and information architecture planners. By distributing sites across servers, you are effectively creating information silos in these regional environments. Operations overhead is also increased, because there are now more servers to look over. The cost is likely higher, given the extra hardware and operational requirements, especially if high availability is still a requirement in these regional instances.

While regionalized deployments will continue to be a valid option, the recent industry drive for datacenter consolidation is likely to ensure that centralized deployments will be the flavor for SharePoint 2007. For example, the internal HP deployment for SharePoint 2003 spanned 6 farms across all regions in an effort to balance network performance with centralized operations model. The deployment architecture was largely driven by the network topology and designed to allow best possible end-user performance in the company's network framework. However, with the new drive for data center consolidation, the architecture was collapsed to a centralized deployment in SharePoint 2007. Given the new drive, the program was able to dictate the required network topology that would support SharePoint deployment, instead of the other way around back in 2003.

10.7 Preparing for Software Deployment

Before installing the products, you should take a moment to review some upfront tasks that will help streamline the deployment process and save you time and effort. Because certain infrastructure configuration changes, such as adding DNS aliases and pointers, can take a while to synchronize across the organization, it's good to plan and execute them in advance.

10.7.1 Service Accounts

Service accounts are straightforward domain user identities. However, to reduce operational overhead, they are often configured with different security policies than normal user accounts. Typically, service accounts are set to

never expire and sometimes created without interactive logon rights. While SharePoint does not require anything special from the domain service accounts, I would recommend setting the service accounts to expire in longest intervals your network security team allows. Changing the service account password does require system downtime, thus limiting the change execution to occur out of hours or during weekends.

WSS 3.0 and MOSS 2007 offer a multitude of possibilities for segregating different components to run under their own individual accounts, even down to the Web application level. Thus, it's very easy to over-architect the service account model for a deployment, causing overhead account management down the line. While certain segregation is certainly advisable to reduce downtime caused by account lockouts, for example, it's good to find a middle ground between over- and under-architecting the service account model.

At minimum, you will need to create at least one SSP service account regardless of which SharePoint product you are deploying. The question is, how many other accounts are required for your deployment scenario, although you can choose to use a single service across resources. However, by separating different components to run under separate identities, you can mitigate the impact of an account lockout and security impact, in case the service account is compromised.

Although in some scenarios running IIS application pools under different accounts might mitigate security risks, for typical intranet setups you should consider the security compartmentalization benefits against added management overhead for both SharePoint and SQL.

For SharePoint search, you should seriously consider using a dedicated default content access account to reduce the risk of compromising the service account during indexing operations. However, if you trust the resources you will be indexing (or if you will only index local content) it is possible to reuse the same service account as for other services. If you plan to index a lot of external content sources in addition to local SharePoint Sites, you should consider using a separate content access account for external content sources to prevent an account with administrative access from falling into the wrong hands.

After choosing the right balance of convenience vs. security compartmentalization for your deployment, you should document the accounts and their purpose and secure the passwords accordingly. In a disaster recovery scenario, you do not want to lose time by having to hunt around for accounts and passwords.

For more information on planning for security and service accounts, please refer to the Microsoft guide located at: http://office.microsoft.com/download/afile.aspx?AssetID=AM101638561033.

10.7.2 Preparing Your DNS Architecture

While SharePoint fully supports WINS name resolution, Fully Qualified Domain Names (FQDN) are usually employed to ensure everyone can access a service in an enterprise deployment. Since DNS changes usually take a while to replicate and process depending on an organization's strategy, it's good to assign the DNS names well in advance and use them prior to installing SharePoint.

Most organizations have standards for naming DNS pointers, based on the region or business unit the servers belong to. While, for example, dubwspro001.emea.company.com is nice for IT organizations, they can quickly deduct that the server is a production (pro) Web server (ws) node (001) located in Dublin (dub), it is unlikely that end users will have a clue as to how domain naming schemes work. Luckily, DNS architecture supports domain name aliases, which can point to existing records and provide much more user-friendly names for business applications. For example, a DNS alias for the above server could be teams.company.com, which end users would certainly remember more easily than the complicated IT naming scheme.

There are three ways to create more than one virtual servers on a single server (or farm). You can assign each virtual server to operate in a specific TCP/IP port, such as port 88. Unfortunately, this method is the least obvious for end users, as they will need to append the port to the url (http://server.com:88) when accessing the virtual servers; thus I would not recommend using it unless it is the only available option.

Typically, most organizations elect to add additional IP addresses to the Web server, configuring each virtual server to use a dedicated IP address. This approach offers all the functionalities in IIS without compromises, but obviously adds a management overhead, as you have to worry about creating DNS pointers/aliases for various IP addresses and assigning them all to the servers. In a farmed environment, you will quickly have an IP address management problem, since, for example, two front ends would require a total of three IP addresses for each virtual server configured (one per front end and one for the load balanced address).

Another option is to utilize host headers, which allow several virtual servers to operate using the same IP address, as seen in Figure 10.1. Host headers are essentially DNS aliases that resolve to the same IP address. However, since every browser provides the requested host name as part of the HTTP query, Windows is able to detect which virtual server the query is intended for and queue it up accordingly. Sounds perfect; minimal management overhead with pleasing URLs for end users! Unfortunately, as you might guess, there's a catch. Due to the way SSL-encrypted requests are processed, IIS cannot read the encrypted host headers and thus is unable to queue the requests to the correct virtual servers. SSL-encrypted requests are

always forwarded to the default virtual server, which means that you can **only** provide SSL encryption for one virtual server with SharePoint products and technologies when using host headers.

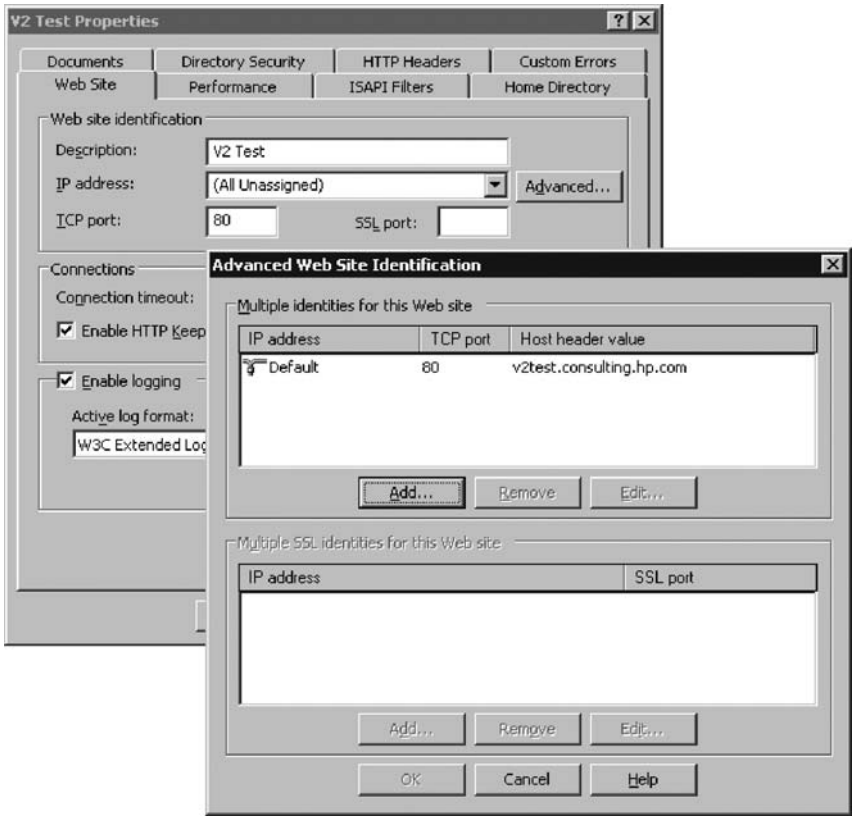


Figure 10.1
Several virtual servers can be run using host headers.

10.7.3 SQL Server Configuration

Most database administrators prefer to run SQL Server with a nonstandard TCP/IP port. For example, the Slammer SQL worm, which affected numerous systems around the world, exploited the fact that most SQL setups used the standard 1434 UDP and 1433 TCP ports. Thus, it's good to make life a little harder for malicious users and viruses by taking steps to at least slow down attempts of exploiting services remotely. On the SQL 2000 Server you can change the SQL ports by opening the server network utility and changing the default port 1433 to something else as long as the port is not reserved by another application running on the server.

If you are planning to use a nonstandard TCP/IP port for SQL, SharePoint requires the SQL client options to be configured prior to installation.

For a guide and good rundown on other SQL Server–related security hardening steps, please refer to the Microsoft guide located at: http://office.microsoft.com/download/afile.aspx?AssetID=AM101638561033.

10.7.4 Quota Management

As file sizes get larger and larger, the storage requirements of collaborative spaces are also rocketing skyward. Anyone who has been involved in trying to restrict file share or exchange public folder growth is no doubt painfully aware that most users seem to treat these central resources as personal filing cabinets for a myriad of files, some dating back more than 6 or 7 years. Digital audio and video files certainly occupy a fair amount of space, but in most of the cases I've been involved with, the largest piece of the hard disk pie is still taken over by Microsoft Office files. Nowadays, it's not uncommon to see 6- to 7-megabyte Microsoft PowerPoint presentations, and Microsoft Word files also seem to be 3- to 4-megabytes and up, largely due to embedded image content. At the same time, the size of personal computer hard disks has multiplied to keep up with the pace, and the datacenter has seen the introduction of Storage Area Network (SAN) devices that provide shared disk space on demand for servers at more cost-effective prices. However, it's good to look at the past lessons learned and take steps to ensure that adding disk space is not the only answer to the growing storage problem.

Windows SharePoint Services introduces a proactive method of restricting storage growth beyond the boundaries of designed architecture in the form of site collection quota limits. As we discussed earlier in the chapter, site collection use confirmations can ensure that old information doesn't use valuable resources other active spaces could benefit from, but it is still very important to restrict the few eager adopters from using an unfair chunk of the resources for their spaces.

We always recommend setting the quota limits to as small as possible while taking into account the usage scenario. We know this is a somewhat strange recommendation coming from employees of a company that manufactures a large array (no pun intended) of storage solutions to fit any need under the sun. Nevertheless, quota limits of just 250MB for team sites and 150MB for My Sites have been well received by users in many large enterprises, after some initial rumblings from both user and management sides of the table.

10.8 Planning Backup and Restore

Part of your overall deployment planning exercise should be considerations for disaster recovery, especially since all of the out-of-the-box options require a significant amount of storage space. The backup and restore plan is not only important to prepare for disasters such as database corruption. Most of the lost data is typically deleted by end users, albeit accidentally. Luckily, WSS 3.0 now supports a multistage recycle bin, allowing both users as well as site administrators to recover accidentally deleted files, thus effectively removing 80 percent of typical restore operations in comparison to previous version.

Unfortunately, we must plan for exceptional circumstances when planning our deployment architectures. What if there is a fire in the building hosting the servers, and all of the servers are burned to the ground? Most organizations require a business continuity plan for such exceptional circumstances continue operations as soon as possible in another location.

10.8.1 Out-of-the-Box Backup Options

Out-of-the-box SharePoint ships with tools for both products that allow you to take online backups to disk of production portals and WSS site collections. Of course, you can also use all of the disaster recovery options provided by SQL Server, except of course for SharePoint search indexes, which are stored in another database engine. SharePoint 2007 has combined backup and restore into the central administration backup tool accessible from the web, thanks to feedback received from SharePoint 2003 users.

With the new Recycle Bin functionality provided by the core platform in SharePoint 2007, the lack of granularity in the out of the box backup tools is not such a big issue as it was in the previous version of SharePoint. However, Microsoft has still limited the central administration backup and restore tool to only allow content database backup/restore at the lowest level. If you wish to perform site or subsite level backup/restore, you will need to look into one of the 3rd party products that provide further granularity. Alternately, you could chose to not back up the index at all and accept an increased recovery period for the search component as the content is being re-indexed from scratch.

10.8.2 Third-Party Backup Solutions

Of course, there are a number of third-party solutions from various providers, which offer per-site and even per-document back up/restore capabilities for a cost. The first one to market in 2001 was AvePoint (www.avepoint.com), with a product called Doc Ave that offered both site- and document-level

backup and restore capabilities for SharePoint 2003. Today, there are also a number of other products available from vendors, such as Veritas Backup Exec. Keep an eye on the Microsoft Office System Solutions Directory for updates on Backup, as well as other add-on products for SharePoint at http://directory.partners.extranet.microsoft.com/.

10.9 Planning Data Migrations

Even though migration from a legacy platform might not be part of your deployment plan, it is highly likely you will be faced with migrations sooner or later, be it migrations within your SharePoint deployment or from other platforms. After all, maintaining and supporting disparate knowledge environments is not only costly from an operational and support perspective, but it also creates knowledge silos that prevent effective sharing of knowledge, experiences, and best practices within an enterprise.

Although one might consider migrations as mainly taking data from System A to System B, a much more likely and common migration scenario is to actually move data within the same product. Having separate development, staging, and production environments is common place in any enterprise application deployment and often moving data between these environments is part of a production deployment cycle. In regionalized deployments, moving production data is commonplace, due to changing network conditions and users changing locations, not to mention consolidating infrastructures to a central farmed deployment from previous single server configurations. More far-reaching changes, such as mergers or divestitures, can also prompt the consolidation or separation of data.

10.9.1 When Company Structures Change

Many of us have been involved in bringing the IT landscape of an enterprise to a steady harmony after a merger, but we suspect few of us have also been involved with figuring out how to separate enterprise systems after a divestiture. From experience, we can vouch that these mergers can even occur while deploying a technology only to find that your new colleagues are in the process of doing the same. While neither task is without challenges for an enterprise deployment of SharePoint, a merger is typically more straightforward to plan out and execute than a divestiture.

Typically, the first step to providing interoperability during a merger is to ensure that the two domain infrastructures trust each other and that network connectivity exists between the two companies. When HP and Compaq underwent the largest IT merger in history, we talked about "Day 1" interoperability, which included both network bridges and domain trusts. Thus on Day 1 of the merger, both company's SharePoint environments

were fully accessible by all employees. A few days later, the new color schemes had also been applied, and both environments at least looked identical to end users.

Divestitures are slightly trickier, as they can include situations where an existing SharePoint deployment needs to be broken up and also often include domain account migrations for at least one of the new companies. As with consolidations, data itself can be moved from server to server using one of the migrations methods we discussed earlier. Additionally, the migrated spaces and data belonging to the divested entity need to be cleaned up from the "old" environment after the data separation, but this can easily be accomplished by scripting the stsadm tool or via custom application using the object model. However, authentication will prove more challenging, since this time around trusts are being broken, not forged.

Typically in a divestiture, one of the new entities will create a new domain infrastructure for the employees and will recreate or migrate accounts to that domain infrastructure. A straightforward back up/restore method of moving data will therefore result in portals and sites being inaccessible by these new accounts. Thus, a custom tool to map accounts between old and new identities needs to be run after the data migration to enable access once again and remove identities belonging to employees of the divested entity.

10.9.2 Preparing for Cross-Platform Migrations

While migrations within the SharePoint deployments are commonplace, when approaching a new deployment, it's more likely to be given an opportunity to consolidate other technologies to the new SharePoint environment. Given today's feature-rich environments and the multitude of possibilities for deploying and structuring those environments, there's no silver bullet for migrations. Approaching migration projects requires extensive planning and careful study of the existing environment.

We have been involved in discussing, planning, and even on one occasion executing migrations from enterprise implementations of LiveLink, eRoom, and Windows File Shares, and the basic approach to these projects has been similar. In a nutshell, you must map out and address the functionality differences, decide where to move the data, agree how the migrated data will be structured and find a way to map the user identities. Unfortunately, the best-laid plans cannot cover every eventuality, so you should also be prepared to adapt along the way.

Users often hate change; just as they've gotten comfortable using a system, the rug is yanked from beneath them and they must learn a new way of working. We technologists often forget that a typical business user doesn't usually care about what product they use, as long as it does what they need it

to do. Users are too busy to learn about new cool functionality, and rightly so—it's not their job. Trying to justify migrations with "but you'll get this, this, and this new functionality" is often a lost battle, since the disturbance to business is not only in downtime during migrations (careful planning can minimize downtime but it is unlikely that you will achieve zero downtime during any migration!), but undoubtedly in having to learn the new technology and more importantly trying to apply past experience with the "old" way of working to the new system.

Migration planning is not only about the tools and technology; it's about making the experience as seamless to the users as possible. I've often heard users still refer to the new SharePoint environment they've been migrated to from whichever legacy product by using the old product's name, and I left such conversations without comments or corrections. Personally, the less change people notice, the better, and thus that is one of the goals I like to apply when planning any migrations.

10.9.3 Addressing Functionality Changes

User interface and the functionality that is presented by that interface are without a doubt the first things users will notice after migration; on my list, they are the highest priority items to address early on. While user experience can be changed with customizations to the user interface, which we cover in Part II of the book, addressing functionality gaps between two products is much harder.

Unless you are prepared to address gaps by developing additional functionality on the platform, your only other option is to recognize the differences and advise users how to deal with the loss of functionality. However, you must first know what the functionality gaps are and then find out how important those features are to the end users. Therefore, perhaps the first task in any migration project should be to create a functionality matrix for the two products. Unfortunately, this is often much harder than it first appears, and most matrixes we've seen have been biased towards one product or the other, depending on the personal opinion of the author. We've compiled such biased lists ourselves as well, not intentionally mind you, but we've seen too many to honestly say that such lists should not be a part of the process when deciding which product should be deployed. They do, however, serve a useful purpose after the decision has been made and migration project is underway.

By creating a functionality matrix, you will also learn a great deal about the product you're deploying, since you'll likely have to investigate features such as security and document management capabilities in depth. Personally, although we have deployed and managed several servers in various stages of the SharePoint Beta cycle, we still picked up new tricks and features of the

product while mapping out the differences between SharePoint and LiveLink. It's like Microsoft Word—how many of us can honestly say we know exactly what the product can do? But if we were to get 1,000 users to the same room and ask everyone to write down what features they use, I'm sure we'd get quite a comprehensive list together by merging the answers. Thus, it is very important to spend the time and effort to list both products' functionality and identify the gaps and impact to the users.

We like to pass on the functionality gap list to a few power users and ask them to rate each one with "Must have, nice to have or don't care" labels, since let's be honest—none of us knows exactly what the users consider important. On the other hand, a business is not a democracy, and in the end the buck stops with the business sponsor. So while we can map out the gaps and ask users to rate them, it doesn't mean they will be addressed, due to time, effort, and cost to the overall project.

10.10 Planning Support for SharePoint

It's important to decide a clear support strategy at the beginning, as there's nothing worse than having to "play as it goes" when dealing with issues. Be clear about what you support and who is in charge of what, and finally, make sure the end users know the support policy and how to get in contact. I'll discuss some of the more important topics for support strategy, such as operational model and Service Level Agreements (SLA) shortly, but first let's discuss some of the higher level points you should consider.

10.10.1 Planning a Support Strategy

First of all, let us emphasize the importance of a staging process, or in other words making sure that all changes go through a testing environment that mirrors the production setup. While this might be a moot point, we have seen deployments that did not include this, due to budgetary reasons. We've also seen these deployments pay dearly later on, with downtime due to various operating system changes that changed the environment in an unforeseeable manner, causing the application to fail. You don't have to mirror the production environment exactly; it's enough to have a similar configuration without all the latest and greatest hardware. Note, though, that when the staging environment isn't a perfect match, there can be situations, such as updating device drivers, that you will have to perform without prior testing. Thus, we recommend getting as close as you can to the production environment if high availability is absolutely key for your deployment (see Figure 10.2).

Although "what do you support?" might sound like a simple question to answer, you should consider this carefully for your SharePoint deployment. Even the out-of-the-box capabilities are extensive, putting burden on the

Figure 10.2
Staging process is key for high-availability enterprise applications.

support organization to learn everything to be able to support it. If you add on top of that the customizations possible with SharePoint Designer, your task just got a whole lot more complicated. WSS and MOSS are not only applications, they are a technology framework to build line of business applications upon as well, so supporting absolutely everything will become a daunting task.

We recommend that support organizations draw the line limiting support to customizations that are possible with Internet Explorer and come up with a model for custom Web Parts and applications that puts most of the support tasks on the developers. Supporting SharePoint Designer customizations is too wide of a field to tackle; the training to operations staff alone would need to be extensive. We are not recommending that SharePoint Designer modifications should be banned altogether, but we recommend deciding on a strategy that will not offer support defined in the SLAs for SharePoint Designer–modified Webs. This can be a controversial view, and has been widely debated in the Web Logs and community forums around the Internet, but we've seen the support calls generated and would never sign off on a support strategy for our organization that included SLA for SharePoint Designer customizations. We have recommended and implemented support strategies, which included end-user education and clear statements that SharePoint Designer–modified Webs are allowed, but only supported on a best-effort basis.

Custom Web Parts and other applications can become an issue for operations staff, unless the new Web Part deployment process includes steps to ensure that the staff is informed of the customizations and aware of their intended functionality. Since it is unlikely that the operations staff possess developer skills, you should ensure that the developers accept responsibility for their work and commit to ongoing support of the customizations. While developers might change, properly documented code can be modified and supported by their replacements. Therefore, one key point for support strategy is to take steps to ensure the code is properly documented instead of relying on developers to do this out of the kindness of their hearts.

We've seen operational model shown in Figure 10.3 used for Web Parts and other custom code successfully on several large enterprise deployments of SharePoint. While it can at first look quite complicated (although the

drawing has already been significantly generalized from the "live" versions) the basic thinking behind the process is straightforward. As with any development, the requirements must first be defined and documented, based on which the application is designed and developed. Ensuring that the requirements are documented up front helps the operations team support the end result better, as they understand what the usage scenario is. Badly coded Web Parts can, in the worst possible scenario, bring the environment to a halt, and while this is extremely unlikely, I would still recommend making sure that the functional testing phase of the custom code deployment process included a performance benchmark. Lastly, once the code has been verified and modified as necessary, all changes should go through a staging process to ensure that the production environment does not introduce any surprises.

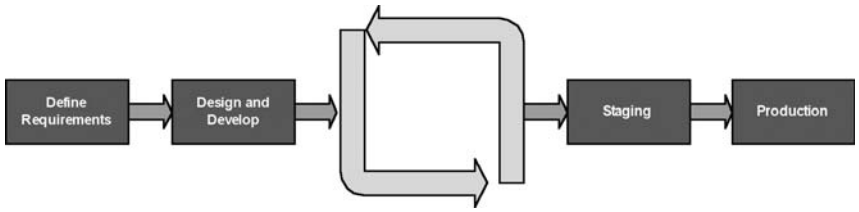


Figure 10.3 *Custom development process.*

One often-forgotten important point to remember when developing a support strategy is to properly document it once it is done and to make it available to the end-users. While most end-users will likely not care to know or read the documents, key business process owners will benefit from the gained insight and make use of the documentation effort. Of course, you can always point people to the documentation when complaints are voiced on some of the more controversial points, for example, the SharePoint Designer modifications we mentioned earlier. A condensed support policy presentation should also be compiled for users who do not care to wade through the full documentation.

10.10.2 Planning an Operational Model

Most organizations have well established and preferred operational models in place for various applications, such as Exchange. Thus, choosing the operational model will likely be just an exercise to find out your organization's preferred model and adopting it. However, if there are no established methods in your organization, you might need to spare some thought for this point as well.

Typically, in an enterprise environment with shared infrastructure, the application support team relies on other teams to perform at least the Hardware, power, and network support and operations. It's natural that the support for these shared infrastructure and physical components is delivered

from a centralized resource, since the application support team might not be on the site at all the datacenters where SharePoint has been deployed. While this introduces reliance to other teams and their ability to meet service level agreements, the benefit is reduced cost, especially if you are considering a geographically distributed deployment. On the other hand, today it would be completely possible for a remote support team to support a remote deployment, given that most high-end servers ship with an integrated remote control hardware that allows support to perform everything they could from a local console, including powering the server on/off. You would still have to contract someone to perform hardware replacements and other tasks that require physical presence.

Today, more often than not, Operating System (OS) is considered as part of the underlying shared infrastructure and support, for it is also delivered by a dedicated team. Given the frequency of operating system security updates today, sourcing OS support externally certainly reduces the management overhead for the application support team. IT organizations might in fact require this, in order to have the operating environment managed in a secure and accountable model.

For the actual application support, most enterprise applications run tiered operational model, especially if 24/7 operations capability is required. The tiered model established a number of operations teams with varying levels of knowledge and sometimes access to the systems. Tiered operational models often also include an escalation step outside of the operations team to a subject matter expert or external consulting provider. For example, in the four-tiered operations model shown in Figure 10.4, tier one through three are "in house," while the fourth-tier can be reached via an escalation to the subject matter experts.

In the four-tier model, the first-tier is really the helpdesk. Helpdesk usually is delivered out of a lower cost center, which handles a number of technologies and thus is not expert in any given application. The first-tier helpdesk typically gathers information from the end user and checks it against the known issues and resolutions list to attempt to resolve the case immediately. However, if the error does not sound familiar, the first-tier will transfer the case to the second-tier helpdesk, which is often run from the same low cost center. To deal with these filtered calls, the second-tier helpdesk staff has usually been trained to understand the application quite well and is thus able to troubleshoot the issues with the end users. If the issue cannot be solved by the second-tier helpdesk, they transfer the case to the third-tier who typically double as the application administrators for the environment. The third-tier staff are application experts, as they are in charge of the day-to-day administration and change management. If the issue is too complex for the third-tier staff to solve, they can escalate it to the subject matter experts—who might be you if you're planning to deploy SharePoint in your

organization. The fourth-tier is typically comprised of architects and other key knowledgeable staff within the organization. fourth-tier typically also works with the Microsoft Product Support Services to troubleshoot and resolve serious issues that might require a hotfix.

Obviously, the model shown in Figure 10.4 can be adapted to your organizations by dropping the levels that are not required. It would be advisable to have at least two levels, so that the application administrators do not need to spend their time helping with trivial cases when they could be doing something much more productive.

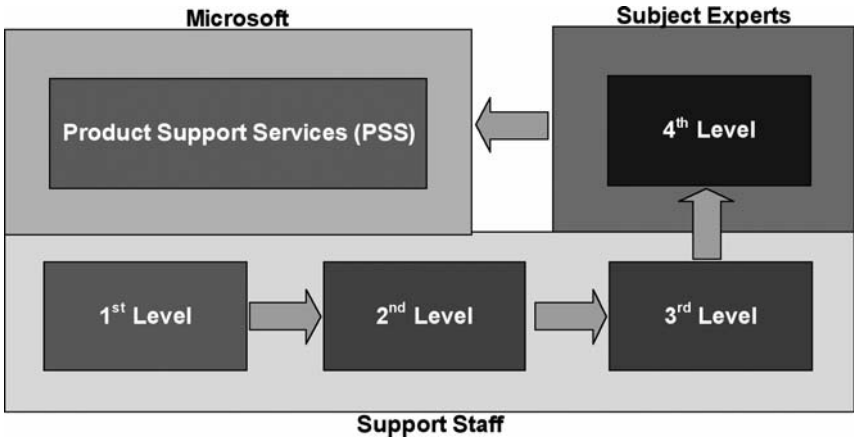


Figure 10.4
Tiered operational model.

10.10.3 Planning Service Level Agreements

The Service Level Agreement (SLA) is the key contract signed by the business and the operations team that defines the support services delivered for an application. SLA is not only a percentile number for availability (such as 99.8 percent) although it certainly includes the agreed upon availability goal as well. An SLA should cover the whole support strategy, expectations, and metrics for the environment. SLAs should address at least following key points:

- The specific services provided
- Roles and responsibilities
- Priority service levels and handling expectations for different levels
- Communication and escalation paths
- How the SLA is measured and reported

Most support organizations prioritize cases and thus commit to different response times based on issue priority. For example, system downtime would no doubt be a Priority 1 issue that should be dealt with immediately, a user asking how to upload a batch of files at the same time will no doubt be put towards the back of the queue and handled when more critical issues are out of the way. In the SLA, you should clearly define the different priority levels and what lead time the end users should expect when there are more important issues to deal with. Table 10.8 shows an example prioritization breakup with descriptions of issue severity and indications of how soon the users should expect someone to get back to them on the issue they logged.

Table 10.8 *Example Prioritization Level*

Issue severity	Description	1st Response
Priority 1 (Critical)	Application is down or major business process impact and end users are critically impacted.	15 minutes
Priority 2 (Serious)	Major functions are not working or business processes are impacted; however, end users are not critically impacted, but do notice the issue.	1 hour
Priority 3 (Important)	The application is functioning, but there are issues with some components. Minor nuisance to end users; however, they can continue to use the system.	6 hours
Priority 4 (Low impact)	"How to" questions and requests for more information.	24 hours

Unfortunately, sometimes the defined response times cannot be kept and thus you should clearly define the escalation path users can take to raise the priority of their case. Escalations should go through a function in the business to prevent the support organization from being burdened with users requesting escalations just because they think the case is just that much more important. Some kind of filter must be applied to adjust the levels and ensure fair treatment. Same business contacts should also be listed somewhere, so that the end users know who to call when they do not feel they are receiving adequate support for their case. In case of a system downtime or other serious issues, the business contacts should also be kept up to date at all times, so they know how to respond to user queries.

The SLA should also include a definition of the roles and responsibilities of all entities involved in one way or another with the SharePoint deployment. It's good to know who to contact, but it's also important not to knowingly step on someone else's toes. Usually the roles and responsibilities are

defined in a RACI chart, those who are responsible, accountable, consulted and informed (RACI) of different tasks on the environment.

Of course, the SLA should include the all-important commitment to certain availability target. Availability is a percentile number derived from the time period the application has been down in any given time span for example, if your application is down for 15 minutes during a 24-hour time period, your availability was 98.96 percent (*((24*60 – 15 / (24*60)) * 100*). Coming up with the availability target is somewhat of a refined art form, we're told, because you are dealing with a number of unknowns. Typically, you have to calculate the availability target from the hardware, network, power, operating system, and last but not least the application availability commitments to come up with a one single magic number for the business. For example, even if you can guarantee 99.9 percent availability for SharePoint the application, it doesn't matter much if the datacenter can only provide you with a network link rated with 97 percent availability target. Sure, the application is up, but since the end users can't access it you're right out of luck. You should study the full picture carefully before committing to an availability target you'll measured against. As a rough guideline, your availability should be the number of hours per year the least resilient infrastructure has been rated for, minus the planned downtime for the application itself. For example, if your network is rated for 98 percent availability, and you plan to perform 32 hours of application maintenance that requires complete downtime, your total availability will be 97.63 percent.

Finally, you should agree a set of criteria that you will be measured against. Undoubtedly, one of them will be the availability of the environment, but support issue resolution times should also form part of the overall measurement. The "traffic lights" approach for measurement works here, just as it does in number of other situations for indicating how well you've done. There should be thresholds agreed for green, yellow, and red with green obviously being above or at the agreed-upon service levels. The yellow and red are up for decisions though, so you should agree these with the business that is funding the environment and provide them with a monthly summary of support performance, events, and explanations when things did not go as planned, with an action plan to remedy it for the next reporting period.

10.10.4 Expected Support Call Volume

Estimating the support call volume is very hard to generalize; every organization is different, and the culture affects how the users behave. For example in a technology company, users are tech-savvy and are often used to things breaking every now and then; therefore they are unlikely to complain right away. Tech-savvy employees will also typically prefer "self-help" methods, such as a support portal with known issues and FAQs, as well as the Internet with monster search engines such as Google. On the other hand, a company

with a less knowledgeable user base is more likely to use the support helpdesk for smaller issues and thus more likely to call.

Luckily, it's highly unlikely that SharePoint will be the first application deployed within an organization, so you can go to the other application support teams to get a feel for the user base. The desktop support team, for example, is a good gauge for the per-person call rate within your organization. Prepare for at least the same per-user case volume in the beginning, and adjust accordingly as the time goes on; nobody expects you to get it right on the first go.

As a rough guideline, we've seen call volumes of approximately 1 call for every 80 users per day after 3 months in production. Using that guideline, if you have a 20,000-user organization with 40 percent (very high) of the users visiting the environment every day, you should expect 100 calls per day. Obviously, in the beginning, the volume is likely to be higher, as the users are learning to use the tools.

10.10.5 Staffing an Operational Team

Unfortunately, some support managers might interpret the outsourcing of operating system and other infrastructure support as one less skill set the operations staff has to possess. While it is true that the SharePoint operations staff doesn't have to worry about security patching and service pack updates, it would be a mistake to be fooled into thinking that you can run a successful SharePoint operations without having skills in the various infrastructure components upon which SharePoint is built.

We've had numerous discussions with various support managers on the subject, and still maintain that the operations team must be able to quickly analyze the problem and consider outside factors without having to go back to the team responsible for the component. Otherwise, you'll quickly find yourself in a position where the system is down, and the application support staff will simply note that it's not SharePoint's fault and get another team involved. Of course, this takes time, and the application support often has to prove that the fault is indeed with the underlying infrastructure and not the application, thus adding to downtime length.

We highly recommend seriously considering including OS and basic network skill sets in the higher-tier support personnel requirements. Understanding how DNS works, being able to perform light network traffic analysis to make sure that authentication is properly working, and understanding how IIS responds to user queries saves valuable minutes when the application is down without an immediately identifiable cause. While this certainly adds to the support delivery cost, given the additional training and staff's technical level requirements, it all comes down to how important the environment is to the business. Ask the business sponsors if they would be willing to spend

$10,000 more per year on support if they would be able to save 10 hours of application downtime, for example.

Once you understand the operations staff technical level requirements and expected call volume, you can put in plans to staff the team appropriately. Even in a smaller deployment, if application availability is a key requirement you need to ensure extra staff for the operations team to provide adequate coverage for holidays, sick days and so forth. Higher Tier levels can provide assistance for lower tiers, but this does not work the other way around. Thus you should assign extra funds to the higher tier staffing so when required they can cover holidays and sick leaves for the tier 1 helpdesk, for example.

10.10.6 Pathway to Production

One of the best ways to prevent issues is to try to make sure they never get to the production platform in the first place. Having a change management process is key for limiting issues caused by customizations, configuration changes, and system patches. Although we already covered the staging process in the beginning of this chapter, we cannot stress enough its importance with enterprise deployment of SharePoint.

While the staging process does add considerable overhead to the operations, especially if the environment is only serving a couple of thousand users, it's still good to adopt at least some form of change management process for your deployment. If the requirements do not justify the expense of maintaining a complete shadow environment, consider using smaller single server deployments to at least be able to test service packs and other patch upgrade process on a platform on which you can afford to make mistakes.

Complete shadow environments have other benefits aside from being a staging environment; they can also act as a disaster recovery platform and take on the production role in case the production farm has to be reconfigured for some reason. Storage reconfiguration is one such case that many of you might face in the next 12 to 24 months of production, unless you've taken steps to ensure high storage growth possibilities. It's often prohibitedly expensive to scale up the storage infrastructure right away, especially if you're deploying a new service to the enterprise without knowing how popular it will become. Thus SAN storage might be out of bounds and you might be faced with a directly connected storage to SAN/fiber-connected storage migration in the future. Having an environment to fail over to and be able to reconfigure and test the new hardware architecture in peace is a luxury worth fighting for.

Especially when opening cases with Microsoft Product Support Services (PSS), being able to demonstrate a reproducible error condition is worth its weight in gold when dealing with a critical issue. Like all of us, PSS does not

hold the key to unlocking every mystery just by hearing the problem description. They have to debug, test, and capture logs and network traces to solve a difficult case they haven't faced before. Having to do this on a production environment is unnerving at best, so for your support managers' continued good health and the reliability of the delivered service to end- sers, include a staging environment in your organization's budgetary plans.

We will climb down from our soapbox now, hoping we've illustrated the need for seriously considering a change management process that includes a staging environment testing process.

10.10.7 Active Monitoring

It would be naïve to think that all possible issues could be filtered with planning and change management. SharePoint is just as likely to encounter a software or hardware fault that causes system downtime for the end users. There's no way around this; it's going to happen to your environment as well sooner or later.

So as you accept that downtime will occur, you should take proactive steps to catch it as soon as it happens and, in the best-case scenario, deal with it before the end users even notice it. Therefore, having an automated monitoring and alerting infrastructure, that will keep an eye on the application operation 24 hours a day, 7 days a week, and 365 days a year should be a key priority for the operations team.

There are numerous product offerings available from various vendors, from Microsoft Operations Manager (MOM) to SiteScope to HP OpenView, which all provide SharePoint-specific modules that keep an eye on hardware alerts to event log errors and application-specific services, along with the URL monitoring. Some of the tools offer even automated tasks to perform based on the error context—for example, restarting the search service when a database memory fragmentation warning event appears in the application event log. Typically, an organization has an existing license a monitoring product, since it's very likely most organizations already monitor their Exchange and Web server environments. If there is no infrastructure in place however, evaluate the different service offerings and make your decision based on functionality criteria and cost.

The monitoring service should at least be able to poll SharePoint-specific service status from Windows (such as the alert service) and most importantly perform a content match for URL queries. Being able to just report that the service is responding with a HTTP status quote of 200 is by no means enough for SharePoint, since the product now reports with "friendly" error messages when SQL connectivity, for example, is having problems. Thus, if you do not check to be sure that the actual HTML content returned

matches the expected values, many problems will go unnoticed until end users complain.

Remember that SharePoint relies heavily on SQL being available and performing up to standard. Therefore, it's important to include SQL services and databases in the monitoring and ensure that you fully follow SQL management best practices. You should, for example, set up alerts to ensure that databases and transaction logs never run out of space and schedule various management jobs, which keep the databases healthy and at peak performance. Please refer to the extensive documentation in the SQL Online Books for more information on SQL management and monitoring.

Last but not least, you should make sure that the alerts don't simply go to a rarely-checked e-mail inbox. If possible, you should try to put in place an alert management plan that uses different means of communication to ensure that critical alerts are dealt with even at 4:00am on Saturday morning, within a reasonable time period defined (obviously) earlier in the SLA. Use pagers and other devices, since e-mail infrastructure, just like your SharePoint deployment, is subject to service availability.

10.10.8 Passive Analyzing

In addition to monitoring that the service is responding to user queries, you should take steps to proactively seek possible clues for trouble down the line. Periodically analyzing the event logs, for example for warnings and errors, can save you time and effort, as you can deal with the issues before they become serious cases possibly causing system downtime.

In addition to event Logs, you should analyze the IIS log files for usage patterns to be able to better understand the user scenario and adapt the architecture to the way the users are really using the infrastructure. You might discover, for example, that users are performing more searches than you've expected, and you can thus prepare well ahead of time, even before the performance analysis catches the extended resource usage. SQL Server logs should also be periodically analyzed for possible issues with index fragmentation and other database events that might cause problems in the future.

10.11 Performance Monitoring

Of course, the most important passive analysis method is system performance monitoring, which can be set up to continuous log and alert based on predefined thresholds. Most of the monitoring products mentioned earlier also record performance information and thus can report on trends and send alerts. In addition, Windows server ships with perfmon, which gathers and logs data from performance counters (see Figure 10.5). Perfmon also allows alerts to be setup with thresholds, although you must have a program

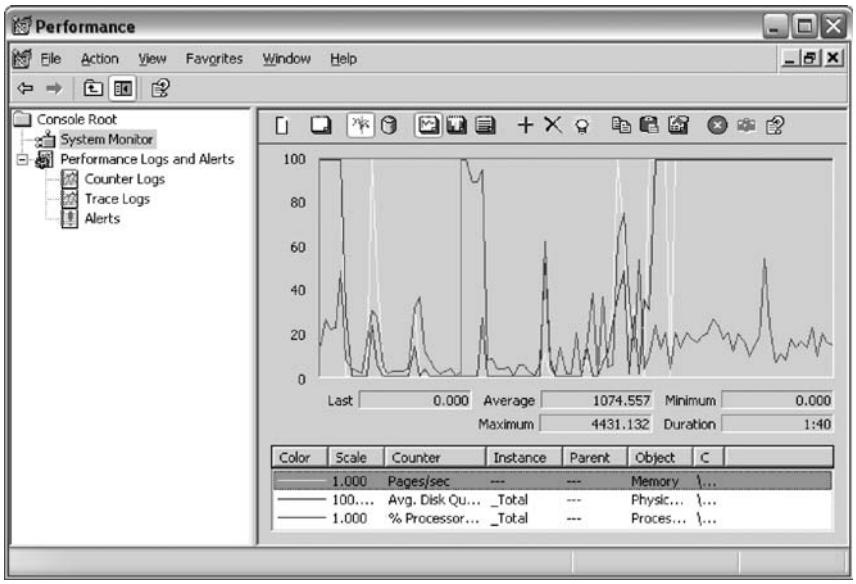


Figure 10.5 *Perfmon performance monitoring tool*

which does something more intelligent than logging the alert to system log to make this feature useful.

10.11.1 Monitoring Strategy

It's important to consider exactly what counters you should capture and how often they need to be recorded. To establish a system baseline, for example, you will need sampling data over several days to be able to draw conclusions. With hundreds of counters available for both system and application components, it doesn't make sense to capture everything, thus allocating a lot of space and resources for just recording the performance. Instead, it's better to continuously monitor key aspects of system performance, which give you clues to what goes on and when you should pay closer attention, and add a number of specific counters for deeper analysis.

Additionally, you should ensure that your monitoring strategy includes sending out alerts when key thresholds are exceeded, such as available disk space. You should also make sure that key performance counters, such as Processor utilization and system queue, also generate alerts if their threshold is exceeded for a period of time. Peaks will obviously occur; thus, you need to take steps to ensure that you do not generate false alarms for peaks.

It's important to be able to establish growth patterns and trends, which will guide you through the production phase and ensure that any given component is never allowed to become a bottleneck for usage growth in your enterprise. You should be able to predict capacity requirements 6 months

ahead of time, giving you plenty of time to process necessary funding approvals, purchase the needed hardware, and plan for deploying it in the least obtrusive manner for the end users. If you find out that you've got 1GB of disk space left, there's very little time to plan for anything but turning off site creation and effectively freezing the environment while you scramble to source more disk space. A monitoring strategy with periodic reviews ensure you never have to duck for cover when the growth pattern exceeds what was initially thought.

10.11.2 Identifying Future Bottlenecks

There are three main utilization areas, which you should frequently monitor and use to establish growth patterns: processor, memory, and disk. Typically network is included in this, but personally I've yet to see a full duplex 100MBit network card become saturated with SharePoint products and technologies before one of the other components has reached a limit, so I've left it to the list for items to monitor less frequently. However, if you are operating on a 10MBit link, you are more likely to reach a network bottleneck much sooner than another component.

While today 100MBit network speeds are starting to be more common in enterprise deployments, especially if all the users are located in a central location or within the same country, you are still likely to counter deployments where you have to deal with slower WAN network links. Unfortunately, monitoring WAN routes is not possible from the physical servers and must be done by the network administrators, who typically are in charge of upgrading links as thresholds creep closer. Thus, you should talk with your network team about your organization's WAN links and thresholds and ask whether it's possible for you to get a report from them on network usage to the SharePoint so that you can identify possible WAN bottlenecks. Armed with this knowledge, you can keep the end users informed of the situation and involve management as necessary if the situation needs to be addressed. Now, let's get back to the bottlenecks we can monitor and control.

Table 10.9 lists the key system performance counters that you should monitor to establish a performance trend and set up alerts when the utilization creeps near the thresholds. You should establish the alert margins based on how long it takes in your organization to source required extra capacity.

Please see Microsoft Knowledge Base Article Q300237 for more information on setting up alerts on key system components.

Table 10.9 *Important System Performance Counters*

Counter Name	Description	Threshold
Processor: % Processor Time _Total	Combined average CPU utilization. SharePoint, like many other applications, will not perform consistently after 80% constant CPU utilization. Use in conjunction with the Processor Queue Length to identify a processor bottleneck.	80%
System: Processor Queue Length	Number of execution threads in the Processor Queue. Typically, when the queue goes over 6, the server is not able to process requests fast enough, indicating a processor performance bottleneck	6
Memory: Pages/sec	"Hard" pages/sec. When a page is not in memory, it has to be fetched from the page file, which is also referred to as a "hard" page fault. Arguably the most important indicator for memory bottlenecks, and should not get close to 20. Use in conjunction with other memory counters to verify that paging is really an issue.	20
Memory: Available bytes	Memory available for applications and the operating system to use. Use in conjunction with pages/sec to identify a memory bottleneck, as SharePoint search service will happily use as much memory as is available. Low available bytes on its own doesn't necessarily mean you have to add more memory, but low available bytes together with a high pages/sec count indicates the need to add some RAM.	128Mb
LogicalDisk: % Disk Time (for each volume)	Indicates whether the disk is busy performing operations. If the disk time is high for a period of time, it does not necessarily indicate an I/O bottleneck, but does warrant a more in-depth investigation utilizing the Disk Average Read/Write Queue Lengths. High I/O can also be caused by excessive paging, so remember to never read the values in isolation and to keep other factors in mind as well.	50%

This page intentionally left blank

11

Introduction to SharePoint Development

So far in this book we have examined the offerings provided by the SharePoint Server 2007 platform and discussed the various options you have for deploying the product within your organization. This section of the book examines the components of particular interest to developers and Web designers who wish to customize and extend the out-of-the-box functionality. First we'll take a quick look at how the underlying architecture has changed since Windows SharePoint Services 2.0 and SharePoint Portal Server 2003 and discuss some of the major enhancements that the new architecture brings to the table. Then we'll look at some of the development tools that are available to us when customizing SharePoint, namely SharePoint Designer 2007 and Visual Studio 2005. Keep in mind that many of the topics we review in this chapter are covered in more detail later in the book.

11.1 Development Architecture Overview

The underlying platform for WSS 3.0 and SharePoint Server 2007 has been considerably redesigned from WSS 2.0 and SharePoint Portal Server 2003; the most notable changes appear at the ASP.NET and IIS levels. For example, WSS 2.0 and SharePoint Portal Server 2003 use ASP.NET 1.1, whereas WSS 3.0 and SharePoint Server 2007 are built upon ASP.NET 2.0 (Figure 11.1). ASP.NET 2.0 offers significant improvements over its predecessor, such as the introduction of master pages, a native Web Part framework, and pluggable virtual path providers that allow you to retrieve and control the parsing of .aspx pages.

From a performance perspective, the design implemented by WSS 2.0 and SharePoint Portal Server 2003 produces suboptimal results, because each request to a SharePoint site first must be processed by a special ISAPI filter, developed by the SharePoint team, before it can be processed by ASP.NET. The job of the ISAPI filter is basically to examine the URL and determine whether it is requesting a SharePoint managed path. The ISAPI filter uses the list of inclusions and exclusions maintained by the administrator to make its

decision. If the ISAPI filter sees that the URL contains a path listed in the exclusions list, it does not intercept the request. If the request is for a SharePoint managed path, the ISAPI filter either passes the request to the ASP.NET handler or handles the request through a SharePoint ISAPI extension call. The ISAPI extension calls are used to process legacy protocols, FrontPage protocols, the DAV protocol, and any requests for static pages, such as a .doc or .htm file. Requests for dynamic pages, such as .aspx pages, pass to the ASP.NET handler. With WSS 3.0 and SharePoint Server 2007 the ISAPI filter has been completely removed, and *all* incoming HTTP requests route directly to the ASP.NET runtime, regardless of their extension. ASP.NET then forwards the request to Windows SharePoint Services for further processing. So how does ASP.NET know to route a particular request to Windows SharePoint Services? This is now achieved through the use of an HTTP module component and an HTTP handler. An HTTP module is basically a class that implements the IHttpModule interface and is used to intercept HTTP requests. In addition to being able to intercept requests, the HTTP module components are capable of modifying the overall context of the request being processed. An HTTP handler is a class that implements the IHttpHandler interface, which is designed to handle the HTTP request. The Microsoft .NET Framework defines a number of modules out of the box, such as the WindowsAuthentication module, which enables ASP.NET applications to use Windows and IIS-based authentication, and the FormsAuthentication modules, which enable applications to use forms authentication. Modules that are available to all applications are defined in the machine.config file on a server, and application-specific modules are defined in the web.config of the associated Web application. If you examine the web.config of a typical SharePoint Web application, you will see two SharePoint specific HTTP modules defined in the httpmodules section, namely, SPRequest for intercepting SharePoint requests and PublishingHttpModule for intercepting publishing requests. If you take a quick look at the httphandlers section of the same Web.config file, you will also notice a SharePoint-specific httphandler: SPHttpHandler. The great thing about this design is that as a developer, you can create your own custom HTTP module and HTTP handler components to perform additional processing on incoming HTTP requests. Another area of improvement is the ability of ASP.NET 2.0 to load .aspx pages that reside in locations other than the local file system, such as a SQL Server database. ASP.NET 2.0 achieves this through the use of virtual path providers. Virtual path providers allow a Web application to retrieve resources from a virtual file system. Windows SharePoint Services and SharePoint Server 2007 implement a SharePoint-specific virtual path provider (SPVirtualPathProvider) that retrieves content from the SharePoint SQL databases and converts it into a format that ASP.NET can understand.

Figure 11.1 provides an architectural overview of the Windows SharePoint Services 3.0 and SharePoint Server 2007 platform from a developer's

perspective. As mentioned many times throughout this book, Windows SharePoint Services 3.0 is built upon ASP.NET 2.0, and as such it inherits all the new enhanced capabilities of ASP.NET 2.0, a few of which are listed in Figure 11.1. SharePoint Portal Server 2001 first introduced the concept of Web Parts to the enterprise and it was subsequently rolled up as a key component of the Windows SharePoint Services 2.0 and SharePoint Portal Server 2003 architecture. A Web Part is basically a control that renders content to the user and that can be placed in any desired region within a Web page. Web Parts became very popular among the developer community because they were easy to develop and, if coded correctly, provided much in the way of flexibility and portability. Their successful adoption rate ultimately led to the inclusion of the Web Part Framework architecture in ASP.NET 2.0, which means that you no longer require SharePoint to develop and implement Web Parts in your Web applications. Web Parts are still at the heart and soul of SharePoint Server 2007, but now you have the advantage of being able to develop a Web Part control in ASP.NET 2.0 and have it run in SharePoint Server 2007.

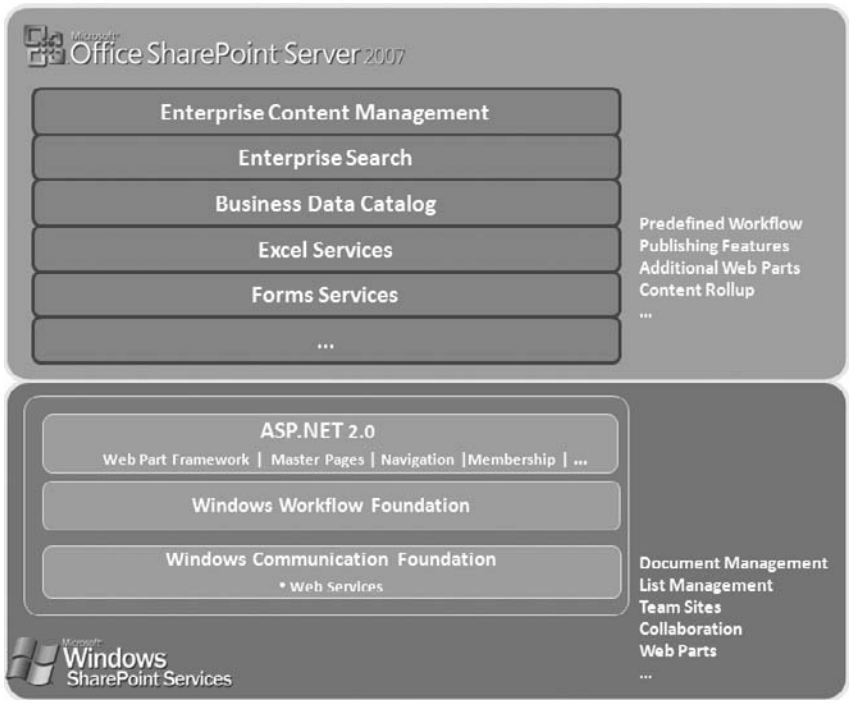


Figure 11.1 *SharePoint Server 2007 architecture.*

Both SharePoint Portal Server 2001 and SharePoint Portal Server 2003 left a lot to be desired when it came to the ease of applying corporate branding to sites. The introduction of master pages combined with the enterprise

Web content management capabilities of SharePoint Server 2007 helps alleviate many of the issues experienced with previous versions. Master pages are an integral part of the ASP.NET 2.0 architecture that has been embraced by SharePoint Server 2007. A master page represents the chrome, or brand, of a page, providing a separation between the presentation and the actual content of the page. The chrome typically consists of elements common to all pages in a site, such as the banner, navigation, search controls, and login controls. Content pages inheriting a specific master page will adopt the chrome of the master. All pages in SharePoint Server 2007 and WSS 3.0 inherit a master page. This is particularly good news in the case where a page has been customized. When a page is customized in SharePoint, a copy of the customized page is stored in the content database. In previous versions of SharePoint the branding code was included as part of the page, so if the branding code was updated directly on the page stored on the file server, all customized versions of the page would not pick up the new branding code. With SharePoint Server 2007 and WSS 3.0, both customized and uncustomized pages will pick up changes made to the master page, because their page contains a reference to the master page and not the actual branding code.

When it comes to navigation, SharePoint Portal Server 2003 provides little in the way of customization. Web designers and administrators often want to change the navigation on a site at the very least to add custom links. ASP.NET 2.0 includes various provider models and associated controls that greatly improve the overall navigation of Web sites. WSS 3.0 and SharePoint Server 2007 take full advantage of this capability. If you examine the navigation areas of one of the SharePoint default master pages, such as default.master, you will notice that the navigation is provided by ASP.NET 2.0 navigation Web controls. The navigation controls include the Menu, TreeView, and SiteMapPath controls. The Menu control displays a menu that can be aligned either vertically or horizontally, the TreeView control displays the entire site map hierarchy in a collapsible vertical tree, and the SiteMapPath displays a breadcrumb. You can easily customize the Menu control and apply your own styles, or reference a custom style sheet, thereby changing the look and feel of the navigation on all pages that inherit your master page.

Previous versions of SharePoint Portal Server and Windows SharePoint Services are limited when it comes to authenticating users in that you are restricted to using Windows NT-based user accounts. The Membership provider interface that ships with ASP.NET 2.0 allows you to create your own custom authentication mechanism for SharePoint. For example, you could store all your user credential information in a separate SQL database and use the AspNetSqlMembershipProvider class, in conjunction with enabling forms-based authentication, to implement your custom authentication process.

In addition to ASP.NET 2.0, WSS 3.0 and SharePoint Server 2007 are built upon Windows Workflow Foundation (WWF) and Windows Commu-

nication Foundation (WCF) and require these components to be installed. Both WWF and WCF are included in the .NET Framework 3.0, so if you install the .NET Framework 3.0 you do not need to install WWF and WCF separately. So what are WWF and WCF and how are they related to SharePoint? WWF is Microsoft's workflow platform, which offers a workflow runtime engine, a common set of tools, and programming interfaces that developers can use to build their workflow applications. In this definition, the word *developers* includes the Microsoft development teams working on various workflow projects, such as the workflow functionality provided with SharePoint Server 2007. WCF is a set of technologies built around the Web services architecture that allow you to build and deploy connected systems.

SharePoint Server 2007 extends the Windows SharePoint Services platform to provide capabilities such as enterprise content management, enterprise search, forms services, and business integration capabilities through the business data catalog and Excel services. We have touched on many of these capabilities in the first section of this book and will revisit them again from a developer's perspective throughout this section. The enterprise content management capabilities of SharePoint Server 2007 cover a wide range of activities, including the ability to see how a page renders *before* you actually make it available to your entire audience, and more importantly, roll back any changes should you wish to revert to a previous version of a page. This staging concept adds an additional layer of control, and comfort, when deploying changes to your portal. Additionally, all libraries, whether it's a regular document library, a Web Part gallery, or a master page gallery, share a common architecture and a common set of capabilities such as the ability to assign workflows, set information management policies, and define security permissions. The SharePoint Server 2007 SDK contains a set of sample code and white papers that show you how to further extend the enterprise content management capabilities of SharePoint Server 2007. For example, the SDK includes sample code to create a document inspector that replaces terminology used in Microsoft Word documents with another terminology better suited for public use.

SharePoint Server 2007 brings with it enterprise search capabilities and provides developers with tools to customize and extend the search experience. For example, in Chapter 6, Search, we showed you how the search user interface is comprised of a set of customizable Web Parts. You can easily customize the XSLT associated with the search core results Web Part to change how the search results appear on the search results pages. Once again, the SharePoint Server 2007 SDK provides samples to show you how this is achieved. The customization possibilities do not end here; you can create your own custom search Web Parts that leverage the query object model to execute queries against the search engine.

Chapter 7, People, and Chapter 8, Business Intelligence and Process, introduced us to two capabilities that will make developers lives much easier when it comes to building business intelligence solutions for SharePoint Server 2007: The Business Data Catalog (BDC) and Excel Services. The BDC is a shared service that allows you to create a mapping between SharePoint Server 2007 and your back-end business applications. Once this mapping is in place, information from the back-end business applications can be presented in SharePoint without the need for additional code. For example, you can use the BDC to display information from an Oracle database, from SAP, or from a home grown application. SharePoint Server 2007 provides five Web Parts out of the box that allow you to display business data from applications registered in the BDC. Excel Services is another shared service that provides the capability of storing and displaying server side Excel workbooks in SharePoint Server 2007. The architecture of Excel Services includes Excel Calculation Services components that are responsible for loading the workbooks, performing all the necessary calculations, and maintaining the integrity of the data. Excel Services includes an Excel Web Service that allows developers to leverage Excel Services programmatically from their applications, which is an extremely powerful feature when it comes to extensibility. For example, your application could retrieve some data stored in an Excel workbook and calculated on the server and, based on the result of this calculation, perform a specific action. Another example is that logic in your application could actually update data in an Excel workbook on the server, which then subsequently triggers an Excel calculation that ultimately presents users with updated information on their portal.

InfoPath has been around since 2003 and has proven to be a great technology for developing sophisticated electronic forms. One of the factors that impeded its adoption was the fact that to use the forms, you needed to have the full-blown InfoPath client installed on your desktop. SharePoint Server 2007 introduces InfoPath Forms Services, which overcomes this limitation by allowing form authors to create Web browser-compatible forms that can be stored in SharePoint. These forms can then be viewed and edited through the browser by users who have the appropriate permissions; users no longer need to have the InfoPath client installed on their desktop to avail themselves of this functionality. Now, the forms that you develop are accessible to your users anywhere on a variety of devices, such as a Web-enabled PDA. As with previous versions of InfoPath, the form templates are designed in the InfoPath client. InfoPath 2007 comes equipped with a compatibility checker, which is a tool that you can use to ensure that your forms are compatible with InfoPath Forms Services. InfoPath Forms, in conjunction with the workflow capabilities offered out of the box with SharePoint Server 2007, provides developers with a powerful development platform from which to build applications and customize SharePoint to suit specific business needs. In Chapter 18, InfoPath Forms and Workflow, we discuss workflow in more

detail and delve into the topic of InfoPath Forms Services to show an example of using an InfoPath form in a workflow.

As you can see, Windows SharePoint Services 3.0 and SharePoint Server 2007 provides a highly extensible platform upon which you can build sophisticated applications to meet your business needs. Thankfully, there are a number of tools that can make the development journey much easier. Let's have a quick look at two of the Microsoft offerings that you will no doubt leverage when customizing and extending SharePoint, namely SharePoint Designer 2007 and Visual Studio 2005.

11.2 Microsoft Office SharePoint Designer 2007

Microsoft Office FrontPage 2003 (FrontPage) remains a very popular tool for editing WSS 2.0 and SharePoint Portal Server 2003. Back in 2003, FrontPage was Microsoft's main product for Web site creation and management and includes all of the necessary design, authoring, data, and publishing tools to create dynamic Web sites. The FrontPage team did a tremendous job of providing an easy-to-use medium for designing and extending your SharePoint site, and FrontPage became Microsoft's recommended product for advanced customizations of both WSS 2.0 and SharePoint Portal Server 2003 sites. One of the more popular integration features of FrontPage is its ability to allow you to create a special type of Web Part, known as a data view Web Part. This Web Part in conjunction with the data source catalog allows you to present live views of data in your SharePoint site. The data can come from a variety of sources, such as a SharePoint list, a SharePoint library, an XML Web service or even an Oracle database. The data retrieved by the data view Web Part is formatted using XSLT for final presentation in the browser. This allowed Web designers, developers, and power users to *easily* create customized views of data to display in their SharePoint sites. The ease of use of FrontPage is a key factor in its success as a customization tool for SharePoint. Building on this success, Microsoft has evolved FrontPage into two separate products, one that specifically targets SharePoint users, known as Microsoft Office SharePoint Designer 2007 (SharePoint Designer), and one that targets professional Web designers that are *not* designing SharePoint sites, known as Expression Web. SharePoint Designer is now Microsoft's recommended tool for creating solutions, designing Web pages, and authoring content for SharePoint sites.

SharePoint Designer contains a host of designer tools and SharePoint integration components that not only ease the customization of SharePoint content but also aid the actual administration and management of the content. From a manageability perspective, SharePoint Designer introduces the concept of contributor mode by providing SharePoint administrators with an additional layer for administering who can do what to the content of

their SharePoint sites. Contributor mode is enabled by default and is controlled by a combination of SharePoint permission levels and SharePoint Designer Contributor settings. For example, default permission levels such as Approve, Full Control, Manage Hierarchy, Design, Contribute, Read, View Only, Restricted Read, and Limited Access can each be linked to a specific contributor group, which further defines what they can do within the confines of SharePoint Designer, such as limiting page edits to a specific set of folders or disabling some of the SharePoint Designer menu options. Figure 11.2 shows the SharePoint Designer Contributor Settings dialog and its associated Contributor Group Properties dialog. You can access the contributor settings by selecting the Contributor option from the bottom of the Task Panes menu in SharePoint Designer. The figure shows the default content authors contributor group linked to the SharePoint contribute permission level.

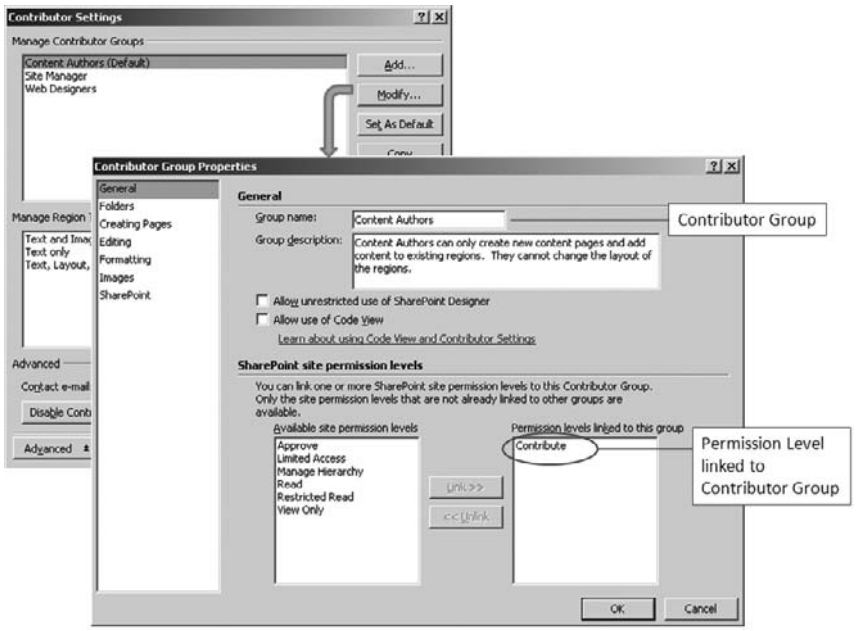


Figure 11.2 *Contributor Settings in SharePoint Designer 2007.*

The list that appears on the left of the Contributor Group Properties dialog shows the different options to which you can refine access. Figure 11.3 shows the default settings for Content Authors when creating pages in SharePoint Designer. In this particular example, the content authors are allowed to create blank pages and Web Part pages, and are restricted to only being able to delete and rename the pages and folders that they create. Additionally, they can only create pages from the listed master page, which in this case is the default.master.

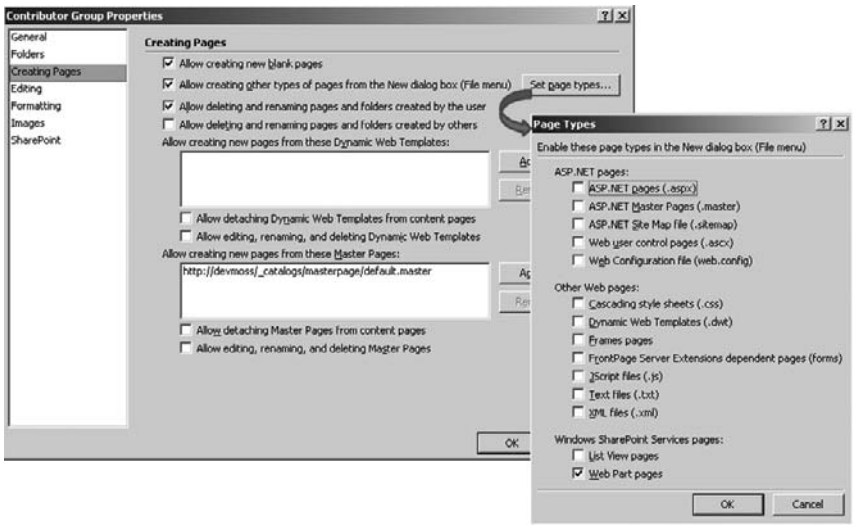


Figure 11.3
Creating pages settings for content authors contributor group.

SharePoint administrators and site managers can leverage the contributor settings of SharePoint Designer to further protect their Web sites from being inadvertently corrupted. This is also empowering and reassuring to content creators because they can have access to the rich design features of SharePoint Designer without worrying that they may break the site. One welcome feature of the Contributor settings is the ability to control which users are allowed to overwrite pages that are included in the site definition, such as default.master and the list pages AllItems.aspx and EditItem.aspx. With previous versions of SharePoint, when you edit a SharePoint page with FrontPage 2003, a version of the page stored with the site definition on the file system is copied to the database along with the changes made to the page. Subsequently, any time that page is requested, the customized page stored in the database is presented to the user. This process was known as unghosting. Ghosted pages are the pages stored on the file system with the site definition, and unghosted pages are the FrontPage 2003 customized versions of these pages stored in the database. Once a page is unghosted, it's basically detached from its corresponding site definition page, and with SharePoint Portal Server 2003 and WSS 2.0 this means that the page will no longer pick up customizations made to the page on the file system. This caused a lot of disruption with respect to rolling out enterprise-wide updates for the average SharePoint deployment. Additionally, with the way that SharePoint Portal Server 2003 and WSS 2.0 is designed, unghosted pages caused a considerable performance hit in large deployments. To avoid these issues, the only real options available to IT departments are to either configure SharePoint to disable the ability for sites to be edited using FrontPage 2003 globally, or to update their usage policies to state that any sites edited with FrontPage 2003 would be unsupported. The concept of ghosted and unghosted pages still exists in the SharePoint Server 2007 and

WSS 3.0 world; however, the terminology has been replaced with uncustomized and customized pages respectively, and architectural improvements along with the introduction of master pages means that unghosted (customized) pages do not cause quite so much disruption as before. Add this to the fact that you can dictate which users have the capability to customize pages included in the site definition, and it becomes apparent that SharePoint Designer is a tool to be embraced by the SharePoint populace. Furthermore, should an appropriately privileged user inadvertently customize a page from its associated site definition page, SharePoint Designer provides the ability to easily identify customized pages and reset these pages back to their site definition (Figure 11.4). This behavior is akin to the WSS 2.0 reghosting tools that were made available by third-party vendors.

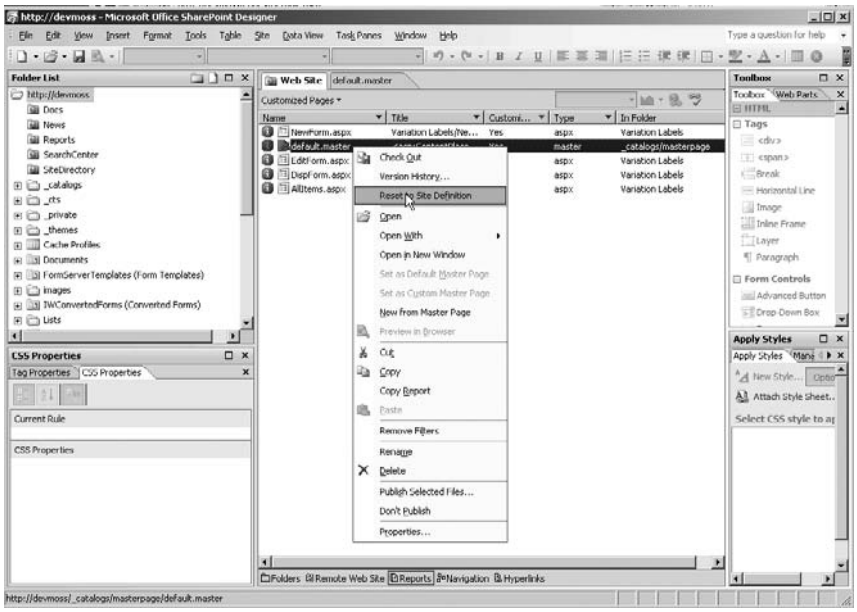


Figure 11.4
Resetting a master page site definition.

The layout of the SharePoint Designer environment is very intuitive and lends itself to both the novice user and the experienced developer. As previously mentioned, SharePoint Designer offers a set of tools to make applying and managing your customizations much easier. The majority of these tools are available via task panes. SharePoint Designer allows you to have multiple task panes open at once, and you can arrange which task panes appear and where they appear within the SharePoint Designer user interface. Figure 11.5 shows the SharePoint Designer displaying, moving left to right, the Folder List task pane, the CSS Properties task pane, the Toolbox task pane, and the Manage Styles task pane. The Folder list task pane provides a hierarchical view of your SharePoint site and the underlying data. Icons provide informa-

tion on the type of data and its current state—for example, there is a special icon that represents master pages and additional icon information that informs the user whether the page is checked in or checked out on the site. When you right-click a folder or a page from the Folder list task pane, a context-sensitive menu appears with a list of options that you can perform on the selected item, such as the ability to access its version history. The CSS Properties, the Manage Styles, and the Apply Styles task panes are great additions for customizing the look and feel of the pages in your site. The Manage Styles task pane informs you of the CSS style associated with the currently selected area in the main designer window. In the example, shown in Figure 11.5, the horizontal navigation bar is selected in the bottom window of the main designer area and the CSS style that matches in the Manage Styles task pane is .ms-banner. If there is a specific area on the page that you would like to customize, the Manage Styles task pane provides a convenient way of finding out the relevant style classes that need to be updated. The CSS Properties task pane works in conjunction with the Manage Styles task pane and displays the properties associated with the currently selected style in the Manage Styles task pane. In our example, the properties of the .ms-banner style are listed in the CSS Properties task pane.

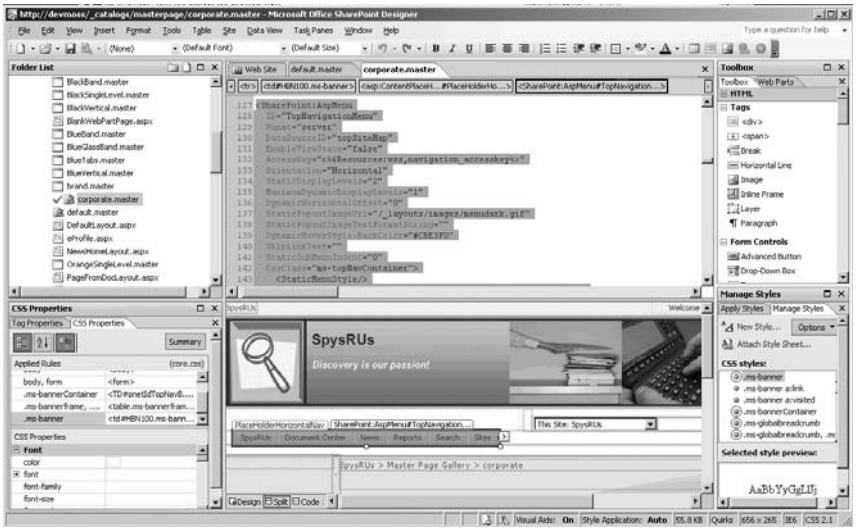


Figure 11.5 *SharePoint Designer environment.*

The CSS Properties task pane also takes into consideration the selected area in the main designer window and lists all the applied styles. The Apply Styles task pane lists all the styles in the style sheets attached to the page and, with a simple mouseclick, allows you to apply a specific style to the currently selected area in the main designer window.

The main designer window has a number of different views associated with it: Design, Split, Code, and Preview. The Design view provides a what-you-see-is-what-you-get (WYSIWYG) editor that allows you to design your Web page using graphical tools and formatting. The Design view closely matches the appearance of the Web page in the browser while still providing information on the structural components of the page. The Code view displays the code of the Web page in the designer window. The Split view divides the designer window into two panes and displays the Design view in one pane and the Code view in another, allowing you to see both views at one time. As you select items in one pane, the other pane scrolls to and highlights the corresponding item.

The ToolBox task pane lists, by category, the various controls that can be inserted onto the current page. For example, there is a large set of SharePoint controls (Figure 11.6) grouped by the following categories: Data View Controls, Server Controls, Page Fields, and Content Fields. The Page Fields and Content Fields controls are specific to publishing pages and allow you to drag and drop controls that are mapped to content types onto your page. The Server Controls group provides convenient access to many of the SharePoint-defined Web controls, and as with all the controls in the ToolBox task pane, allows you to simply drag and drop them onto your page as needed. The Data View Controls group lists all the available Web controls associated with the data view Web Part. The data view Web Part that we are referring to here is an updated version of the FrontPage 2003 data view Web Part that we mentioned at the beginning of this section. The updated data view Web Part provides additional functionality such as the ability to aggregate views across multiple data sources and the ability to update data directly from the data view Web Part. Chapter 13, Building and Deploying Web Parts, delves into more detail on the data view Web Part and shows an example of the data view Web Part in action. SharePoint Designer's support for Web Parts does not end with the data view Web Part; there is a separate Web Parts task pane (Figure 11.6) from which you can browse the various Web Part galleries and drag and drop Web Parts directly onto your page. The Web Parts task pane also provides the ability to add new Web Part Zones onto your Web Part page.

WSS 3.0 and SharePoint Designer 2007 provide a lot of capabilities when it comes to creating workflows and associating those workflows with content stored in SharePoint. SharePoint Designer provides a graphical workflow editor that allows you to define the various steps that comprise a workflow, along with a set of conditions for each step and the associated actions. For example, Figure 11.7 shows a very rudimentary compliance workflow that checks to see whether a document meets the 28CFR compliance requirement. The workflow has two steps: Review Document and Shred Document. The Review Document step has two conditions defined; the first states that if the document has been checked in and the comment field is set

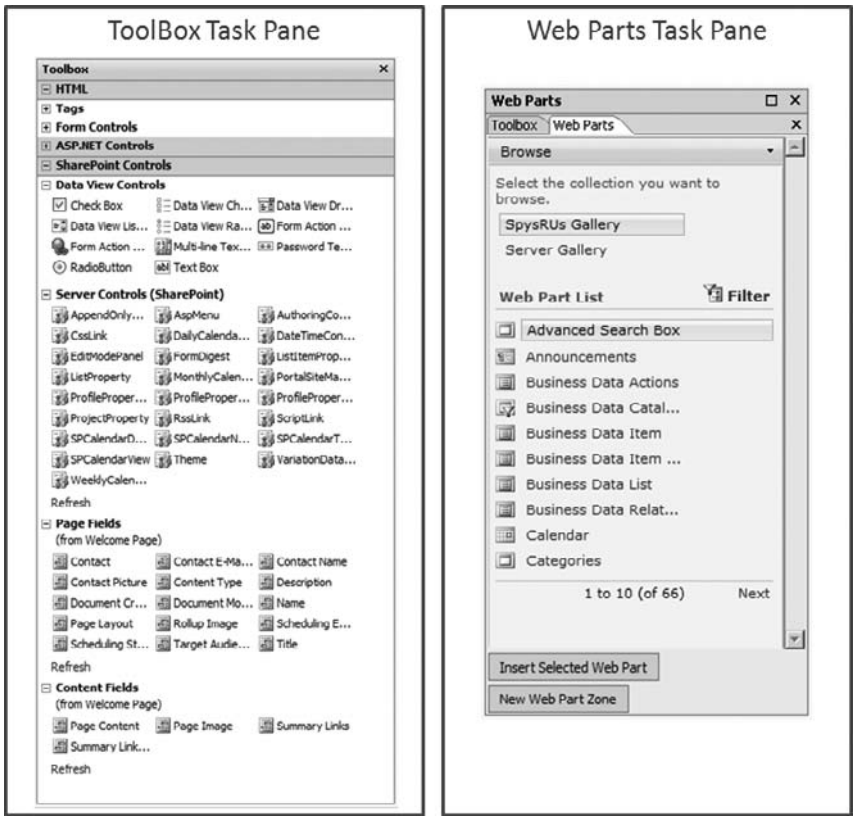


Figure 11.6 *SharePoint Designer task panes.*

to "28CFR Compliant," then the approval status should be set to Approved. The second condition states that if the document has been checked in and the comment field is *not* set to "28CFR Compliant," then the approval status should be set to Rejected. The Shred Document step of the workflow has one condition associated with it that basically checks the approval status; if the approval status equals "Rejected," it deletes the document. The SharePoint Designer workflow editor is much more simplistic in nature than the workflow editor provided by the Visual Studio 2005 Designer for Windows Workflow Foundation, and it is geared towards the rapid development of workflows that do not require custom code or custom activities. We revisit the topic of workflow towards the end of this book.

11.3 Microsoft Visual Studio 2005

SharePoint Designer 2007 is a great tool to have in the SharePoint developer's toolkit and provides an excellent interface for customizing the look and feel of your Web pages, building simple adhoc workflows, adding Web

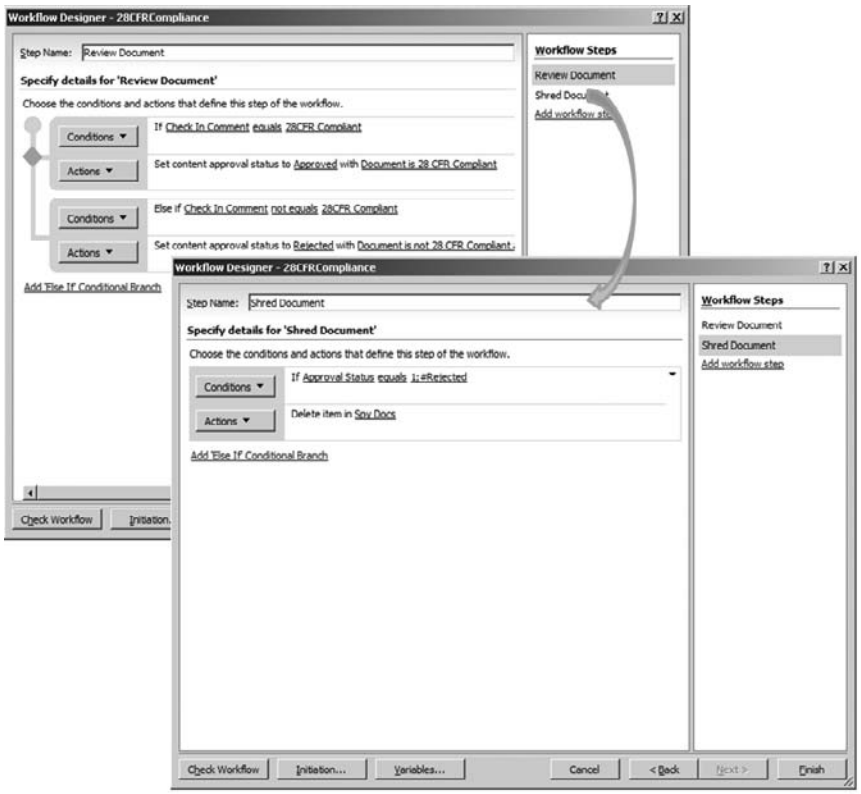


Figure 11.7
Workflow designer.

Parts and updating the content of your site. However, when you need to delve into the more complex aspects of customization and extensibility, such as developing custom Web Parts, site definitions, Features, and server controls, you will find yourself turning to Microsoft Visual Studio 2005. Since Visual Studio 2005 is the primary development environment for building ASP.NET applications, it only stands to reason that it is ideal for developing SharePoint Server 2007 applications. Microsoft has provided a number of extension packs that provide further integration when building SharePoint solutions, such as the Visual Studio 2005 extensions for Windows SharePoint Services 3.0 and the Visual Studio 2005 extensions for Windows Workflow Foundation. The WSS 3.0 extensions package includes project templates, item templates and a solution generator. The Visual Studio project templates provide templates, for building Web Parts, team site definitions, blank site definitions and list definitions. The Visual Studio extensions provide item templates for adding items of a specific type to an existing project. The item templates include templates for building Web Parts, custom fields, content types, list definitions, and modules. Finally, the SharePoint Solution Generator creates a site definition from an existing SharePoint site. This is a great tool, as it allows you to perform as many of

your customizations as possible through the Web browser and SharePoint Designer 2007, and then, instead of saving the site as a custom site template (.stp) file, you can use the Solution Generator to create a custom site definition that ultimately improves performance. It also provides an easy mechanism for rolling out updates to the site definition. The current version of the extensions for WSS 3.0 only supports C#; Visual Basic templates are not provided.

Not only do the Visual Studio extensions help with building SharePoint solutions, but they also automate the complete deployment process. Let's take the example of a very basic Web Part project created using the Visual Studio Web Part project template. In this example, we'll simply uncomment the text line in the sample project code and replace the text with "Hello World" (Figure 11.8). Next we'll build the project, which creates the appropriate assembly file that we'll need when deploying the Web Part. Once we have the assembly file, there is a series of steps that must be performed to actually deploy the Web Part, such as copying the Web Part to the global assembly cache and registering the Web Part as a safe control in the Web application's Web.config. The Visual Studio 2005 extensions for WSS 3.0 provide the capability of automating the deployment process for you so that you maintain consistency when deploying customizations to SharePoint. The deployment functionality is based upon the WSS 3.0 solution framework, which we'll discuss in more detail shortly, and creates all the necessary components required to deploy the Web Part on all the WFE servers within your SharePoint environment. The deployment process is initiated by clicking the Deploy option from the Build menu. Once the deployment process is complete, the Web Part appears in the Web Part gallery, ready to be placed on a SharePoint page (Figure 11.9). The extensions take a lot of the labor out of creating and deploying solutions, freeing developers to focus more of their attention on their creativity and programming efforts.

The Visual Studio Extensions for Windows Workflow Foundation extends the Visual Studio environment to provide support to developers for building workflow applications that leverage Windows Workflow Foundation. Once again, this is great news for developers already used to developing applications in Visual Studio 2005, as it offers a familiar environment from which to build workflow applications. Once the workflow extensions have been installed, an additional workflow option appears under both the Visual Basic and C# project types in the New Project dialog (Figure 11.10). The Workflow option then offers a number of workflow templates that you can use as the foundation for your custom workflow solution. The templates include workflow libraries used for creating sequential workflows, state machine workflow libraries used for creating state machine workflows, and activity libraries used for creating a library of activities that can be reused in future workflow solutions.

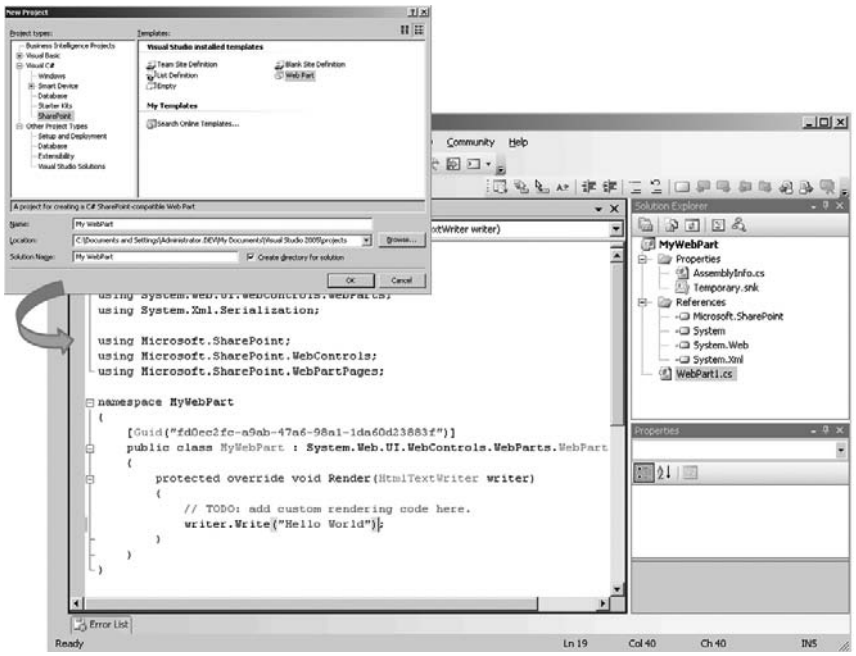


Figure 11.8
Creating a new Web Part with Visual Studio.

Figure 11.9
Web Part deployed to Web Part gallery.

The workflow extensions include a Workflow Designer (Figure 11.11) that provides a visual representation of the workflow code. The Workflow Designer window functions the way most design windows do, in that you can drag and drop components directly onto the designer page, which ultimately generates some code behind the scenes.

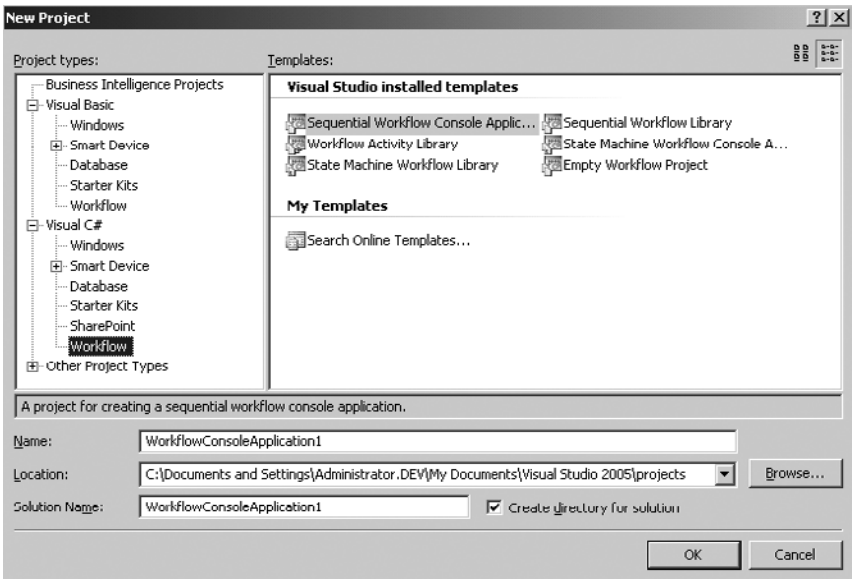


Figure 11.10 *Workflow project templates added to Visual Studio.*

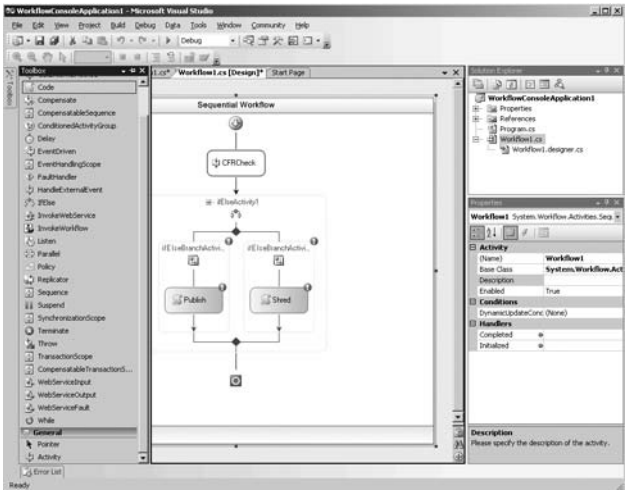


Figure 11.11 *Visual Studio Workflow Designer.*

11.4 Windows SharePoint Services 3.0 Solution Framework

The Windows SharePoint Services 3.0 solution framework provides a method for collating the various components that comprise a solution and packaging them together into a solution file. The solution file is basically a

cabinet based format with a .wsp extension instead of a .CAB extension. As we saw in our previous Web Part example, once the solution is created, the solution file can then be used to deploy the contained functionality to a SharePoint farm, which in this instance was a "Hello World" Web Part.

A solution goes through several phases during its lifecycle, from creation through deletion. Figure 11.12 illustrates the solution lifecycle as it pertains to the WSS 3.0 solution framework. The whole premise behind the solution framework is to provide separation and consistency between the creation of a solution package, which is seen as a developer's task, and the actual deployment of the solution package, which is more of an administrative task. Phase one of the lifecycle shows the creation of the solution file. There are a variety of methods that developers can adopt when creating a solution file, and we'll go into more detail on these shortly—but the most important point to make here is that the creation of the solution is a developer task; you do not need admin privileges to create a solution package. Granted, you will most likely want elevated privileges in your own development environment when building solutions to make your life easier; they are not, however, a requirement. Once you have created your solution file (.wsp file) you pass that file off to your administrator for deployment. The administrator takes your file and, using the SharePoint command line administration tool (stsadm.exe), adds the solution file to the WSS 3.0 configuration database. Once the solution has been added to the solution store, the administrator can then either deploy the solution immediately or schedule the solution to be deployed at a future point in time. If the solution contains resources that are scoped at the Web application level, then the administrator has the option to specify the Web applications to which the solution should be deployed; there is also an option available to deploy to all Web applications. Solutions that do not contain Web application–scoped resources are deployed globally.

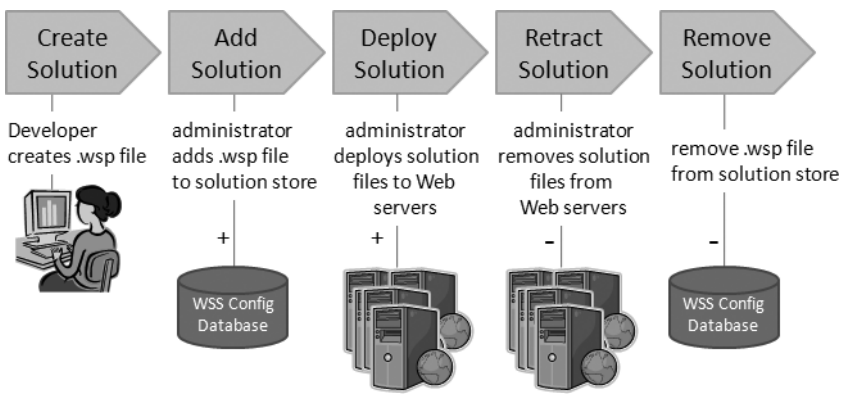


Figure 11.12
Solution lifecycle.

The solution deployment process can be initiated using stsadm.exe, or using the central administration browser user interface via the operations solution management page. Figure 11.13 shows the deploy solution page, from which the administrator can configure when and where a solution is deployed. This particular example shows that the solution package deploysampleWebpart.wsp is configured to be deployed immediately; however, the page also includes a scheduling option. The scheduling option leverages the WSS timer services to create a timer job for the deployment. During deployment, the files contained in the solution package are examined and placed in their appropriate location on the server. For example, if the solution package specifies some _layouts files, they are placed in the _layouts directory, if the solution package includes some Features the Feature folders are copied to the features directory; (`\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\FEATURES`), and if the solution package includes Web Parts, the Web Parts are copied to the appropriate location, such as the global assembly cache and a `SafeControl` entry is placed in the Web.config file of each Web application to which the solution is deployed.

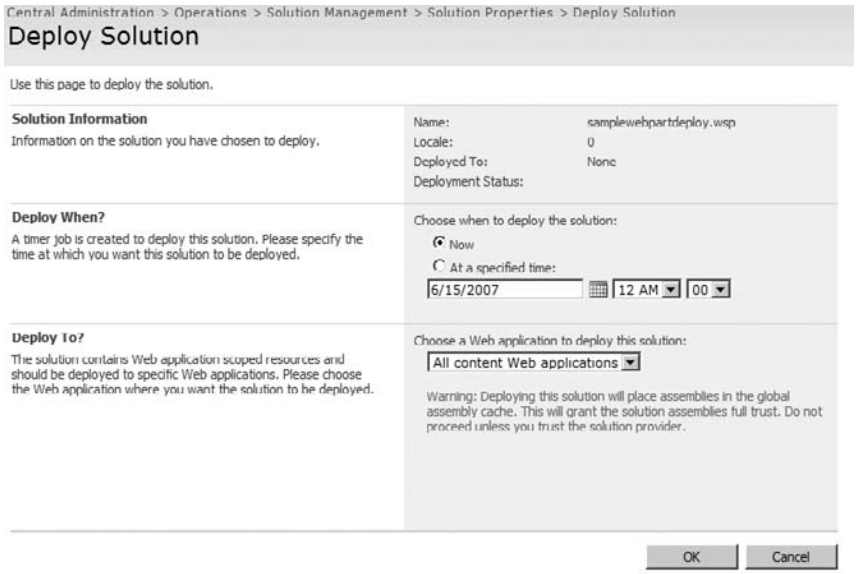


Figure 11.13 *Deploying a solution using the browser user interface.*

The solution file in our example is named deploysampleWebpart.wsp and, as its name suggests, it is used to deploy a sample Web Part. As previously mentioned, the deployment process for a solution that includes Web Parts involves copying the assembly files associated with the Web Part to the global assembly cache and registering the Web Part in the Web.config file of the appropriate Web applications. Figure 11.14 shows the solution proper-

ties page after the deploysampleWebpart.wsp is deployed. Notice that the solution was successfully deployed to three Web applications on ports 80, 1230, and 1210. The Web.config file for each application has a new SafeControl entry registering the Web Part, named SampleWebPart, with the application. In addition, the Web Part assembly file, SampleWebPart.dll, is now installed in the global assembly cache. A solution file can be configured to provision files into SharePoint libraries. In our Web Part example our solution could also add the Web Part to the site's Web Part gallery, thereby making the Web Part instantly available to users of the SharePoint site. This makes a SharePoint administrator's life much simpler, as the solution management administration pages become a one-stop shop for rolling out new functionality.

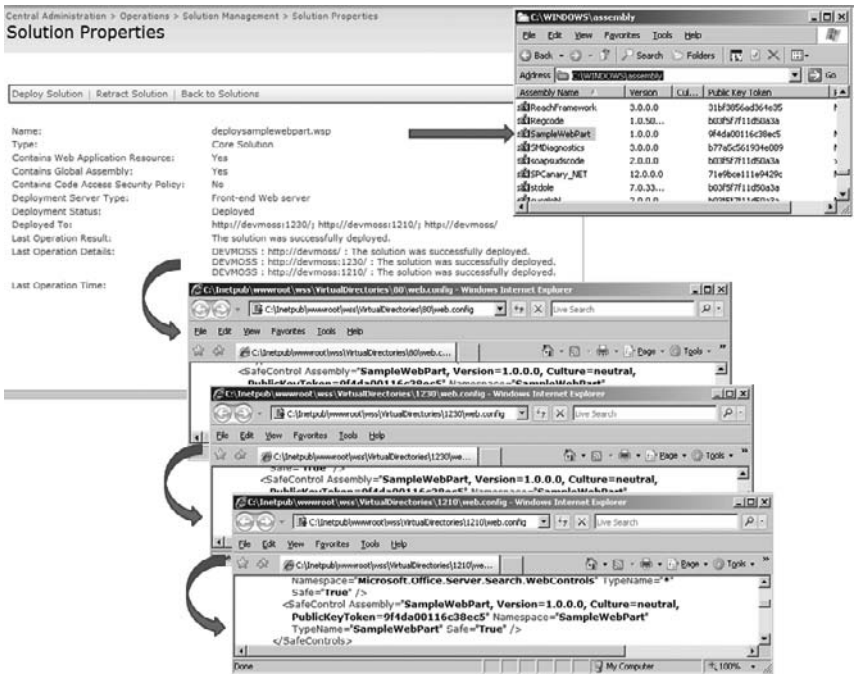


Figure 11.14
Web Part solution file deployment.

Not only is it important to be able to roll out new functionality in a consistent and efficient manner, it is equally important to be able to quickly and consistently roll back the system and a solution to its prior state. The WSS 3.0 solution framework includes a retract capability that allows you to remove the various solution components from the servers and Web applications in which they are installed. If we retract the deploysampleWebpart.wsp solution, the SafeControl entries are removed from each Web application in which the Web Part is installed, the Web Part file is removed from each site's Web Part gallery, and the Web Part assembly is removed

from the global assembly cache of each WFE server in which it resides. The solution file remains in the solution store and is available should the administrator want to redeploy the solution in the future. The retraction process can be initiated using stsadm.exe or using the central administration browser user interface via the solution properties page. If you look again at Figure 11.14, you will notice that the top horizontal menu bar includes the Retract Solution option, which can be used to retract the deploysampleWebpart.wsp solution components.

An administrator has the capability to remove a solution completely from the solution store using the stsadm.exe `deletesolution` option, or the Remove Solution option available from the solution properties page. A solution cannot be completely removed if it is currently deployed; the solution must first be retracted. You can use the `–override` parameter in conjunction with the `deletesolution` stsadm.exe operation; however, while this forcibly removes the solution from the solution store, it does not actually retract the solution.

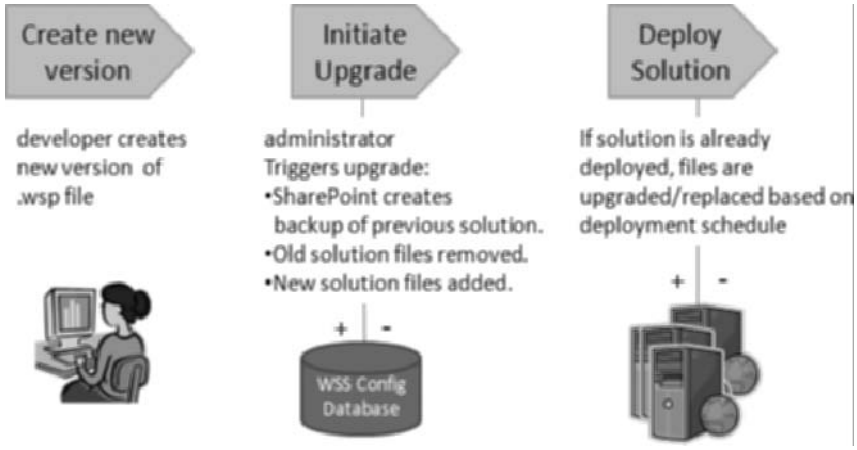


Figure 11.15 *Upgrading a solution.*

So far we have covered the creation, deployment, retraction, and removal of solution files, but we have not discussed how solution files are upgraded. The solution framework supports the upgrade of solutions, removing much of the pain of this process for developers and administrators. In the WSS 3.0 solution framework world, a solution upgrade basically refers to the deployment of a solution that shares the same solution identifier with another, already deployed, solution. Figure 11.15 illustrates the basic steps involved in the upgrade process. The developer creates a new version of the solution and hands the solution package to the SharePoint administrator. The administrator can initiate the upgrade process using the `upgradesolution` stsadm.exe command, which triggers a series of events. First SharePoint creates a backup of the previous solution in the solution store; next, the pre-

vious version is removed from the solution store and the new solution file is added; and then, the various solution components are installed or replaced depending on whether the solution is currently deployed. The backup step is important, as it provides a roll-back option in the event that there is a failure during the upgrade process.

11.4.1 Solution File Composition

So, what is the composition of a solution file? The root of the solution file contains a solution manifest file named manifest.xml. Manifest.xml is a required file in a solution package and cannot be named anything else. A solution file can contain a number of additional components, such as site definitions, Features, Web Part files, assemblies, code access security policies, template files, and root files. The template files and root files can include _layouts files, resources (.resx), and resource files such as .doc or .ppt files. The solution manifest is comprised of a number of elements that define the structure and contents of the .wsp file. The top level Solution element has a number of child elements, as shown in Figure 11.16. Figure 11.17 shows an example of the manifest.xml for deploying a basic Web Part. To deploy a Web Part assembly, you define the current location of the Web Part assembly and its target location using the Assembly element, which is a child element of the Assemblies element. In our example, we are deploying the assembly named SampleWebPart.dll to the global assembly cache. As part of this deployment, a Safe Control entry must be added to the Web.config file of each SharePoint Web application upon you wish to install the Web Part; therefore the next logical entry needed in our manifest file is the SafeControl element.

Figure 11.16 *Manifest.xml elements.*

```
- <Solution SolutionId="4AFC1350-F354-4439-B941-51377E845F2B"
    xmlns="http://schemas.microsoft.com/sharepoint/">
    <ApplicationResourceFiles />
    <FeatureManifests />
    <TemplateFiles />
    <RootFiles />
    <Assemblies />
    <DwpFiles />
    <SiteDefinitionManifests />
    <Resources />
    <CodeAccessSecurity />
  </Solution>
```

In addition to the solution manifest file, a solution file includes all the files being deployed, such as the Web Part assembly and the associated Web Part description file (.dwp or .Webpart). If you are leveraging makecab.exe, to make the cabinet file, then your solution will also require a diamond directive file (.ddf). The .ddf file contains information required to compress your files into a cabinet file; the .ddf file is not placed in the cabinet file.

Figure 11.17 *Sample Web Part Manifest.xml file.*

```
<?xml version="1.0" encoding="utf-8" ?>
- <Solution SolutionId="599D12B4-134B-4a3e-B0AA-4E4FDDAB3902"
    xmlns="http://schemas.microsoft.com/sharepoint/">
  - <Assemblies>
    - <Assembly Location="SampleWebPart.dll" DeploymentTarget="GlobalAssemblyCache">
      - <SafeControls>
          <SafeControl Assembly="SampleWebPart, Version=1.0.0.0, Culture=neutral,
            PublicKeyToken=9f4da00116c38ec5" Namespace="SampleWebPart"
            TypeName="SampleWebPart" Safe="True" />
        </SafeControls>
      </Assembly>
    </Assemblies>
  </Solution>
```

There are three methods available for creating a solution cabinet file. The first method involves using the command line tool makecab.exe which can be found in both the Visual Studio 2005 SDK and the Windows system32 directory. The second method involves using the Visual Studio 2005 CAB Project setup and deployment project template, and the third method involves leveraging the Visual Studio 2005 extensions for WSS 3.0 by using the Deploy Solution build menu option. Let's take a look at how each of these methods would be used to build and deploy a solution file for a simple Web Part.

11.4.2 Building a Solution File using makecab.exe

Building a solution file using makecab.exe is a fairly straightforward process that involves copying all the solution components, such as Web Part assemblies, to their appropriate folder in a given folder structure, creating a solution manifest file at the root folder, creating a .ddf file, running makecab.exe referencing the .ddf file, and renaming the automatically generated .cab file with a .wsp extension. The steps for building a Web Part solution file are detailed as follows:

1. After building your Web Part solution in Visual Studio 2005, create a new folder with the folder structure you desire for your solution. In this example, we will keep all the files together in one folder (C:\DeploySampleWebPart), which will contain a Web Part assembly file and the solution manifest file. When deployed, the solution file will simply copy the Web Part assembly file to the global assembly cache and insert the appropriate Safe Control entry in the Web application's Web.config file. It will not upload the Web Part to the Web Part gallery. If we wanted to do this, we would need to include a .dwp or .Webpart file in our solution, in addition to some Feature element references to create a Feature that would provision the .dwp or .Webpart file into the Web Part

gallery. The examples shown in Chapter 16, Features, illustrate how to use Features to provision files to a SharePoint library.

2. Use Notepad to create the solution manifest file, and name it manifest.xml. Sample content of the manifest.xml file is shown in Figure 11.17. The SolutionID is a GUID that provides a unique identifier for the solution. You can use the GUID Generator (guidgen.exe) tool that comes with Visual Studio 2005 to create a unique identifier for your solution. Additionally, the PublicKeyToken refers to the key assigned to the assembly file. You can retrieve the assigned key using the strong name tool (sn.exe). Please refer to Chapter 13, Building and Deploying Web Parts, for more details.

3. Use Notepad to create the .ddf file, for example deploysampleWebpart.ddf. Sample content for the .ddf file is shown in Figure 11.18.

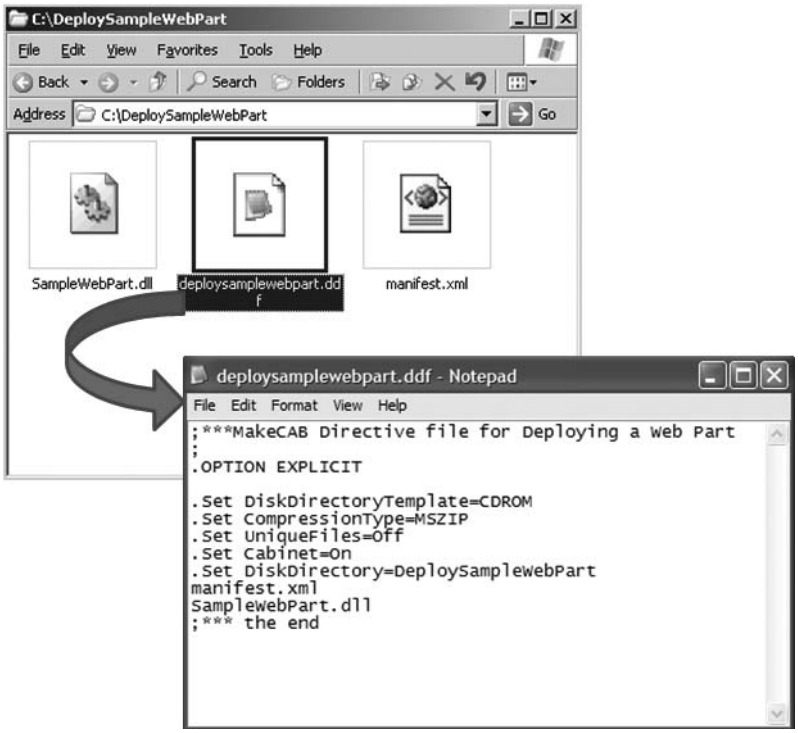


Figure 11.18
Solution files and DDF contents.

4. Run the makecab.exe using the following command:

```
makecab.exe /f deploysampleWebpart.ddf
```

5. The above command creates a .cab file for you; rename this file to a file with a .wsp extension, for example, deploysampleWebpart.wsp.

At this stage you have successfully created your solution file, and it's ready for deployment. Now let's look at some of the other build methods.

11.4.3 Building a Solution File Using Visual Studio CAB Project Template

Building a solution file using the Visual Studio 2005 CAB Project template involves adding the solution manifest file to the Web Part project and then creating a CAB project file in the same Visual Studio solution project. The end result is a .cab file that once again needs to be renamed to a .wsp file before it can be deployed. As with the previous example, when deployed the solution file will simply copy the Web Part assembly file to the global assembly cache and insert the appropriate Safe Control entry in the Web application's Web.config file. It will not upload the Web Part to the Web Part gallery. If we want to do this, we need to include a .dwp or .Webpart file in our solution, and add some feature element references to create a feature that would provision the .dwp or .Webpart file into the Web Part gallery. The steps involved are as detailed as follows:

1. Create your Web Part solution in Visual Studio 2005 and add the manifest.xml file shown in Figure 11.17 to the list of project files. Once again, the SolutionID is a GUID that provides a unique identifier for the solution. You can use the GUID Generator (guidgen.exe) tool that comes with Visual Studio 2005 to create a unique identifier for your solution. Additionally the PublicKeyToken refers to the key assigned to the assembly file. You can retrieve the assigned key using the strong name tool (sn.exe). Please refer to Chapter 13, Building and Deploying Web Parts, for more details.
2. Save the project, and then from the Visual Studio Build menu, click the Build option.
3. Keeping the Web Part solution open, from the File menu, click New, and then click Project. The New Project dialog appears (Figure 11.19).
4. Set the Solution field to Add to Solution and click OK.

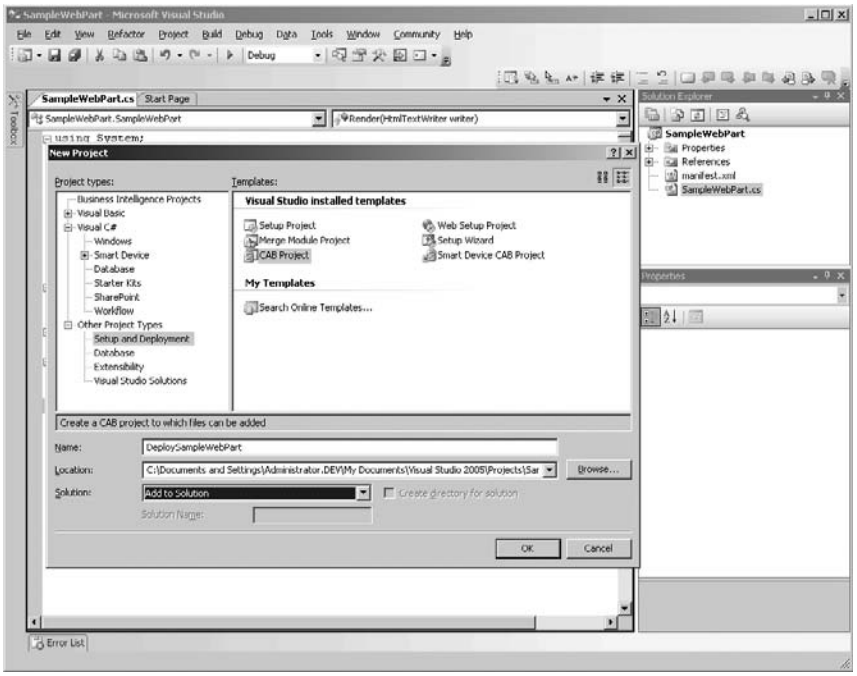


Figure 11.19
Creating the new cabinet project.

5. From the Solution Explorer window, right-click on the cabinet project.
6. Click Add, and then click Project Output.
7. From the drop-down list, ensure that the Web Part project is selected.
8. Select the file categories that you want from the list, making sure that you select Primary output and Content files.
9. Click OK.
10. Save and build the solution to create the cabinet file.
11. Using Windows Explorer, navigate to the cabinet file and rename the file to a file with a .wsp extension.

The solution file is ready for deployment.

11.4.4 Building a Solution File Using Visual Studio Extensions for WSS 3.0

Building a solution file using the Visual Studio 2005 extensions for WSS 3.0 is the most straightforward of all the methods mentioned in this chapter, because all the work is actually performed by Visual Studio and SharePoint. In fact, of all three examples, this particular method provides the most complete solution because it actually deploys the solution to SharePoint *and* installs the Web Part in the Web Part gallery. Granted, we could have implemented the appropriate Feature provisioning elements in the manifest.xml to automatically upload the Web Part to the Web Part gallery upon deployment, but the fact that the solution is automatically added to the solution store and deployed is very convenient. As far as steps go, there's really only one actionable step involved; the rest are basically checks to see that the solution was successfully deployed, but for the sake of completeness we'll list them anyway:

1. After building your Web Part in Visual Studio 2005, keep the development environment open and from the Build menu click the Deploy option. This kicks off the automated solution creation and deployment process. This process can take quite a bit of time, but the Visual Studio 2005 status bar provides feedback on where it is in the process. It iterates through steps such as deploy started, creating solution, adding solution, activating features, resetting IIS, recording macro, and deploy succeeded. Figure 11.20 shows a sample Web Part solution in the creating solution phase of the deploy cycle. The solution is now deployed; all that remains is for us to verify that everything is where it should be.
2. Using your browser, navigate to the Operations page in the SharePoint central administration site.
3. Under the Global Configuration section, click Solution management. The Solution Management page appears.
4. Locate your solution file in the list of solutions, and note that its status is set to deployed.
5. Navigate to your site's Web Part gallery and verify that the Web Part is listed and available to be added to Web Part pages within your site.
6. Using Windows Explorer navigate to the global assembly cache (usually resides in c:\windows\assembly). Verify that your Web Part appears in the global assembly cache.

7. Using Windows Explorer, navigate to the Web.config of your Web application and verify that it contains a SafeControl entry for your Web Part.

8. Using Windows Explorer, navigate to the `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\FEATURES` directory. Verify that there is a feature folder with the name of your Web Part appended with a GUID (for example, SampleWebPart_c81ae886-fae9-4d40-80c4-2ff39ddb9754). The Feature folder should be comprised of three files: a feature.xml file describing the Feature, an elementmanifest.xml file detailing the file to be provisioned, and the .Webpart file. The Feature folder is automatically generated by the Visual Studio solution generator and is used to upload the .Webpart file to the Web Part gallery, making the Web Part available for use as soon as the solution is deployed.

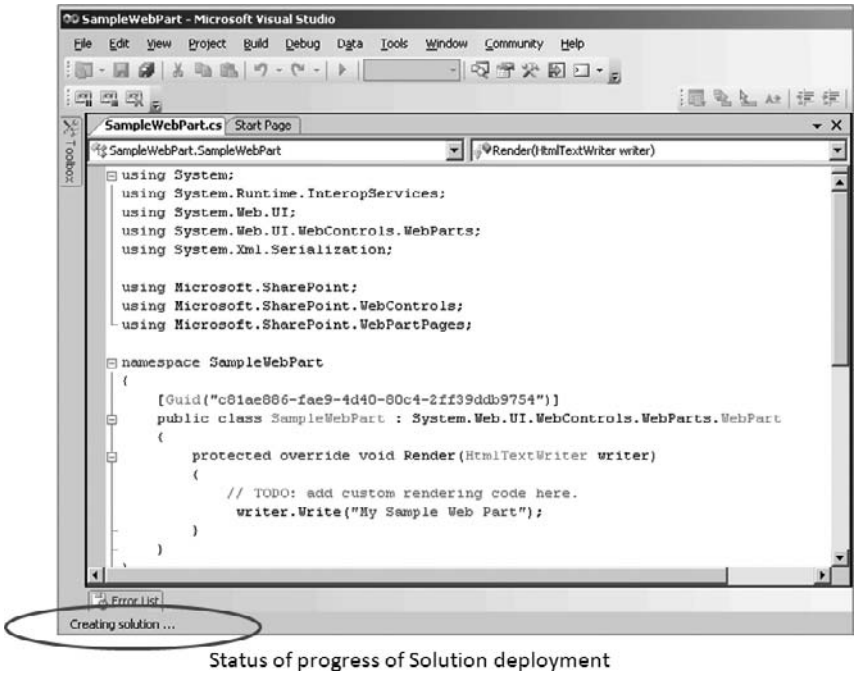


Figure 11.20
Using the Deploy option in Visual Studio.

Status of progress of Solution deployment

While the Deploy feature in Visual Studio is extremely convenient and bodes well for providing a consistent method for solution deployment, it does have a few drawbacks. These drawbacks may simply be limitations of V1.0 of the extensions and will hopefully be improved in subsequent releases. One such drawback is that the naming convention used when implementing features and site definitions appends the Solution ID GUID

to the name of the folder. While this technique always ensures uniqueness and portability of solutions, it doesn't work well for organizations that have naming standards that they wish to enforce. Another drawback is that the solution is deployed immediately, and there doesn't seem to be a way to schedule the actual deployment of the solution. It would be great to be able to add the solution to the solution store and let the system administrators schedule the deployment to run at a specific time. The current design of the Visual Studio Deploy functionality is really only suitable for development environments where the developer has the ability to both develop solutions as well as deploy them to SharePoint. In a true production environment, the Deploy functionality that comes with the Visual Studio 2005 extensions for WSS 3.0 would not be recommended; instead, you should revert to one of the previous techniques for creating a solution package that you can then hand over to the SharePoint administrator for scheduled deployment.

This chapter has provided a brief introduction into the world of SharePoint Server 2007 development. The remaining chapters will expand upon this overview and provide detail on such topics as how to program against the various SharePoint object models, build Web Parts, brand your SharePoint site, implement workflows, and work with InfoPath forms. Where appropriate, we have included some exercises to demonstrate the various concepts that a specific chapter explains. These exercises are typically located at the end of each chapter and are provided in a step-by-step format. While we try not to do all the work for you, there is plenty of guidance throughout.

This page intentionally left blank

12

SharePoint Programmability

Invariably, at some point during your SharePoint customization experience you will need to write custom code that manipulates content in the SharePoint content database. For example, you may need to develop a custom Web application that handles the automatic registration of users to your SharePoint site by creating AD accounts and assigning the membership to the appropriate AD groups and SharePoint groups; alternatively, you may need to create a custom application that provides users with a different interface for creating sites. Such custom applications typically require accessing SharePoint data programmatically.

This chapter acts as a launching pad for subsequent chapters by providing you with an overall understanding of the fundamental building blocks required to develop SharePoint applications. We start off by reviewing how to build custom applications that require a SharePoint context and make them available throughout your SharePoint farm. Next, we discuss the various methods for accessing SharePoint objects, such as using object models and Web services. Throughout the chapter, we take you through a variety of common scenarios that you may find useful in your SharePoint development projects, such as working with SharePoint lists and user profiles.

SharePoint development in and of itself is a huge topic, and this book merely skims the surface of the product's extensibility. We highly recommend that you sift through both the WSS 3.0 SDK and the Office SharePoint Server 2007 SDK, in addition to MSDN, for more detailed information and lots of sample code.

12.1 Custom Web Pages and Web Applications in SharePoint

As we progress through this book, there will be times when we will leverage Web Parts to demonstrate a particular topic; at other times we will create custom .aspx pages that run in SharePoint. When customizing your SharePoint

deployment, you will most likely run into the same situation, in which some of your customizations are presented to your users through custom Web Parts and others are presented through custom Web pages. You may require the Web pages to be accessible to all the sites in your SharePoint farm, or you may require them to run in a custom Web application while still using the context of SharePoint. Chapter 13, Building and Deploying Web Parts, explains how to develop custom Web Parts, but for now let's look at how to build custom Web pages and Web applications in SharePoint.

Making your Web pages accessible to all the sites in your SharePoint deployment simply involves deploying the pages to the `\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\TEMPLATE\LAYOUTS` directory, also referred to as the _layouts directory. Pages stored in this directory are accessible to sites in both a Windows SharePoint Services 3.0 deployment and a SharePoint Server 2007 deployment, using the URL format shown in Figure 12.1.

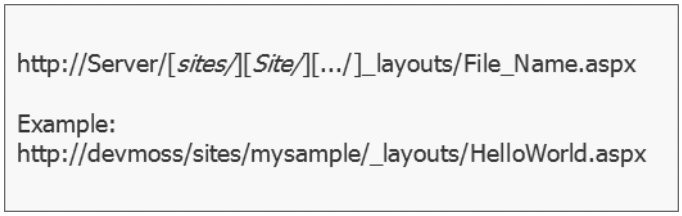


Figure 12.1 *URL format for accessing pages in _layouts.*

A great advantage that you have when you deploy pages to the _layouts directory is that the pages automatically have access to the current SharePoint context. This means that you can create custom Web pages that behave differently depending on which site, or even which user, accesses the page. For example, as part of the code in your custom Web page, you could set the master page file for your page to be the same as the master page setting for the current Web site. Given that Web sites can have different master page file settings, your custom page will present a different master page depending on the site from which it is accessed.

12.1.1 Custom Web Pages

We will walk you through some sample code that performs changes to the master page file later in the chapter. For now, let's walk through a simple Hello World example. In this example, we create a very basic aspx page that contains the heading text "Hello World."

1. Launch Visual Studio, create a basic .aspx page and add "Hello World" text into the page body. Figure 12.2 shows suggested sam-

ple code for the "Hello World" Web page. In fact, you can create the .aspx in Notepad if you wish, as the code is very straightforward.

2. Save the page to your project directory.
3. Copy the page from your project directory to the _layouts directory of your SharePoint 2007 or WSS 3.0 server, with the name HelloWorld1.aspx.
4. Launch your browser, navigate to a sample team site, and append `/_layouts/HelloWorld1.aspx` to the site URL; for example `http://devmoss/_layouts/HelloWorld1.aspx`.

Figure 12.2 also shows the Web page that is displayed when you navigate to the HelloWorld1.aspx file using the _layouts URL. If we were to access a different SharePoint Web site and append the same `_layouts/HelloWorld1.aspx` to the URL, the same file would be displayed. This is convenient for situations in which you would like all Web sites to have access to the same custom Web page. For example, let's say that you wanted to use a different form when uploading documents to the document library, and you wanted the new form to be accessible from all Web sites. You could copy this form to the _layouts directory, and then update the menu links in SharePoint to provide an option to launch the new form.

Figure 12.2 *HelloWorld1.aspx in _layouts directory.*

12.1.2 Best Practices for Creating Custom Web Pages

Now that we have seen how easy it is to add a custom Web page in SharePoint, there are a few best practices we should discuss that which will ease your development experience. In our previous example, we copied the custom Web page to the _layouts directory. The problem with this technique is that it increases the likelihood that the file will be overwritten by Microsoft or a third-party vendor, if they use the same file name. Therefore, it is recommended that all _layouts customizations be deployed in a subfolder of the _layouts directory. If you decide to store your custom application pages in a separate subfolder, the URL that you use to access the page should also include the subfolder. For example: http://devmoss/_layouts/CustomApps/CustomPage.aspx.

When developing your custom Web pages using Visual Studio 2005, a good habit to develop is to create a Visual Studio project that contains the `\TEMPLATE\_layouts\` hierarchy, and then create a subfolder in which to store your custom Web pages. This means that you can manage all your custom Web pages from one Visual Studio project. And, since the deployment process for your custom Web pages is to copy the custom application subfolder and its contents to the _layouts directory, another good practice is to create an installation file that runs as part of the Visual Studio project build process and performs the necessary copying. Figure 12.3 shows an example of a Visual Studio 2005 project which contains the _layouts folder hierarchy and several custom Web pages stored in a subfolder named CustomApps. The project also includes an install.bat file, which copies the contents of the folder hierarchy to the SharePoint `\TEMPLATE\_layouts` directory. The install.bat command is executed during the post-build event process; the post-build event commands to initiate its execution are shown in Figure 12.4.

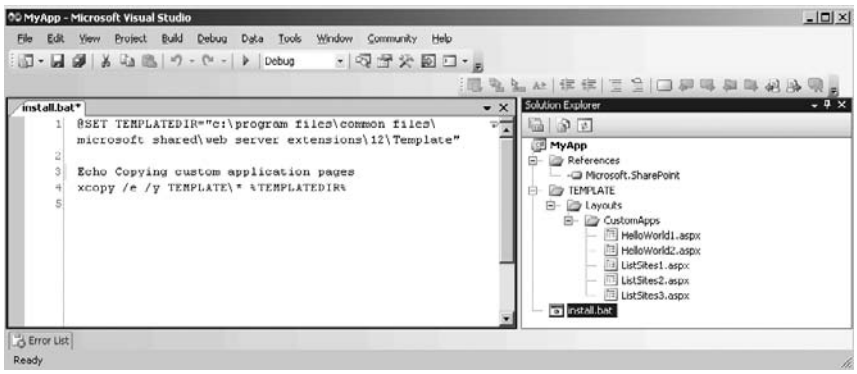


Figure 12.3
Custom application pages in Visual Studio 2005 (C#).

Figure 12.4
Post-build event commands.

12.1.3 Accessing the SharePoint Object Model with Inline Code

For our next example we'll add some logic to the page that will list the names of all the subsites of the current site. Figure 12.5 shows the code for a custom Web application page named ListSites1.aspx, and Figure 12.6 shows the result of the code displayed in the browser. More detail on the various SharePoint objects being used in this particular example is covered later in the chapter. For now, we'll focus on how to prepare a custom Web page so that it can leverage the object model.

To take advantage of the IntelliSense capabilities of Visual Studio 2005 when working with the SharePoint object model within a custom Web page, you first need to add the `@Assembly` directive to reference the `Microsoft.SharePoint` assembly. Line 1 of the ListSites1.aspx code listing shows the `@Assembly` directive code. The custom Web page will function fine without this directive, but the development experience would be suboptimal. Lines 4 and 5 contain two import statements that import the namespaces that we use to retrieve and display the list of sites. The `Microsoft.SharePoint` namespace provides access to SharePoint top-level sites and subsites through the object model. Using this namespace, you can access the current Web site that the user is browsing (line 9). The `Microsoft.SharePoint.Utilities` namespace provides access to various classes that perform a utility-type function. For example, the `SPEncode` class provides methods for encoding strings, such as the `HtmlEncode` method which encodes the specified string so that special characters used in HTML are encoded as HTML entities (line 10).

The inline code that performs all the retrieval work is shown in the script block (lines 7 through 30). The `ListBox` control defined in line 38 is accessible to the script block code using the control `ID`. This means that when we retrieve the list of sites in the current site collection, they can be added to the `ListBox` control.

Figure 12.5
ListSites1.aspx inline code (C#).

```
<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c"%>
<%@ Page Language="C#"  %>

<%@ Import Namespace="Microsoft.SharePoint" %>
<%@ Import Namespace="Microsoft.SharePoint.Utilities" %>

<script runat="server">
   protected override void OnLoad(EventArgs e) {
       SPWeb curWeb = SPContext.Current.Web;
       Response.Write("<H1>List Sub Sites of " + SPEncode.HtmlEncode(curWeb.Title) + "</H1><br/>");
       ListSites(curWeb);
   }

     protected void ListSites(SPWeb cWeb)
     {
         SPWebCollection webColl = cWeb.Webs;
         lstWebs.Items.Clear();
         if (webColl.Count == 0)
         {
             lstWebs.Items.Add("No current sub webs");
         }
         else
         {
             foreach (SPWeb subWeb in webColl)
             {
                 lstWebs.Items.Add(subWeb.Title);
             }
         }
     }
</script>
<html xmlns="http://ww.w3.org/1999/xhtml">
<head runat="server">
    <title>List Sites</title>
</head>
<body>
    <form id="form1" runat="server">
     <div>
       <asp:ListBox ID="lstWebs"  Rows="20" Width="50%" runat="server"/>
     </div>
    </form>
</body>

</html>
```

Figure 12.6
ListSites1.aspx display in browser.

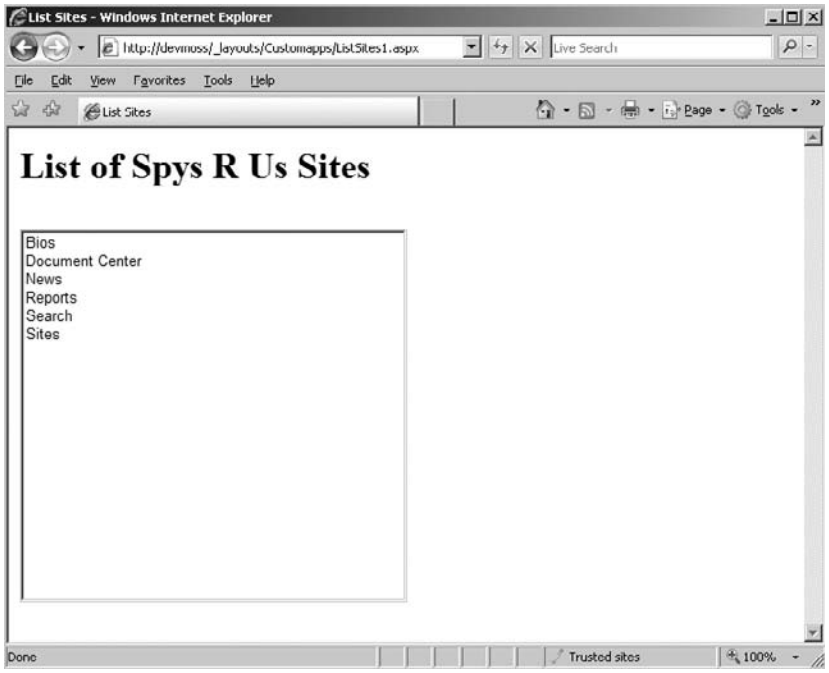


Figure 12.7 *ListSites1.aspx Accessed from Different Site*

In Figure 12.6, the URL being used to access the custom Web page contains the new _layouts subfolder. If we were to navigate to another Web site, and access the same _layouts page, we are presented with a different list of sub sites (Figure 12.7). Depending on your code, custom Web pages can perform different functions, or present different data, from one site to another.

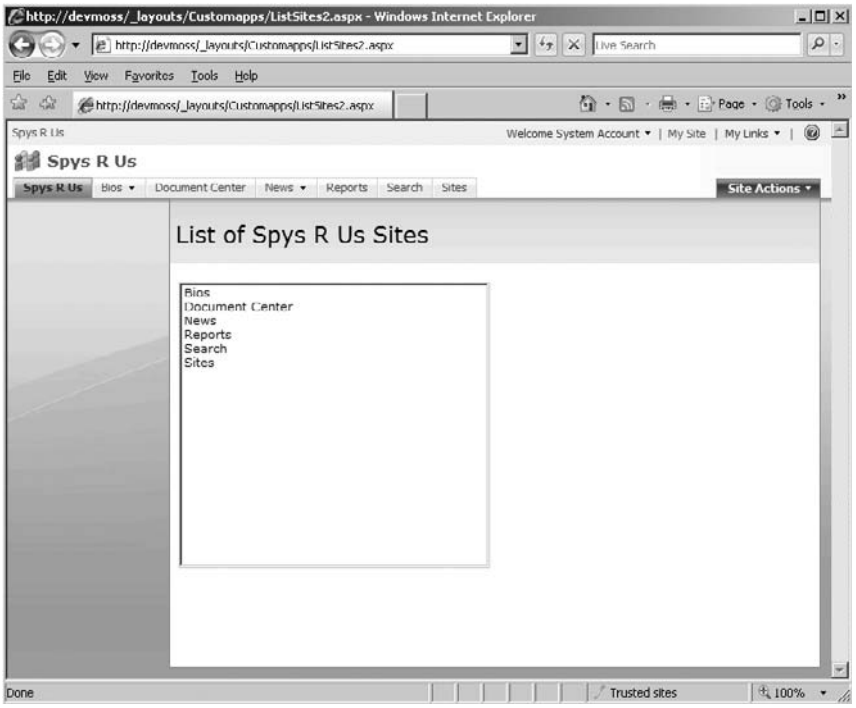
12.1.4 Applying the Layouts Branding

The default files stored in the _layouts directory are known as application pages, and as such share a common master page named application.master. For a custom application page to share the same branding as the Microsoft application pages, it needs to include a reference to the master page file, along with the appropriate content placeholders. Information on the inner workings of master pages and content placeholders can be found in Chapter 14, Branding your SharePoint 2007 Sites. Figure 12.8 shows the code listing for the file ListSites2.aspx, and Figure 12.9 shows the result of this code displayed in a browser. Lines 2 and 3 of the code listing show the page derived from the `Microsoft.SharePoint.WebControls.LayoutsPageBase` class, which is the base class used for SharePoint application pages and provides various methods with respect to security. Although recommended, this inheritance is not required for the page to function; if the inherit statement is omitted, the layout branding will still be picked up from the application.master reference. In fact, you could easily exchange this reference with any class that derives from the `Page` class (`Microsoft.System.Web.UI.Page`). The `Page` class provides the basic behavior for the objects that .aspx pages use. The `LayoutsPageBase` and `UnsecuredLayoutsPageBase` classes override some of the base `Page` methods, and provide additional methods and properties specific to working with SharePoint _layouts pages. For example, Figure 12.10 shows the properties and methods associated with the `LayoutPageBase` class, viewed using the Visual Studio object browser. The highlighted Boolean property `RequireSiteAdministrator` can be overridden and set to true to limit the access of your _layouts pages to site administrators only. Line 2 of the code listing also shows the

Figure 12.8
ListSites2.aspx inline code with branding (C#).

```
<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c"%>
<%@ Page Language="C#" MasterPageFile="~/_layouts/application.master"
        Inherits="Microsoft.SharePoint.WebControls.LayoutsPageBase"  %>

<%@ Import Namespace="Microsoft.SharePoint" %>
<%@ Import Namespace="Microsoft.SharePoint.Utilities" %>

<script runat="server">
    protected override void OnLoad(EventArgs e)
    {
        SPWeb curWeb = SPContext.Current.Web;
        lblSiteTitle.Text = curWeb.Title;
        ListSites(curWeb);
    }

    protected void ListSites(SPWeb cWeb)
    {
        SPWebCollection webColl = cWeb.Webs;
        lstWebs.Items.Clear();
        if (webColl.Count == 0)
        {
            lstWebs.Items.Add("No current sub webs");
        }
        else
        {
            foreach (SPWeb subWeb in webColl)
            {
                lstWebs.Items.Add(subWeb.Title);
            }
        }
    }
</script>
<asp:Content ID="PageTitleInTitleArea" runat="server"
             contentplaceholderid="PlaceHolderPageTitleInTitleArea" >
  List of <asp:Label ID="lblSiteTitle" runat="server"/> Sites <br />

</asp:Content>

<asp:Content ID="Main" contentplaceholderid="PlaceHolderMain" runat="server">
   <asp:ListBox ID="lstWebs"  Rows="20" Width="50%" runat="server"/>
</asp:Content>
```

Figure 12.9
ListSites2.aspx Display in browser.



required reference to the application.master file, which is located in the _layouts directory. Lines 33 through 41 define the content to be presented in the content placeholders. In this example, we are using the content placeholders, named `PlaceHolderPageTitleInTitleArea and PlaceHolderMain`.

To apply a *custom* application master page to all your custom application pages, simply save the custom application master page in the _layouts directory, update the Web page to include the appropriate placeholders, and then update the `MasterPageFile` reference to point to the custom application master page. For example, `MasterPageFile="~/_layouts/corporate.master"`.

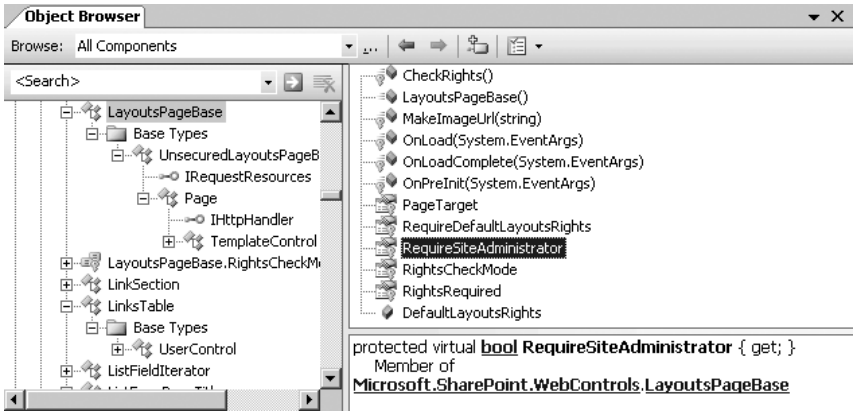


Figure 12.10 *LayoutsPageBase properties and methods.*

12.1.5 Inheriting Master Page from Current Site

What happens if, instead of using a master page in the _layouts directory, you want your custom Web page to inherit the master page settings from the current site? This particular scenario is slightly more complex, because you cannot leverage the tokens such as `~masterurl/default.master` and `~site/default.master` in _layouts pages. However, the default master setting and the custom master setting can be accessed through the object model by using the `MasterUrl` and `CustomMasterUrl` properties of the `SPWeb` object. We used this technique in our Feature receiver example in Chapter 16, Features. Setting the custom Web page to inherit the master page from the current site involves retrieving the `CustomMasterUrl` from the current Web site and applying it to the `MasterPageFile` attribute of the current page. The `MasterPageFile` attribute must be set in the `OnPreInit` event, prior to the loading of the page. Lines 9 through 14 of the code listing shown in Figure 12.11 show sample code that assigns the current Web site's custom master page setting to the custom Web page being accessed.

Figure 12.11
ListSites3.aspx inheriting current site's master page (c#).

```
<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c"%>
<%@ Page Language="C#" %>

<%@ Import Namespace="Microsoft.SharePoint" %>
<%@ Import Namespace="Microsoft.SharePoint.Utilities" %>

<script runat="server">

    protected override void OnPreInit(EventArgs e)
    {
        SPWeb curWeb = SPContext.Current.Web;
        string curMasterPage = curWeb.CustomMasterUrl;
        this.MasterPageFile = curMasterPage;
    }

    protected override void OnLoad(EventArgs e)
    {
        SPWeb curWeb = SPContext.Current.Web;
        lblSiteTitle.Text = curWeb.Title;
        ListSites(curWeb);
    }

    protected void ListSites(SPWeb cWeb)
    {
        SPWebCollection webColl = cWeb.Webs;
        lstWebs.Items.Clear();
        if (webColl.Count == 0)
        {
            lstWebs.Items.Add("No current sub webs");
        }
        else
        {
            foreach (SPWeb subWeb in webColl)
            {
                lstWebs.Items.Add(subWeb.Title);
            }
        }
    }
</script>
<asp:Content ID="PageTitleInTitleArea" runat="server"
             contentplaceholderid="PlaceHolderPageTitleInTitleArea" >
  List of <asp:Label ID="lblSiteTitle" runat="server"/> Sites <br />

</asp:Content>
```

Figure 12.12
ListSites3.aspx display in browser.

Figure 12.12 shows the result of the ListSites3.aspx code displayed in a browser. Notice that while the custom Web page displays the same content, its branding has changed to reflect the same master page setting as the current Web site. Figure 12.13 shows the same custom Web page accessed from different sites within the SharePoint farm; each site shown in our example is

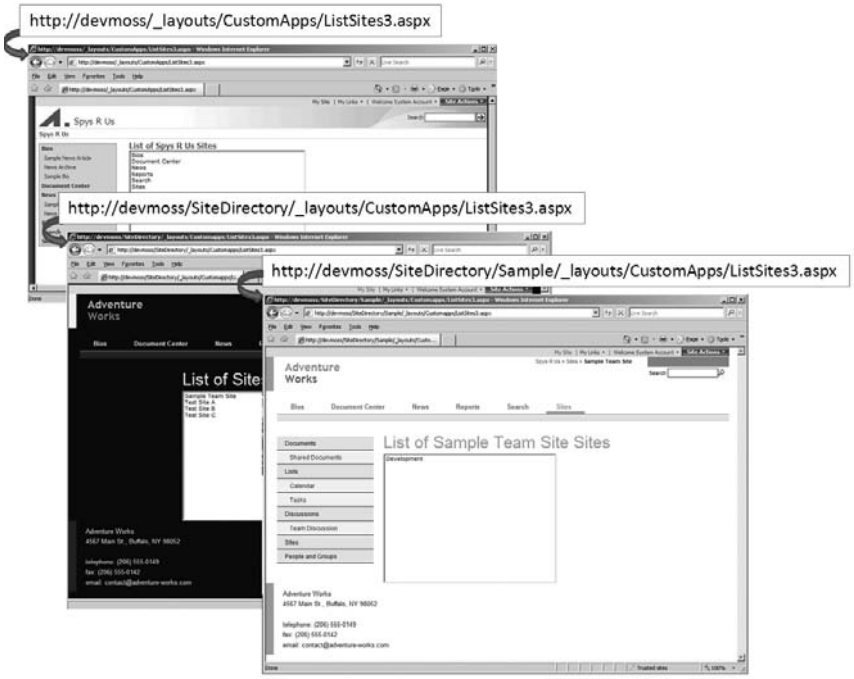


Figure 12.13
ListSites3.aspx accessed from different sites.

set to a different custom master page. As you can see, the custom Web page changes its branding dynamically according to the site context.

12.1.6 Storing Code behind Custom Web Pages

Developing Web pages using inline code is a convenient method for getting your pages up and running as quickly as possible, allowing you to incorporate both the code logic and the presentation logic in a single file. However, this technique can become unwieldy when you have a lot of code within the script block, and particularly when you have more than one developer writing the code for the custom Web page. Another method involves separating the code logic from the presentation logic by including the code in another file, known as a code-behind file. With code-behind files, the separation occurs in the context of two files. The Web form, or .aspx page, contains the presentation layer and references an additional class file that contains all code associated with the Web form. The class file is generally given the same name as the Web form, with an extension suffix of .cs or .vb depending on which programming language is used. For example, if we have a Web form named ListSites.aspx and we are using C# as our programming language, then its corresponding code-behind file is named ListSites.aspx.cs.

The code-behind file can be stored as a standalone file and referenced in the Web form using the `CodeFile` attribute. Using this method, the code file contains a partial class, meaning that it contains a portion of the code that comprises the full class of the page. The partial class inherits from a base page class which, in the case of SharePoint _layouts pages, is the `LayoutsPageBase` class. The .aspx file references the partial class using the `Inherits` attribute of the `@ Page` directive. In addition to the partial class, the ASP.NET compiler generates another class, which inherits from the partial class and contains the code required to build the page. ASP.NET automatically compiles the application code and any associated resources the first time the page is requested by a user. ASP.NET compiles both classes into a single assembly that is loaded into the application domain and ultimately renders the output to the user in the browser. Aside from the improved manageability and maintainability of code, the code-behind method promotes the reusability and portability of code; classes that you develop for one Web form can be leveraged and reused by another.

Alternatively, instead of using a partial class, you can create a complete class, compile it into an assembly file, and deploy the assembly file into the global assembly cache. Once the assembly file is deployed, you can reference the assembly and the appropriate class in your .aspx pages using the `@ Assembly` directive and the Inherits attribute of the `@ Page` directive respectively. This method provides the added benefit of allowing you to manage and maintain all your classes from a single assembly. The downside of this method is that if the version of your assembly changes, you must update the `@ Assembly` directive in all .aspx pages that reference the assembly. This weakness can be overcome by inserting a reference to the assembly in the `assemblies` section of the Web.config.

In our next example we will use the latter technique and create a separate assembly file that we will reference in our .aspx page. Let's begin.

12.1.7 Create the Assembly File

1. Using Visual Studio, create a new C# Class library project.
2. Add the code shown in Figure 12.14 to the class file, and rename the class file as desired; for example, ListSites4.cs.
3. Strongly name the assembly using the Signing tab on the project properties dialog.
4. Save the project and build the assembly.
5. Deploy the assembly .dll file to the global assembly cache by copying the file from the projects folder to the \WINDOWS\ assembly folder.

The code is very similar to the code listing contained within the script block of our inline example. The class derives from the `LayoutsPageBase` class, as shown in line 10. Lines 12 and 13 are new additions to the code and perform the function of mapping the `System.Web.UI.WebControls`, such as the `ListBox` control and the `Label` control, directly to the same controls on the .aspx page.

Figure 12.14
ListSites4.cs code listing (C#).

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Web.UI.WebControls;
5  using Microsoft.SharePoint;
6  using Microsoft.SharePoint.WebControls;
7
8  namespace SpysRUs
9  {
10     public class ListSites4:LayoutsPageBase
11     {
12         protected ListBox lstWebs;
13         protected Label lblSiteTitle;
14
15         protected override void OnPreInit(EventArgs e)
16         {
17             SPWeb curWeb = SPContext.Current.Web;
18             string curMasterPage = curWeb.CustomMasterUrl;
19             this.MasterPageFile = curMasterPage;
20         }
21
22         protected override void OnLoad(EventArgs e)
23         {
24             SPWeb curWeb = SPContext.Current.Web;
25             lblSiteTitle.Text = curWeb.Title;
26             ListSites(curWeb);
27         }
28
29         protected void ListSites(SPWeb cWeb)
30         {
31             SPWebCollection webColl = cWeb.Webs;
32             lstWebs.Items.Clear();
33             if (webColl.Count == 0)
34             {
35                 lstWebs.Items.Add("No current sub webs");
36             }
37             else
38             {
39                 foreach (SPWeb subWeb in webColl)
40                 {
41                     lstWebs.Items.Add(subWeb.Title);
42                 }
43             }
44         }
45     }
46 }
47
```

12.1.8 Create the Web Form

1. Using your editor of choice, such as Visual Studio 2005 or Notepad.exe, create a Web form named ListSites4.aspx.
2. Add a reference to the Microsoft.SharePoint and System.Web assemblies.
3. Add the code shown in Figure 12.15 to the .aspx file, and save the file. Replace the `PublicKeyToken` attribute of the assembly name with the public key token of your assembly file. You can find this value by examining the properties of the file in the global assembly cache.
4. Copy the file to your custom folder in the _layouts directory.
5. Navigate to the page using your browser (e.g. `http://devmoss/_layouts/CustomApps/ListSites4.aspx`)

Figure 12.15 *ListSites4.aspx code listing (C#).*

```
1  <%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
    PublicKeyToken=71e9bce111e9429c"%>
2  <%@ Assembly Name="SpysRUs, Version=1.0.0.0, Culture=neutral,
   PublicKeyToken=31a35a3c7f393ca9"%>
3  <%@ Page Language="C#" Inherits="SpysRUs.ListSites4" %>
4
5  <asp:Content ID="PageTitleInTitleArea" runat="server"
6               contentplaceholderid="PlaceHolderPageTitleInTitleArea" >
7    List of <asp:Label ID="lblSiteTitle" runat="server"/> Sites <br />
8  </asp:Content>
9
10 <asp:Content ID="Main" contentplaceholderid="PlaceHolderMain" runat="server
   ">
11   <asp:ListBox ID="lstWebs"  Rows="20" Width="50%" runat="server"/>
12 </asp:Content>
```

Notice that in Line 2 we now have an additional `@ Assembly` directive that links the application page to the SpysRUs assembly .dll stored in the global assembly cache. Line 3 adds a reference to the specific class within the assembly that we want our page to inherit, which in this instance is the `SpysRUs.ListSites4` class. The ID of the Web controls defined in lines 7 and 11 matches the `ListBox` and `Label` declarations in the class file.

12.1.9 Custom Web Applications

In addition to creating .aspx pages in the _layouts directory, you can create entire Web applications in the same location. This time, your Web application, with all its encompassing pages, is accessible to all the sites in your SharePoint implementation via the _layouts URL. Creating a Web application involves the same steps as those described for the .aspx page, except on this

occasion you create a Web site in Visual Studio instead of a Web form. If your Web application focuses on manipulating the global SharePoint settings of your SharePoint deployment, and your code only makes use of the `Microsoft.SharePoint.Administration` namespace, then you should deploy the Web application on the administrative port, in the `\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\TEMPLATE\ADMIN` directory. The administrative port is basically where all the central administration pages are located, and pages stored in this directory are accessible by the URL format shown in Figure 12.16. For a quick demonstration, copy the HelloWorld.aspx file that you saved to the _layouts directory to the _admin directory. Next, launch your browser and navigate to the central administration page. Finally, append `_admin/HelloWorld.aspx` to the server/port URL. The Hello World page should display.

Figure 12.16 *URL format for accessing pages in _admin.*

Creating a Web application is relatively straightforward, but there are a few details that you need to pay particular attention to when deploying Web applications that run in the context of a SharePoint application. The following method defines the steps to create a Web application in the SharePoint _layouts directory. We will continue with the same example of listing the subsites of the current site and changing the branding of the page so that it reflects the current branding of the site from which the page is accessed.

1. Launch Visual Studio 2005 and create a Web site project in a local project directory. Creating the Web site project in a local directory first allows you to rework your code until it compiles successfully, after which it can be deployed to the _layouts or _admin folder.
2. Add a reference to the `Microsoft.SharePoint` and `System.Web` assemblies.
3. Open the default.aspx page and insert the code listed in Figure 12.17.
4. Open the code-behind file for your application's default.aspx. The code-behind file should be named default.aspx.cs. If you wish,

Figure 12.17
Default.aspx code (C#).

```
<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c"%>

<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<asp:Content ID="PageTitleInTitleArea" runat="server"
             contentplaceholderid="PlaceHolderPageTitleInTitleArea" >
  List of <asp:Label ID="lblSiteTitle" runat="server"/> Sites <br />
</asp:Content>

<asp:Content ID="Main" contentplaceholderid="PlaceHolderMain" runat="server">
  <asp:ListBox ID="lstWebs"  Rows="20" Width="50%" runat="server"/>
</asp:Content>
```

you can change the names of the .aspx page and the corresponding code-behind file.

5. Add the code listed in Figure 12.18 to the code-behind file.
6. Save and build the project. The project should compile successfully without build errors. Keep the Visual Studio project open.
7. Using Windows Explorer navigate to the _layouts directory.
8. Create a new folder in the _layouts directory to host your Web site.
9. Go back to Visual Studio and from the Website menu, click Copy Web Site.
10. From the Connections field, double click Connect to…
11. Browse to the folder you created in Step 8.
12. Click Yes when prompted "The Web <folder> is not marked as an application in IIS. Do you want to continue?"
13. From the Source Files list box on the left, select all files *except* the Web.config file. Since we are operating in the context of SharePoint, we can use the Web.config from the parent application. This helps us avoid errors such as the virtual path error: "The virtual path …maps to another application, which is not allowed."
14. Copy the selected files to the destination list box on the right by clicking the right-arrow, as shown in Figure 12.19.
15. You are now ready to test your application. Launch your browser and navigate to a SharePoint team site; for example, http://devmoss/sitedirectory/Sample.
16. Navigate to your application's default page by appending the _layouts URL to the site URL; for example: http://devmoss/sitedirectory/Sample/_layouts/SpysRUs/default.aspx

17. Navigate to the application's default page from the root of the site collection; for example: http://devmoss/ _layouts/SpysRUs/default.aspx

Figure 12.18
Default.aspx.cs code for Web application (C#).

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;

public partial class _Default : LayoutsPageBase
{

    protected override void OnPreInit(EventArgs e)
    {
        SPWeb curWeb = SPContext.Current.Web;
        string curMasterPage = curWeb.CustomMasterUrl;
        this.MasterPageFile = curMasterPage;
    }

    protected override void OnLoad(EventArgs e)
    {
        SPWeb curWeb = SPContext.Current.Web;
        lblSiteTitle.Text = curWeb.TitleA;
        ListSites(curWeb);
    }

    protected void ListSites(SPWeb cWeb)
    {
        SPWebCollection webColl = cWeb.Webs;
        lstWebs.Items.Clear();
        if (webColl.Count == 0)
        {
            lstWebs.Items.Add("No current sub webs");
        }
        else
        {
            foreach (SPWeb subWeb in webColl)
            {
                lstWebs.Items.Add(subWeb.Title);
            }
        }
    }
}
```

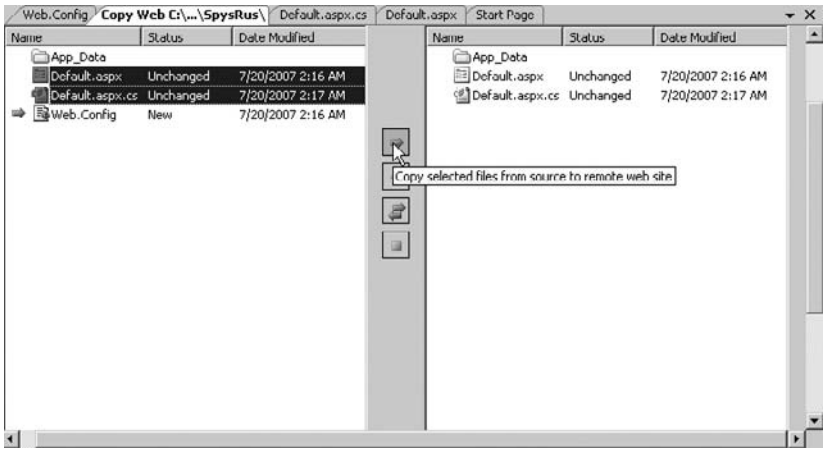


Figure 12.19
Copy Web Site Files to _Layouts Folder

The code listed in this example is very similar to our previous example. The only real difference is that in this example, we are implementing the code-behind technique that uses a partial class. Line 3 of the code listing in Figure 12.17 shows the `@ Page` directive that uses the `CodeFile` attribute to reference the code-behind file and the `Inherits` attribute to reference the partial class. Line 13 of the code listing in Figure 12.18 shows the partial class declaration. This Web application example produces the same output as our previous example, shown in Figure 12.13.

12.2 The Object Model

Both WSS 3.0 and SharePoint Server 2007 provide object models that expose various classes, methods, and properties that allow developers to perform many tasks programmatically. For example, in our previous section we showed how you can leverage the object model to access the custom master page setting of the current Web site and retrieve a list of all the subsites of that site. You can use the object model to create sites, lists, and items; apply custom templates and headers; initiate a workflow process; and even perform administrative functions such as adding and removing components (a solution package) from the server farm; the possibilities are endless. While we can't go into detail about all the classes, methods, and properties exposed by the object models (this information is available in the SDKs), we will provide you with enough information to get you started with your own exploration.

12.2.1 WSS 3.0 Object Model

The WSS 3.0 object model is accessible through the Microsoft.SharePoint.dll assembly and contains a host of namespaces which you can work in your code.

Table 12.1 lists a subset of the available namespaces, which should give you an idea of the customization opportunities available through the object model.

Table 12.1 *Microsoft.SharePoint Namespaces (Subset)*

Namespace	Description
Microsoft.SharePoint	Provides access to the classes, methods, and properties of many SharePoint objects, including top-level sites, subsites, lists, folders, files, content types, list items, and fields.
Microsoft.SharePoint.Administration	Provides access to the classes, methods, and properties of the SharePoint administrative objects, including the content database, the configuration database, the SharePoint timer service, and the SharePoint solution collection.
Microsoft.SharePoint.Navigation	Provides access to the classes, methods, and properties of the SharePoint objects with which you can customize the navigational structure and the site maps associated with a SharePoint site.
Microsoft.SharePoint.WebControls	Provides access to the server controls that can be used on site and list pages in SharePoint, such as the `LayoutsPageBase` class used in our previous examples.
Microsoft.SharePoint.WebPartPages	Provides access to the classes, methods, and properties of for creating custom SharePoint Web Parts.
Microsoft.SharePoint.Workflow	Provides access to the classes, methods, and properties that represent the WSS 3.0 workflow functionality.

The namespace in which you will likely spend most of your time is the `Microsoft.SharePoint` namespace, as it is the namespace that provides access to all the classes, methods, and properties for working with sites, subsites, lists, and even content types. A great way to get an insight into the object model is by looking at a graphical representation of the SharePoint hierarchy, with respect to the objects that you may want to work with. Figure 12.20 illustrates the `Microsoft.SharePoint` object model hierarchy; the figure only shows a subset of the available objects and classes. The top-level classes in the hierarchy are the `SPSite` class and the `SPWeb` class. The `SPSite`

class represents the collections of sites, including the root site, and is used to manage existing Web sites and for accessing site and list data. The `SPWeb` class represents an individual Web site, and is used to work with the lists, files, and security associated with the site. Using the `SPWeb` class, you can work with the users of the site and their corresponding groups and permissions.

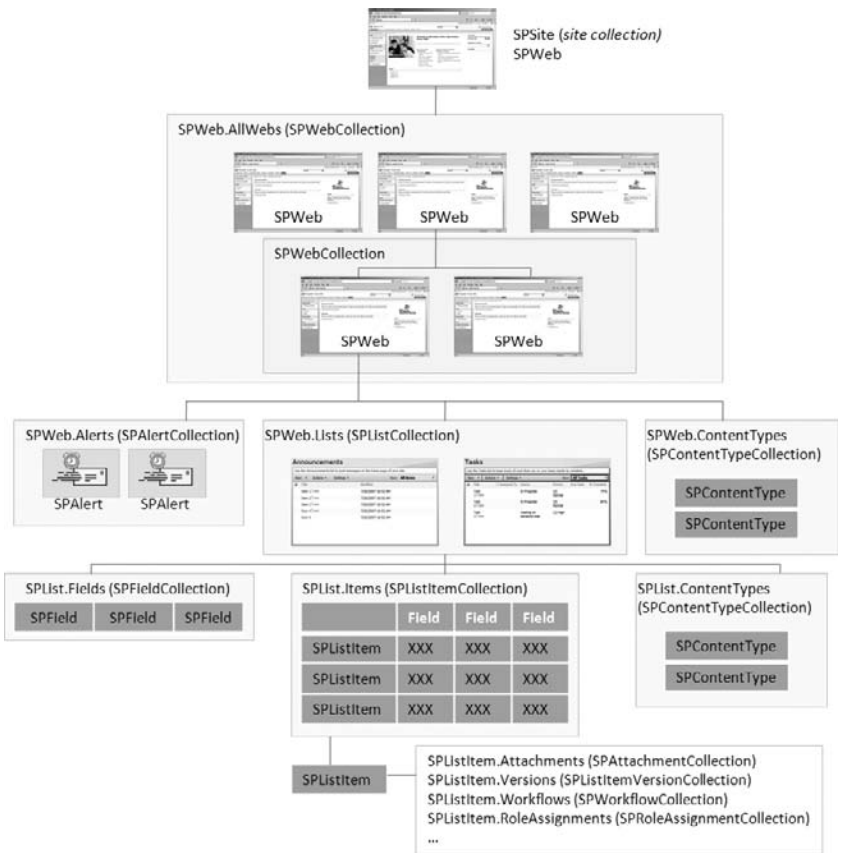


Figure 12.20 *Microsoft.SharePoint Object Hierarchy*

12.2.2 Retrieving SharePoint Data

Figure 12.21 shows how to use the `SPSite` class to retrieve a list of the Features that are currently activated at the site collection level. Figure 12.22 shows the resulting output from this code. The custom Web page we are using in this example is stored in the _layouts directory; therefore, we have access to the current SharePoint context. To access the current site collection, we can leverage the `Microsoft.SharePoint.SPContext` object. The `SPContext` object represents the `HTTP` context. Line 23 of the code listing shows that we are instantiating an `SPSite` object with the current site collec-

tion. A few lines prior to this, in the `PreInit` method, we have the code we used in our earlier examples to change the master page of our application page. Line 6 of this code uses the `SPContext` object to get the current Web site (`SPWeb`) object, as opposed to the site collection (`SPSite`) object. Using the returned `SPWeb` object, we can go on to access the properties of the current Web site, which in this instance is the custom master setting. So let's get back to the `SPSite` class. Now that we have the current `SPSite` object, we can retrieve the Features currently activated within that site collection. The `SPFeature` class is a member of the `SPSite` class and returns the collection of currently activated Features (line 29). The code listed in lines 37 through 40 iterates through the collection of Features and, for each Feature, displays the display title of the Feature.

Figure 12.21
ListFeatures.cs code listing.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Web.UI.WebControls;
5  using Microsoft.SharePoint;
6  using Microsoft.SharePoint.WebControls;
7
8  namespace SpysRUs
9  {
10     public class ListFeatures:LayoutsPageBase
11     {
12         protected ListBox lstFeatures;
13
14         protected override void OnPreInit(EventArgs e)
15         {
16             SPWeb curWeb = SPContext.Current.Web;
17             string curMasterPage = curWeb.CustomMasterUrl;
18             this.MasterPageFile = curMasterPage;
19         }
20
21         protected override void OnLoad(EventArgs e)
22         {
23             SPSite curSiteColl = SPContext.Current.Site;
24             displayFeatures(curSiteColl);
25         }
26
27         protected void displayFeatures(SPSite cSiteColl)
28         {
29             SPFeatureCollection siteFeatures = cSiteColl.Features;
30             lstFeatures.Items.Clear();
31             if (siteFeatures.Count == 0)
32             {
33                 lstFeatures.Items.Add("No current Features");
34             }
35             else
36             {
37                 foreach (SPFeature spFeature in siteFeatures)
38                 {
39                     lstFeatures.Items.Add(spFeature.Definition.DisplayName);
40                 }
41             }
42         }
43     }
44 }
45
```

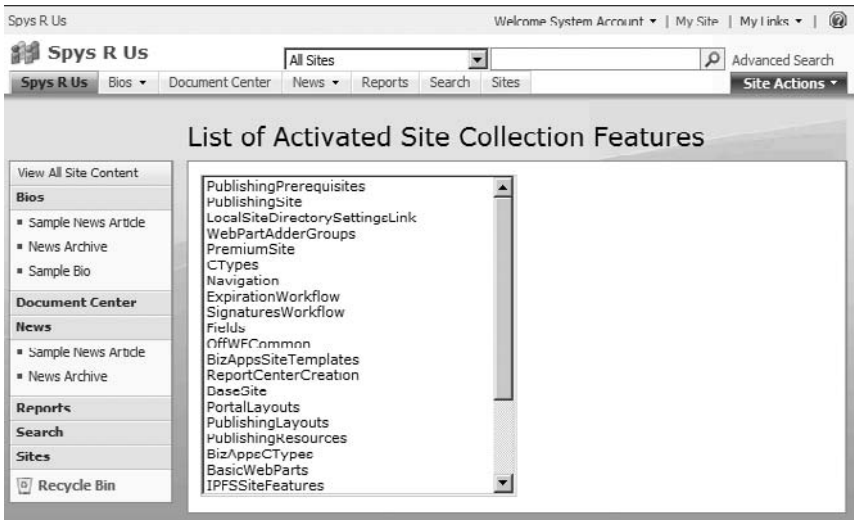


Figure 12.22 *ListFeatures.aspx output.*

If the application you are working with is a console application, you cannot leverage the `SPContext` object. However, you can access a specific site collection using an `SPSite` constructor, and then use the `OpenWeb` method of the `SPWeb` class to access a specific site. Figure 12.23 shows the code listing for a console application that displays the fields associated with a specific document library, within a specific Web site, and within a specific site collection. Figure 12.24 shows the output from the list fields console application.

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

namespace SpysRUsConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            using (SPSite siteCollection = new SPSite("http://devmoss/"))
            {
                using (SPWeb webSite = siteCollection.OpenWeb("/SiteDirectory/Sample"))
                {
                    SPList docList = webSite.Lists["Classified Documents"];
                    SPFieldCollection fieldColl = docList.Fields;
                    foreach (SPField spField in fieldColl)
                    {
                        Console.WriteLine("Field Name: " + spField.Title + " || Type: " +
spField.TypeDisplayName);
                    }
                }
            }
            Console.ReadLine();
        }
```

Figure 12.23 *List fields of classified documents document library.*

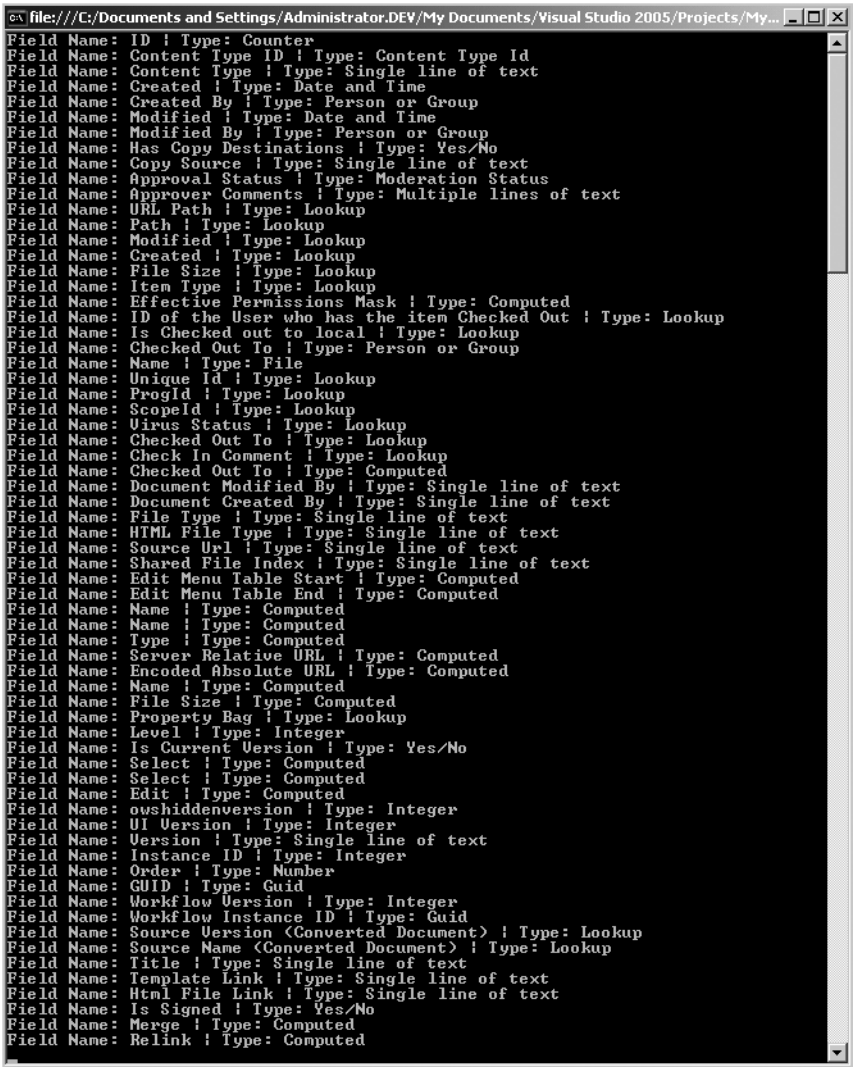


Figure 12.24 *Output from list fields console application.*

Line 12 of the code listing creates a new `SPSite` object that gives us access to the http://devmoss site collection. We then take this `SPSite` object and use the `OpenWeb` method to access a specific Web site within the site collection. The `OpenWeb` method accepts a site-relative URL. Line 14 passes the site-relative URL /SiteDirectory/Sample to the `OpenWeb` method, causing the site http://devmoss/SiteDirectory/Sample to be returned. Line 16 shows how to access a specific list within the returned Web site, and lines 17 through 23 show how to iterate through the collection of fields associated with a list and display the field title and the field type. In this example, we are leveraging the

`Using` statement to ensure that our `SPSite` and `SPWeb` objects are disposed of appropriately. We did not need to worry about disposing these objects in our previous examples, as we were using the current HTTP context, which hands the disposing responsibility over to SharePoint. It is important to note that the code that we have shown throughout this book is merely used to illustrate how to work with particular SharePoint components and objects. To improve the readability of the code listings, we have not applied all the necessary coding practices that a production application should be afforded, such as exception handling; therefore, it is imperative that when you develop your own code, you ensure that you are adhering to all the standard best practice coding techniques. Among many of the available Best Practice resources, Microsoft has an article on the MSDN Web site titled "Best Practices: Using Disposable Windows SharePoint Services Objects." It's a worthy read for any SharePoint developer.

12.2.3 Adding Content to SharePoint

So far we have looked at how you can easily retrieve information from a SharePoint site collection, or Web site. We have not yet examined how to add content to SharePoint. The next example shows how to add an item to a SharePoint Links list. Figure 12.25 shows the code listing, and Figure 12.26 shows the result of running the code. In this example, we are once again creating a console application, and therefore we must access the `SPWeb` object using the `OpenWeb` method (line 14). Once we have the desired Web site, we need to retrieve the Links list from the collection of lists in the Web site (line 16). Now that we have the list, we are ready to add new items. Lines 17 through 19 show how to add a new link item to a Links list. Pay particular attention to the format of the URL field. When adding a link, the URL field is comprised of the link URL, a comma followed by a space, and then the title of the link. The item is not added to the list until the `Update` method executes.

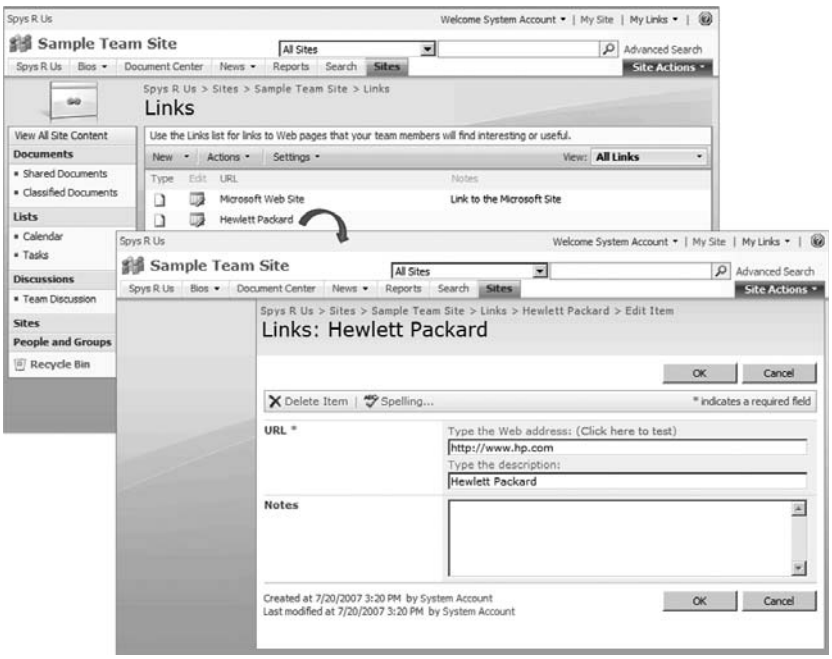
When updating data in the content database in SharePoint from a custom Web application, you need to add a `FormDigest` control to the page. The `FormDigest` control is a Web control provided by the `Microsoft.SharePoint.WebControls` class that generates a security validation preventing malicious attacks. When a user requests a page, the server returns the page with the appropriate security validation inserted. When the user subsequently submits the form, the server verifies that the security validation has not changed. Failing to insert the FormDigest control into your custom Web application results in the error shown in Figure 12.27.

To insert the `FormDigest` control, you must use the `@ Register` directive to declare a `TagPrefix` for the `Microsoft.SharePoint.WebControls` namespace in which the `FormDigest` control resides. Once you have declared the tag prefix, you can then place the `FormDigest` control onto the form of

Figure 12.25
Add Item to links list (C#).

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

namespace SpysRUsConsole
{
    class Program
    {
        static void AddLinks(string sitecollurl, string siterelurl, string
listname, string linkUrl, string linkTitle)
        {
            using (SPSite siteCollection = new SPSite(sitecollurl))
            {
                using (SPWeb webSite = siteCollection.OpenWeb(siterelurl))
                {
                    SPList lnkList = webSite.Lists[listname];
                    SPListItem newLnk = lnkList.Items.Add();
                    newLnk["URL"] = linkUrl + ", " + linkTitle;
                    newLnk.Update();
                }
            }
            Console.WriteLine("Operation successful");
            Console.ReadLine();
        }

        static void Main(string[] args)
        {
            AddLinks("http://devmoss/", "/SiteDirectory/Sample", "Links",
"http://www.hp.com", "Hewlett Packard");

        }
    }
}
```

Figure 12.26
Item successfully added to list..



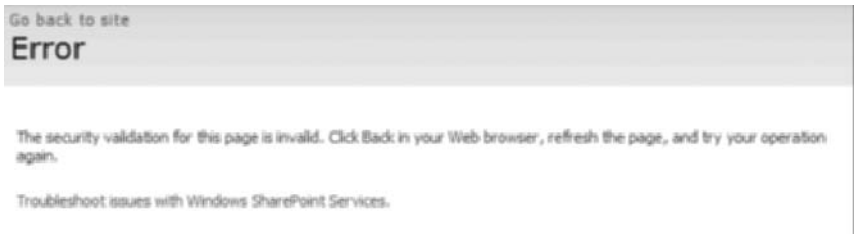


Figure 12.27 *Security validation error.*

your page. Figure 12.28 shows the code to add a link to the list, and Figure 12.29 shows the resulting output. If you are leveraging master pages, then insert the `FormDigest` control onto the master page, as opposed to the .aspx page; all the default SharePoint master pages already include this control.

Figure 12.28 *Add links code with formdigest control.*

```
<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=
71e9bce111e9429c"%><%@ Register Tagprefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"
Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c"
%>

<%@ Page Language="C#"  %>

<%@ Import Namespace="Microsoft.SharePoint" %>
<%@ Import Namespace="Microsoft.SharePoint.Utilities" %>

<script runat="server">
   protected override void OnLoad(EventArgs e)
   {
       SPWeb curWeb = SPContext.Current.Web;
       Response.Write("<H1>" + SPEncode.HtmlEncode(curWeb.Title) + "</H1><br/>");
       Response.Write("<H2>Add Links to Links List </H2><br/>");
   }

    protected void btnAddLink_Click(object sender, EventArgs e)
    {
        SPWeb curWeb = SPContext.Current.Web;
        string strListName = txtListName.Text;
        string strlinkUrl = txtLinkURL.Text;
        string strlinkTitle = txtLinkTitle.Text;
        SPList lnkList = curWeb.Lists[strListName];
        SPListItem newLnk = lnkList.Items.Add();
        newLnk["URL"] = strlinkUrl + ", " + strlinkTitle;
        newLnk.Update();
    }
</script>
<html xmlns="http://ww.w3.org/1999/xhtml">
<head runat="server">
    <title>Add Links</title>
</head>
<body>
   <form id="form1" runat="server">
    <SharePoint:FormDigest runat="server" />
    <div>
          
        <asp:Panel ID="Panel1" runat="server" Height="86px" Width="369px">
             
            <asp:Label ID="lblListName" runat="server" Text="List Name"></asp:Label>
            <asp:TextBox ID="txtListName" runat="server"></asp:TextBox> <br /> 
            <asp:Label ID="lblLinkTitle" runat="server" Text="Link Title"></asp:Label> 
            <asp:TextBox ID="txtLinkTitle" runat="server"></asp:TextBox><br /> 
            <asp:Label ID="lblLinkURL" runat="server" Text="Link URL"></asp:Label>
            <asp:TextBox ID="txtLinkURL" runat="server"></asp:TextBox><br /><br /><br />
            <asp:Button ID="btnAddLink" runat="server" OnClick="btnAddLink_Click" Text="Add Link" />
        </asp:Panel>  
    </div>
   </form>
</body>

</html>
```

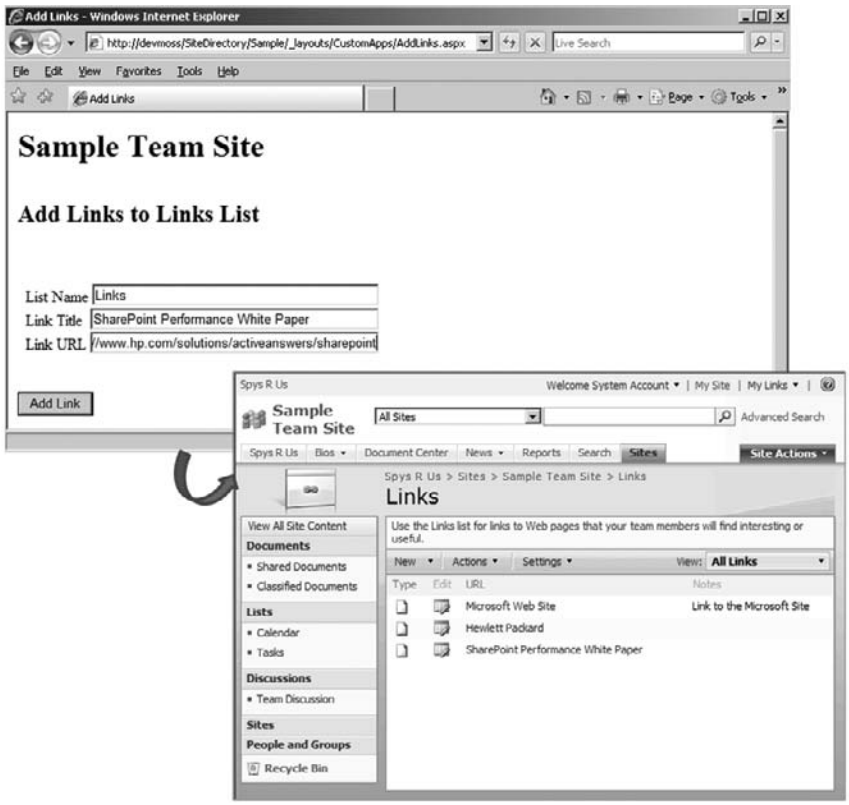


Figure 12.29
Add links form display.

12.2.4 Removing Content from SharePoint

Removing content from the SharePoint content database is similar to adding content, in that it also requires the `FormDigest` control to be present on the custom Web page from which the remove action is initiated. The actual removal of an object, such as an item, or list, or Web site, is performed by a `Delete` method in the context of the object. For example, Figure 12.30 shows the script block for a custom Web page that lists the subsites of the current site (lines 14 through 36), and then allows the user to select a subsite and delete the site (lines 37 through 52). In this instance, the actual `Delete` method occurs on the `SPWebCollection` of the current site and deletes a specific Web site from that collection using the site-relative URL. Let's assume that the user is currently accessing the Web site at the URL of http://devmoss/sitedirectory/WebSiteA, and WebSiteA contains three subsites: WebSiteB, WebSiteC, and WebSiteD. If the user selects WebSiteC to delete, then the site relative URL passed to the `Delete` method is simply the last part of the URL: WebSiteC; SharePoint appends this URL to the Web collection URL to determine which site to delete.

Figure 12.30
Code to delete selected Web site.

```
<script runat="server">
    protected override void OnLoad(EventArgs e)
    {
      SPWeb curWeb = SPContext.Current.Web;
        ListSites(curWeb);
    }
    protected void ListSites(SPWeb cWeb)
    {
        string relUrl = string.Empty;
        if (!IsPostBack)
        {
            SPWebCollection webColl = cWeb.Webs;
            lstWebs.Items.Clear();
            if (webColl.Count == 0)
            {
                lstWebs.Items.Add("No current subsites listed");
            }
            else
            {
                foreach (SPWeb subWeb in webColl)
                {
                    relUrl = subWeb.Url.Replace(cWeb.Url, "");
                    relUrl = relUrl.Replace("/", "");
                    lstWebs.Items.Add(relUrl);
                }
            }
            lstWebs.SelectedIndex = 0;
        }
    }
    protected void btnDeleteSite_Click(object sender, EventArgs e)
    {
        SPWeb curWeb = SPContext.Current.Web;
        SPWebCollection curWebColl = curWeb.Webs;
        string strSiteRelUrl = lstWebs.SelectedValue;
        lblStatus.Text = "Deleting: " + strSiteRelUrl;
        if (!(strSiteRelUrl == "No current subsites listed"))
        {
          curWebColl.Delete(strSiteRelUrl);
          lblStatus.Text = "DELETION COMPLETE";
        }
        else
        {
            lblStatus.Text = "Cannot delete Web site: No Web site selected";
        }
    }
</script>
```

If you want to delete an item from a list, then use the `Delete` method of the `SPListCollection` class, which takes the index of the item in the list as its parameter. Consequently, unless you know the index position of the item in the list, you must iterate, or search through the item collection until you find the item you would like to remove. The code in the WSS SDK shows how to delete an item from a list. This code works fine when you are only deleting one item, or when only one item matches your query. But if you employ this technique, and you want to delete more than one item, you will most likely receive an out of range error, and the last item in the range will not be deleted. This error is due to the way that deletions are handled within loops. There are several workarounds to this problem, one of which is the use of the `List<>` class of the `System.Collections.Generic` namespace. The `List<>` class enables you to create a strongly typed list that can be accessed by index. Figure 12.31 shows the code to delete all links

with a specific URL from the Links list. Instead of iterating through *all* the items in the collection, we use the `SPQuery` class to return only the link items with a URL that matches the link to be deleted (lines 23 and 25). The `SPQuery` object accepts queries in the Collaboration Application Markup Language (CAML) format. In our example, line 24 shows that we are requesting all list items where the value of the URL field is equal to the `strLinkUrl` value entered by the user. Line 25 gives us our collection of list items by executing the query. You might expect that you should immediately be able to loop through the collection and delete all the items; however, this method results in the out-of-range error previously mentioned. Instead, we create a list of items of type `SPListItem`, then iterate through the list collection and add each item to the newly created list (lines 28 through 36). With this list, it is safe for us to delete multiple items.

Figure 12.31
Code to delete links from a links list.

```
17      protected void btnDeleteLink_Click(object sender, EventArgs e)
18      {
19          SPWeb curWeb = SPContext.Current.Web;
20          string strlistName = txtListName.Text;
21          string strLinkUrl = txtLinkURL.Text;
22          SPList lnkList = curWeb.Lists[strlistName];
23          SPQuery qry = new SPQuery();
24          qry.Query = "<Where><Eq><FieldRef Name='URL' /><Value Type='URL'>" + strLinkUrl +
    "</Value></Eq></Where>";
25          SPListItemCollection itemsToDelete = lnkList.GetItems(qry);
26          if (itemsToDelete.Count > 0)
27          {
28             List<SPListItem> deleteItems = new List<SPListItem>();
29             foreach (SPListItem lstItem in itemsToDelete)
30             {
31                 deleteItems.Add(lstItem);
32             }
33             for (int i = 0; i < deleteItems.Count; i++)
34             {
35                 deleteItems[i].Delete();
36             }
37          }
38      }
```

12.2.5 Using Elevated Privileges

Although not recommended, there may be times when you need your code to perform certain functions that the current user does not have the necessary permissions to perform. For example, let's refer back to our previous code that showed how to add a link to a list; suppose that you wanted to provide this add capability to all users who use your application, regardless of their permissions on the list. How would this be achieved? The `SPSecurity` class provides a method (`RunWithElevatedPrivileges`) that allows you to run a *subset* of code in the context of an account with higher privileges than the current user. The premise is that you wrap the `RunWithElevatedPrivileges` method around your code, as shown in Figure 12.32. In certain circumstances, such as when working with Web forms, you may also need to set the `AllowSafeUpdates` method to true to temporarily turn off security validation within your

code. If you use this technique, it is imperative that you set the `AllowSafeUpdates` method back to false to avoid any potential security risks.

Figure 12.32 *Using RunWithElevated Privileges (C#).*

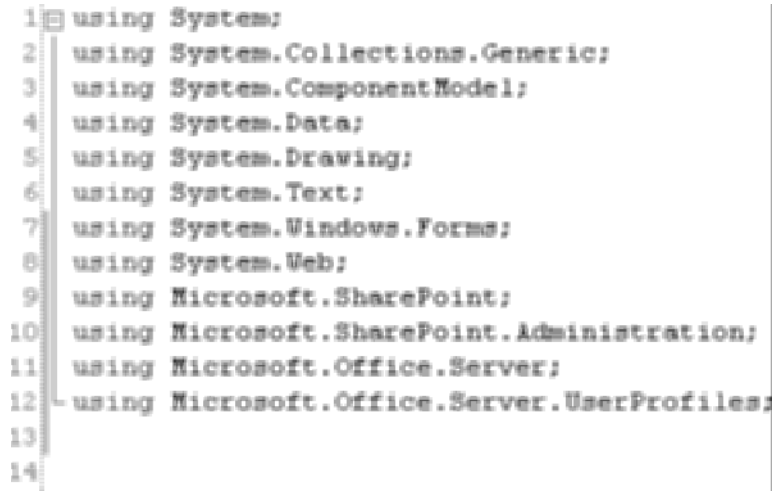
```
protected void btnAddLink_Click(object sender, EventArgs e)
{
    SPWeb curWeb = SPContext.Current.Web;
    Guid curWebId = curWeb.ID;
    Guid curSiteID = curWeb.Site.ID;
    string strlistName = txtListName.Text;
    string strlinkUrl = txtLinkURL.Text;
    string strlinkTitle = txtLinkTitle.Text;
    SPSecurity.RunWithElevatedPrivileges(delegate()
    {
        using (SPSite privSite = new SPSite(curSiteID))
        {
            privSite.AllowUnsafeUpdates = true;
            SPWeb privWeb = privSite.OpenWeb(curWebId);
            privWeb.AllowUnsafeUpdates = true;
            SPList lnkList = privWeb.Lists[strlistName];
            SPListItem newLnk = lnkList.Items.Add();
            newLnk["URL"] = strlinkUrl + ", " + strlinkTitle;
            newLnk.Update();
            privSite.AllowUnsafeUpdates = false;
            privWeb.AllowUnsafeUpdates = false;
        }
    });
}
```

12.2.6 SharePoint Server 2007 Object Model

SharePoint Server 2007 builds upon the WSS 3.0 platform; therefore, all of the WSS 3.0 object model namespaces and classes we have discussed to date are also applicable to SharePoint Server 2007. SharePoint Server 2007 extends the programming capabilities by introducing a suite of new namespaces covering a wide range of platform services, including; Excel services, records management, business auditing, workflow, user profiles, audiences, and enterprise search. As an example of the extensibility afforded by the SharePoint Server 2007 object model, we'll now take a brief look at the `UserProfiles` namespace. Additional information on the other namespaces can be found in the Microsoft Office SharePoint Server 2007 SDK.

The `UserProfiles` namespace provides an extensive list of classes that you can leverage to work with user profiles. For example, the `UserProfile` class represents the user profile for a person in the user profile database; using this class, you can access information such as the SharePoint groups to which the user belongs. A common request from an extensibility perspective is to be able to programmatically update information stored in a user's profile, such as their telephone details. Figures 12.33, and 12.34 show the code listing for a sample Windows application that provides a very rudimentary

interface for setting some of the properties in a user's profile. The form is comprised of five text boxes: `txtSite`, `txtUsername`, `txtHomePhone`, `txtCellPhone, and txtAboutMe`. Also included on the form are two buttons: one retrieves information from the user's profile, and another updates the profile of the user whose username is entered in the `txtUsername` text box; the profile magic happens in the button's _click event code.



```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Text;
7  using System.Windows.Forms;
8  using System.Web;
9  using Microsoft.SharePoint;
10 using Microsoft.SharePoint.Administration;
11 using Microsoft.Office.Server;
12 using Microsoft.Office.Server.UserProfiles;
13
14
```

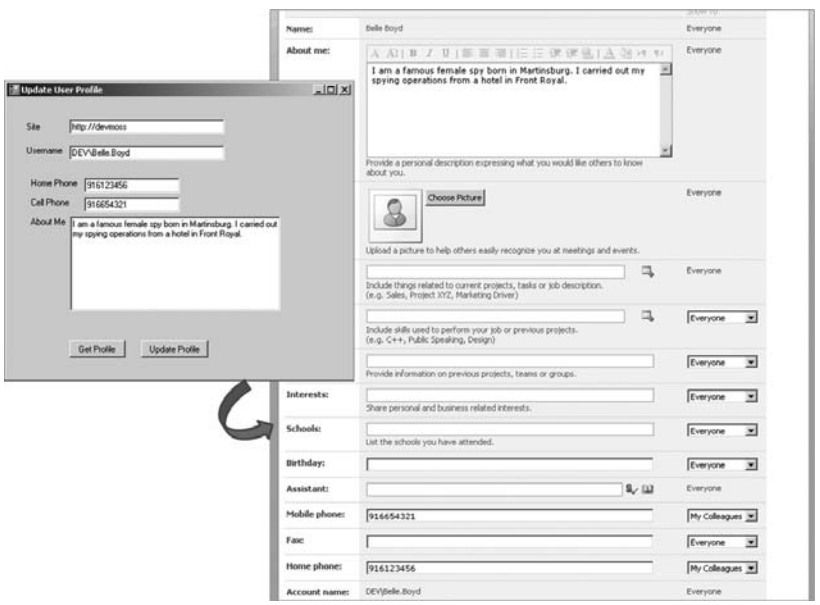
Figure 12.33
Using directives required for user profile application (C#).

When working with user profiles, you must first create a `UserProfileManager` object before you can access a user profile. The `UserProfileManager` class contains a collection of user profile objects used to access user profile information. Once you have created the `UserProfileManager` object, you can then use this object to retrieve individual user profiles. In our code example, we use the `GetUserProfile` method of the `UserProfileManager` class to retrieve the user profile for the user entered in the `txtUsername` text box (lines 44 through 47). The `PropertyConstants` class contains the fields associated with the user profile; for example, `PropertyConstants.AboutMe` specifies the name of the property that contains the commentary about the user. Lines 48 through 50 use the `PropertyConstants` class to retrieve the values of the `HomePhone`, `CellPhone`, and `AboutMe` fields of the user profile. Finally, the `Commit` method of the `UserProfile` class commits the changes to the user profile database (line 51). Lines 24 through 37 use the same technique to retrieve the properties for a specific user. Figure 12.35 shows the running Windows application and the corresponding SharePoint Edit User Profile page, verifying that the update was successful.

Figure 12.34
Update user profile Windows application code (C#).

```
namespace UpdateUserProfile
{
    public partial class frmUpdateUserProfile : Form
    {
        public frmUpdateUserProfile()
        {
            InitializeComponent();
        }

        private void btnGetProfile_Click(object sender, EventArgs e)
        {
            string spSite = txtSite.Text;
            using (SPSite site = new SPSite(spSite))
            {
                ServerContext context = ServerContext.GetContext(site);
                UserProfileManager profileManager = new UserProfileManager(context);
                string strUserName = txtUserName.Text;
                UserProfile uP = profileManager.GetUserProfile(strUserName);
                txtHomePhone.Text = uP[PropertyConstants.HomePhone].Value.ToString();
                txtCellPhone.Text = uP[PropertyConstants.CellPhone].Value.ToString();
                txtAboutMe.Text = uP[PropertyConstants.AboutMe].Value.ToString();
            }
        }

        private void btnUpdateProfile_Click(object sender, EventArgs e)
        {
            string spSite = txtSite.Text;
            using (SPSite site = new SPSite(spSite))
            {
                ServerContext context = ServerContext.GetContext(site);
                UserProfileManager profileManager = new UserProfileManager(context);
                string strUserName = txtUserName.Text;
                UserProfile uP = profileManager.GetUserProfile(strUserName);
                uP[PropertyConstants.HomePhone].Value = txtHomePhone.Text;
                uP[PropertyConstants.CellPhone].Value = txtCellPhone.Text;
                uP[PropertyConstants.AboutMe].Value = txtAboutMe.Text;
                uP.Commit();
            }
        }
    }
}
```

Figure 12.35
Update user profile Windows application.



12.3 Web Services

The Windows SharePoint Services and SharePoint Server 2007 object models are specifically server-side object models, meaning that the code that leverages them must run on a server running Windows SharePoint Services 3.0 and SharePoint Server 2007 respectively; the code cannot run remotely on a client. If you need to access SharePoint objects remotely through code you can use Windows SharePoint Services Web services.

There are two levels of Web services available with SharePoint Server 2007: the Web services that ship with WSS 3.0 and those that ship with SharePoint Server 2007. Once again, since SharePoint Server 2007 is built upon the WSS 3.0 platform, all WSS 3.0 Web services are available to use with SharePoint Server 2007. Table 12.2 lists the WSS 3.0 specific Web services, and Table 12.3 lists the Web services available for working with SharePoint Server 2007. The Simple Object Access Protocol (SOAP) interfaces leveraged by these services provide developers with an object model that can access data remotely. In fact, the SharePoint Web services infrastructure is built upon the WSS 3.0 object model, and, as such, provides much of the same basic functionality, such as working with lists, Webs, views, and users. In SharePoint Portal Server 2003 and WSS 2.0, Microsoft improved the performance of the provided SOAP interfaces by optimizing the Web services to reduce the number of roundtrips required to the server. For example, the `GetList` method of the Lists Web service returns the schema of the specified list and assigns it to an `XMLNode` object; in order to retrieve additional information from the list, you simply query the `XmlNode`. The same efficiency has been carried forward into the WSS 3.0 and SharePoint Server 2007 Web services architecture.

Table 12.2 *WSS 3.0 Web Services*

Web Service	Web Reference	Description
Administration	http://<AdminSite>/_vti_adm/Admin.asmx	Provides methods for working with site collections, such as `CreateSite`, `DeleteSite`, `GetLanguages`, and `RefreshConfigCache`.
Authentication	http://<Site>/_vti_bin/Authentication.asmx	Provides methods for authenticating users, such as `Login` and `Mode`.
Alerts	http://<Site>/_vti_bin/Alerts.asmx	Provides methods for working with alerts, such as `GetAlerts` and `DeleteAlerts`.

Table 12.2 *WSS 3.0 Web Services (continued)*

Web Service	Web Reference	Description
Copy	http://<Site>/_vti_bin/Copy.asmx	Provides methods of copying items between locations in SharePoint, such as `CopyIntoItems`, `CopyIntoItemsLocal`, and `GetItem`.
Document Workspace	http://<Site>/_vti_bin/Dws.asmx	Provides methods for working with a document workspace, such as `CreateFolder`, `DeleteFolder`, and `GetDWSMetaData`.
Forms	http://<Site>/_vti_bin/Forms.asmx	Provides methods for returning forms used when working with list contents, such as `GetForm` and `GetFormCollection`.
Imaging	http://<Site>/_vti_bin/Imaging.asmx	Provides methods for working with Picture Libraries, such as `ListPictureLibrary`, `GetListItems` and `Upload`.
List Data Retrieval	http://<Site>/_vti_bin/DspSts.asmx	Provides a `Query` method for performing queries against WSS 3.0.
Lists	http://<Site>/_vti_bin/Lists.asmx	Provides methods for working with lists, such as `AddList`, `DeleteList`, `ApplyContentTypeToList`, `GetList` and `GetListCollection`.
Meetings	http://<Site>/_vti_bin/Meetings.asmx	Provides methods for working with sites created with the Meeting site template, such as `AddMeeting`, `RemoveMeeting` and `GetMeetingsInformation`.
People	http://<Site>/_vti_bin/People.asmx	Provides methods for working with security groups, such as `SearchPrincipals`, and `ResolvePrincipals`.
Permissions	http://<Site>/_vti_bin/Permissions.asmx	Provides methods for working with permissions, such as `GetPermissionCollection`, `AddPermission`, `RemovePermission`, and `UpdatePermission`.
Site Data	http://<Site>/_vti_bin/SiteData.asmx	Provides methods that retrieve information about the sites in a server farm, such as `GetSite`, `GetWeb`, `GetList`, and `GetListCollection`.
Sites	http://<Site>/_vti_bin/Sites.asmx	Provides methods for working with the site templates on a server, such as `GetSiteTemplates` and `ImportWeb`

Table 12.2 *WSS 3.0 Web Services (continued)*

Web Service	Web Reference	Description
Search	http://<Site>/_vti_bin/spsearch.asmx	Provides methods for performing searches remotely. Methods include `Query`, `QueryEx`, `Registration`, and `Status`.
Users and Groups	http://<Site>/_vti_bin/usergroup.asmx	Provides methods for working with users and groups, such as `GetGroupCollectionFromUser`, `RemoveUserFromGroup`, `RemoveRole`, and `RemoveUserCollectionFromSite`.
Versions	http://<Site>/_vti_bin/Versions.asmx	Provides methods for working with file versions, such as `RestoreVersion`, `GetVersions` and `DeleteVersion`.
Views	http://<Site>/_vti_bin/Views.asmx	Provides methods for working with List Views, such as `AddView`, `DeleteView`, `GetView`, `UpdateView`, and `GetViewCollection`.
Web Part Pages	http://<Site>/_vti_bin/WebPartPages.asmx	Provides methods for working with Web Parts, such as `AddWebPart`, `DeleteWebPart`, `GetWebPart` and `GetWebPartProperties`.
Webs	http://<Site>/_vti_bin/Webs.asmx	Provides methods for working with sites and subsites, such as `CreateContentType`, `GetActivatedFeatures`, `GetWeb` and `GetWebCollection`.

Table 12.3 *SharePoint Server 2007 Web Services*

Web Service	Web Reference: (http://<Site>/_vti_bin/...)	Description
Official File	OfficialFile.asmx	Provides methods for working with records management documents, such as `SubmitFile`, `GetRecordRouting`, and `GetRecordRoutingCollection`.
Published Links	PublishedLinksService.asmx	Provides a `GetLinks` method for working with published links.

Table 12.3 *SharePoint Server 2007 Web Services (continued)*

Web Service	Web Reference: (http://<Site>/_vti_bin/...)	Description
Search	Search.asmx	Provides methods for working with enterprise search, such as `GetPortalSearchInfo`, `GetSearchMetaData`, `Query`, and `QueryEx`.
User Profile Change	UserProfileChangeService.asmx	Provides methods for working with changes made to user profiles, such as `GetUserAllChanges`, `GetUserChanges`, and `GetUserCurrentChangeToken`.
User Profile	UserProfileService.asmx	Provides methods for working with user profiles, such as `AddMembership`, `CreateMemberGroup`, `AddLink`, `GetUserLinks`, `AddColleague`, and `GetUserMemberships`.
Workflow	Workflow.asmx	Provides methods for working with workflows, such as `StartWorkflow`, `GetWorkflowTaskData`, and `GetWorkflowDataForItem`.

Visual Studio can be used to view the list of available methods for the Web services provided by WSS 3.0 and SharePoint Server 2007. Simply create a project in Visual Studio 2005 and add a Web reference to the specific Web service in which you are interested. The dialog displays the various methods exposed by the services (Figure 12.36), and clicking on a specific method displays information regarding the actual SOAP request.

12.3.1 Working with Web Services

Regardless of the actual Web service you are working with, there are a few common steps that need to be performed. The first step is adding a reference to the Web service that you intend to use. Adding a reference to your Visual Studio project is a simple matter of entering the Web service URL and clicking the Add Reference button on the Add Reference dialog (Figure 12.36). The next step involves instantiating the Web service and then establishing the credentials to use. Once those initial steps have been performed, you are free to work with the various methods provided by the Web service.

To illustrate the process, let's create a simple Windows application that creates a new custom list in a SharePoint site. The form will have a single text box control, which the user enters the name of the list to create, and a button control which triggers the list creation. In this application, the URL that we use to reference the Web service results in defining the location where the list is created. For example, if we add a Web reference for http://devmoss/

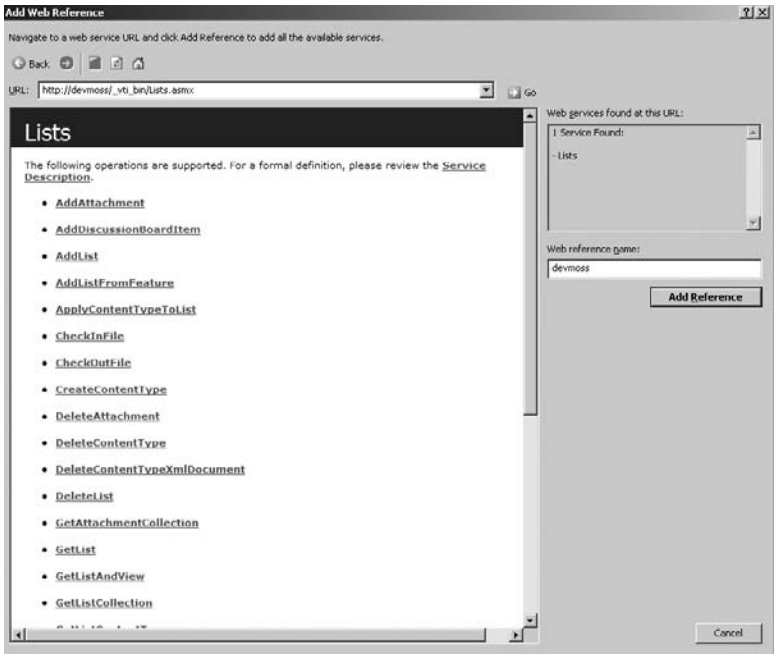


Figure 12.36
Adding a reference to the Lists Web service.

_vti_bin/Lists.asmx, then the list is created in the root site; on the other hand, if we add a Web reference for http://devmoss/sitedirectory/Sample/_vti_bin/Lists.asmx, the list is created in the Sample subsite. For the development of production Web service applications, the Web service URL can be defined at runtime by setting the `<serviceinstancename>.Url` property to the appropriate URL. For example:

```
lstSvc.Url="http://devmoss/sitedirectory/Sample/_vti_bin/
Lists.asmx"
```

1. Using Visual Studio 2005, create a new C# Windows Application project.
2. Add a Web reference to the Lists Web service.
3. Add a Text Box control and a Button control to the form.
4. Insert the code listed in Figure 12.37 behind the form.
5. Compile and run the application.

The code listed in Figure 12.37 shows how to leverage the `Lists` Web service to create a new list; as you can see, there's not much to it. Line 26

Figure 12.37
Web Services code to add a list.

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Text;
7  using System.Windows.Forms;
8  using System.Xml;
9
10 namespace WebServiceApp
11 {
12     public partial class Form1 : Form
13     {
14         public Form1()
15         {
16             InitializeComponent();
17         }
18
19         private void label1_Click(object sender, EventArgs e)
20         {
21
22         }
23
24         private void button1_Click(object sender, EventArgs e)
25         {
26             devmoss.Lists listSvc = new devmoss.Lists();
27             listSvc.Credentials = System.Net.CredentialCache.DefaultCredentials;
28             XmlNode ndLst = listSvc.AddList(textBox1.Text, "Created by Web Service",
   100);
29             MessageBox.Show(ndLst.OuterXml);
30         }
31     }
```

instantiates the `Lists` Web service; `devmoss` represents the Web reference folder within the Visual Studio project, and `Lists` represents the Web service to which we are connecting. Line 27 authenticates the user running the code to the Web service by passing the user's credentials from the system credential cache. Line 28 creates the new list using the `AddList` method from the Lists Web service; the text entered by the user in the Text Box control determines the name of the new list, and the "`100`" parameter dictates the list type, which in this case equates to a custom list. Line 29 is merely to show you the XML that is returned by the service (Figure 12.38). This data can be quite useful when you are deciphering the structure of existing lists, since it provides you read access to the associated list schema. Figure 12.39 shows the outcome of our sample code.

When working with the Lists Web service, knowing the various attributes and their values is essential. For example, if you want to update items in a list, you need to know the internal name, or `GUID`, of the list, such as `{5CE7D36-DADE-4FCD-BE00-A1976032C7BE}`. Fortunately, this information is readily available from the XML node returned from the `GetList` and `GetListCollection` methods of the Lists Web service.

Our next example demonstrates an application that retrieves the lists associated with a specific site and displays their names in a List Box control representing the available lists. A second list box displays the attributes, along with their corresponding values, of the currently selected item in the

Figure 12.38
OuterXML display.

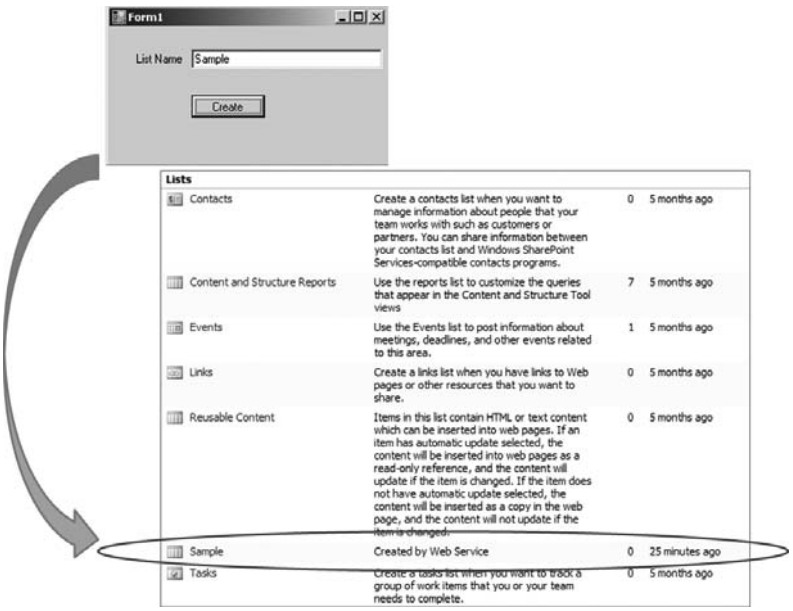


Figure 12.39
Result of lists Web service application.

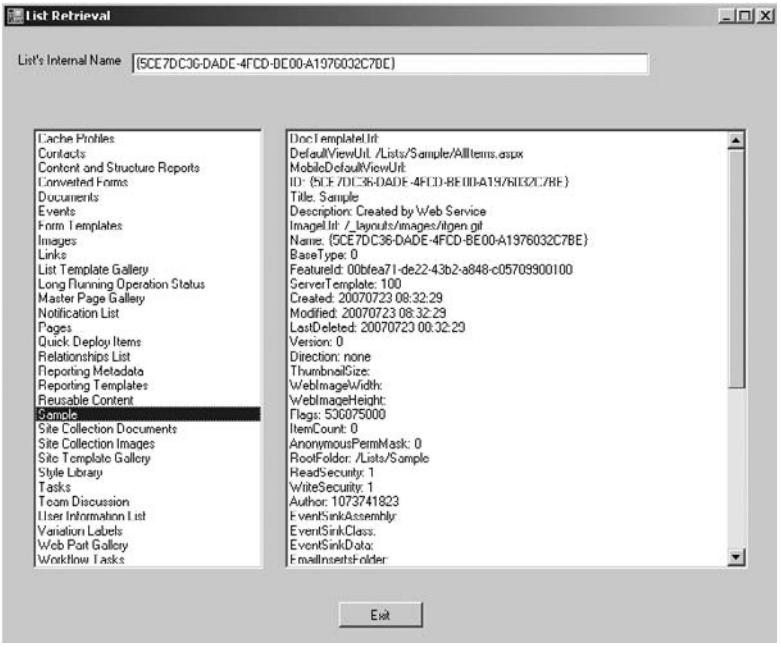


Figure 12.40
List retrieval sample application.

list of available lists. Figure 12.40 shows the running application and Figure 12.41 shows the accompanying code listing. Notice that the internal name of the list also appears in a text box at the top of the form; as a developer, you can use this simple application to retrieve the internal names of the various lists with which you want to work by copying the name directly from the text box.

There are two main methods that provide the functionality for our application: `Form1_Load` and `lstLists_SelectedIndexChanged`. `Form1_Load` is called when our application initially loads the Windows form; consequently, this method contains the code necessary to populate our list boxes with the appropriate data. Line 25 authenticates the user running the application, and line 26 calls the `GetListCollection` method from the Lists Web service to retrieve the collection of lists associated with the referenced site. Lines 27 through 30 traverse the collection and extract the title of each list; each title is subsequently added to the list box: `lstLists`. Line 31 extracts the internal name of the first list and populates the text box with this value.

When the user selects another item in the `lstLists` list box, `lstLists_SelectedIndexChanged` is called. Since we have already established our credentials during the form load, we can immediately start working with the `Lists` Web service. This time we leverage the `GetList` method of the Lists Web service by passing the name of the currently selected item in

the `lstLists` list box (line 41). The `GetList` method returns the schema for the specified list in an XML node. Lines 43 through 48 traverse the schema and extract the name and value of each attribute associated with the list. The name and its corresponding value are then added to the second list box (`lstAttributes`). Finally, line 49 extracts the internal name of the selected list and populates the text box with this value.

Figure 12.41 *List retrieval application code (C#).*

```
10  namespace ListRetrievalApp
11  {
12      public partial class Form1 : Form
13      {
14          private devmoss.Lists listSvc = new ListRetrievalApp.devmoss.Lists();
15          private XmlNode ndLstCol;
16          private XmlNode ndLst;
17
18          public Form1()
19          {
20              InitializeComponent();
21          }
22
23          private void Form1_Load(object sender, EventArgs e)
24          {
25              listSvc.Credentials = System.Net.CredentialCache.DefaultCredentials;
26              ndLstCol = listSvc.GetListCollection();
27              for (int nNode = 0; nNode <= ndLstCol.ChildNodes.Count - 1; nNode++)
28              {
29                  lstLists.Items.Add(ndLstCol.ChildNodes.Item(nNode).Attributes.
GetNamedItem("Title").InnerText);
30              }
31              txtInternalName.Text = ndLstCol.ChildNodes.Item(0).Attributes.
GetNamedItem("Name").InnerText;
32          }
33
34          private void btnExit_Click(object sender, EventArgs e)
35          {
36              Form1.ActiveForm.Dispose();
37          }
38
39          private void lstLists_SelectedIndexChanged(object sender, EventArgs e)
40          {
41              ndLst = listSvc.GetList(lstLists.Text);
42              lstAttributes.Items.Clear();
43              for (int nNode = 0; nNode <= ndLst.Attributes.Count - 1; nNode++)
44              {
45                  string lstItem = ndLst.Attributes.Item(nNode).Name.ToString();
46                  lstItem += ": " + ndLst.Attributes.Item(nNode).Value.ToString();
47                  lstAttributes.Items.Add(lstItem);
48              }
49              txtInternalName.Text = ndLstCol.ChildNodes.Item(lstLists.SelectedIndex).
Attributes.GetNamedItem("Name").InnerText;
50          }
51      }
52  }
```

Throughout this chapter we have discussed the basic concepts involved in programmatically manipulating content in the SharePoint database. We provided several examples of accessing SharePoint content using both the object model, and using Web services. The chapters that follow build upon this information by providing more examples of working with SharePoint data, and introducing new methods of extending the functionality of your SharePoint implementation.

This page intentionally left blank

13

Building and Deploying Web Parts

13.1 Introduction

Throughout this book we have touched on the concept of Web Parts and Web Part pages; by now you should understand that Web Parts remain a cornerstone of the SharePoint architecture, in that they are responsible for presenting much of the functionality to the user. For example, the announcements that appear on the home page of a Team site are presented to the user though a Web Part. This chapter examines the concept of Web Parts and Web Part pages, and where they fit into the overall SharePoint architecture. SharePoint Server 2007 and Windows SharePoint Services come equipped with a host of Web Parts that you can tailor to suit your business needs. In this chapter we highlight two such Web Parts, the data view Web Part and the content query Web Part, and show how they can be leveraged in your SharePoint deployment. Finally, we show you how to increase the functionality of your site by building your own custom Web Parts.

13.2 What is a Web Part?

The Web Part has come a long way since the Digital Dashboard days. The Digital Dashboard technology was used by SharePoint Portal Server 2001, and was superseded by the Web Part page in SharePoint Portal Server 2003 and Windows SharePoint Services 2.0. The Web Part page still exists today and is basically an ASP.NET page that presents the user with information derived from a variety of sources. The information is served through the use of Web Parts. A Web Part is basically a Web control that inherits from the `System.Web.UI.WebControls.WebParts` namespace and renders content to the user. A Web Part can be placed in the desired region of a Web page. Web Parts became very popular among the developer community because they were easy to develop and, if coded correctly, provided much in the way of flexibility and portability. Their successful adoption rate ultimately led to the inclusion of the Web Part framework architecture in ASP.NET 2.0, which

means that you no longer require SharePoint to leverage Web Parts in your Web applications. As discussed numerous times throughout this book, SharePoint Server 2007 and Windows SharePoint Services 3.0 are built upon ASP.NET 2.0, and consequently inherit and extend many of the ASP.NET capabilities. The ASP.NET 2.0 Web Part framework architecture is an example of one area that SharePoint has inherited and extended by replacing components of the Web Part framework with SharePoint-specific components. Figure 13.1 shows the composition of an ASP.NET 2.0 page that uses Web Parts and a WSS 3.0 Web Part Page. Any ASP.NET 2.0 page that wants to leverage the use of Web Parts must contain one, and only one, `WebPartManager` control, and one or more `WebPartZone` controls. The ASP.NET 2.0 `WebPartManager` control is responsible for keeping track of the Web Parts that have been added to the page and the zone in which they were added, and also manages the customization and personalization data for the Web Parts. A `WebPartZone` control's sole function is to provide a home for Web Parts. Pages in SharePoint are referred to as Web Part Pages and house an `SPWebPartManager` control and one or more SharePoint specific `WebPartZone` controls. The `SPWebPartManager` inherits from the ASP.NET `WebPartManager`, and the SharePoint specific `WebPartZone` inherits from the ASP.NET `WebPartZone`. Both controls are used to ensure that the data associated with each Web Part is managed and maintained in the appropriate SharePoint content database.

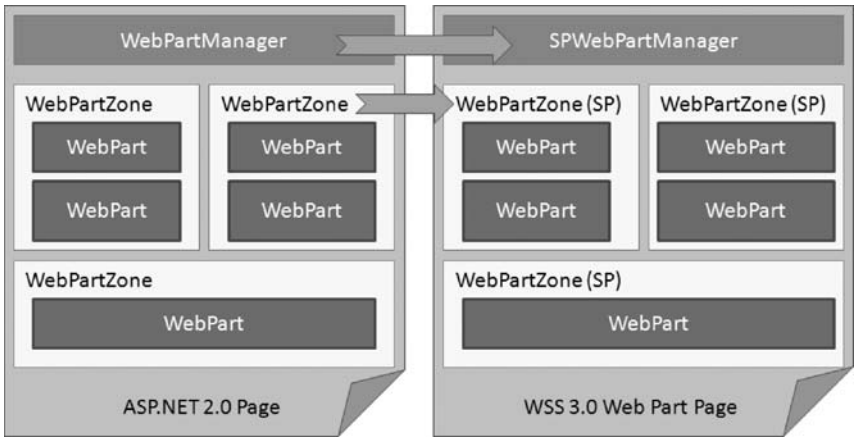


Figure 13.1 *ASP.NET 2.0 Web parts.*

A Web Part page consists of one or more Web Part zones, which in turn contain one or more Web Parts. Web Parts that live in a zone are referred to as dynamic Web Parts. Dynamic Web Parts and their associated properties, such as the title of the Web Part, are stored in the SharePoint database and can be customized and personalized by users with the appropriate permissions.

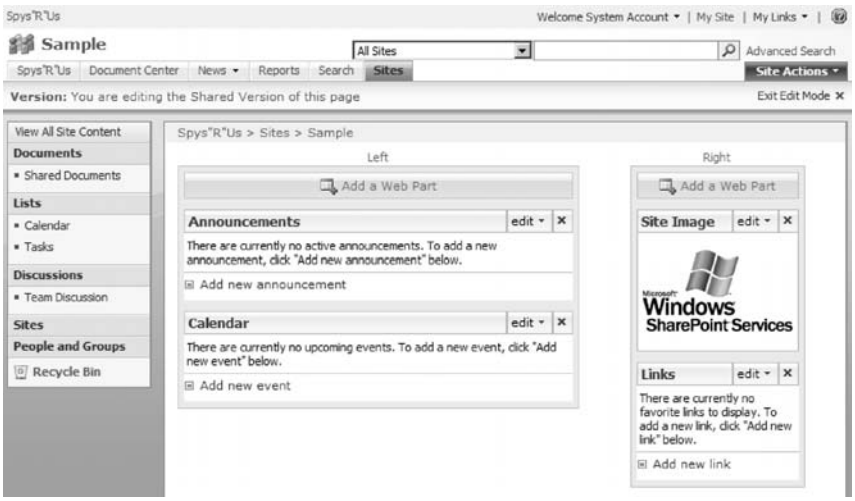


Figure 13.2
WSS 3.0 team site Web Part zones.

Figure 13.2 shows the zone layout on the default Web Part page of a SharePoint team site. Note there are only two zones, named Left and Right, on this Web Part page, and each zone contains two Web Parts. The zone layout and naming can differ between Web Part pages. For example, the home page of a standard collaboration portal contains five zones named: right zone, top zone, middle left zone, middle right zone, and bottom zone (Figure 13.3). The Web Part zone layout is viewable in edit mode, which is accessible from the Edit Page command on the Site Actions menu. In addition, you can add and remove Web Part zones from a Web Part page by customizing the site templates, or using SharePoint Designer 2007, using custom code.

Web Parts that do not live within a Web Part zone are referred to as static Web Parts. Static Web Parts are added to the Web Part page by developers at design time. In addition, static Web Parts and the properties associated with the Web Parts are stored directly in the Web Part page and not in the SharePoint database.

13.2.1 Creating a Web Part Page

SharePoint sites come equipped with many Web Part pages hosting multiple Web Parts that provide the functionality of the site. You can easily extend and customize this out-of-the-box functionality by adding, removing, and rearranging Web Parts. In addition to this, you also have the ability to extend your site by creating your own Web Part pages. The steps to create a new Web Part page are as straightforward as creating a SharePoint list or document library:

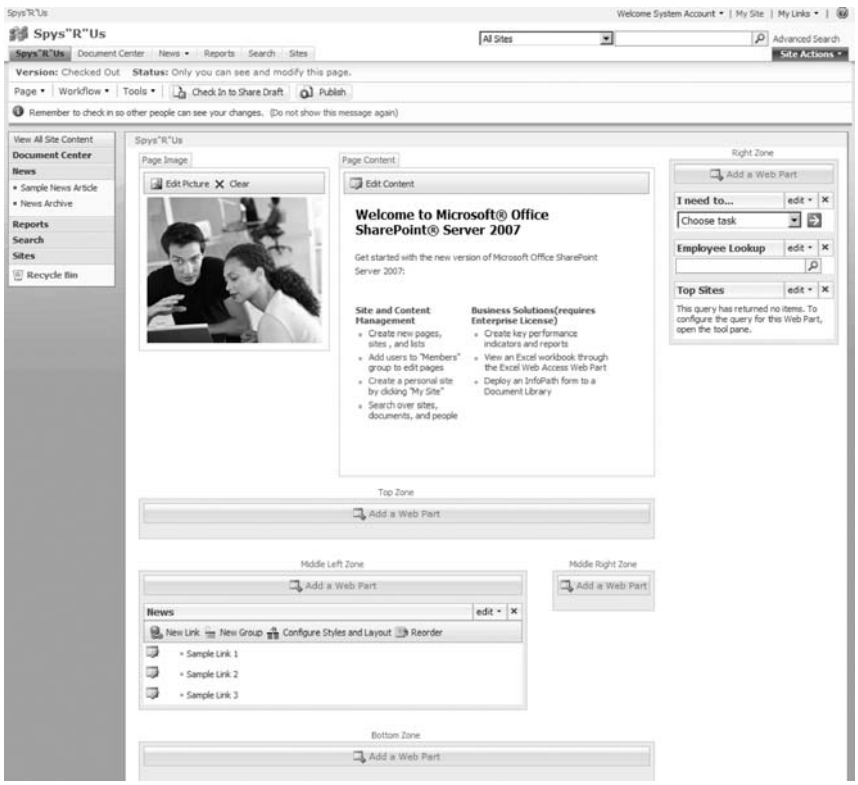


Figure 13.3 *Collaboration Portal Home Page Web Part Zones*

1. Navigate to a WSS 3.0 team site in which you wish to add a new Web Part page.
2. From the Site Actions menu, click the Create link. The Create page appears.
3. From the Web Pages column on the right of the page, click Web Part page.
4. Type the file name of the Web Part page—for example, mypage.aspx.
5. Select the layout template that you wish to apply to the page. For example, the header, left column, body layout template creates a Web Part page with three Web Part zones: header, left column, and body.
6. Select the document library in which you wish to save the Web Part page—for example, Shared Documents.
7. Click the Create button.

Once you click the Create button, SharePoint creates a new Web Part page in the specified location and automatically opens the page in edit mode so that you can start adding Web Parts to the page (Figure 13.4). Notice that the new Web Part page already has a lot of functionality, such as a navigational structure, a title bar, and all the appropriate Web Part zones. This allows you to focus on the real functionality of the page.

To create a Web Part page from a SharePoint Server 2007 site, such as a collaboration portal, simply replace steps 1 and 2 with the following:

1. Navigate to the SharePoint Server 2007 site in which you wish to add a new Web Part page.
2. From the vertical navigation bar, click View All Site Content. The All Site Content page appears. Click the Create link. The Create page appears.

Figure 13.4
Creating a Web Part page.

13.2.2 Adding, Modifying, and Arranging Web Parts

Adding, modifying, and arranging Web Parts on a Web Part page is also very straightforward. WSS 3.0 comes equipped with a host of useful Web Parts that provide additional functionality to your site. For example, the content

editor Web Part is great for adding rich text or HTML code to your page. Clicking the Add a Web Part banner associated with a specific zone when you are in edit mode displays the Add Web Parts Web page dialog (Figure 13.5). From this dialog you can browse through all the available Web Parts, check those that you want to add to the zone, and add them to your page with the single click of the Add button. The Add Web Parts Web page dialog lists the available Web Parts by category. For example, all the list and library Web Parts, such as the announcements Web Part, are grouped together under the heading Lists and Libraries. You can browse all the Web Parts in a single Web page dialog by scrolling through the various categories.

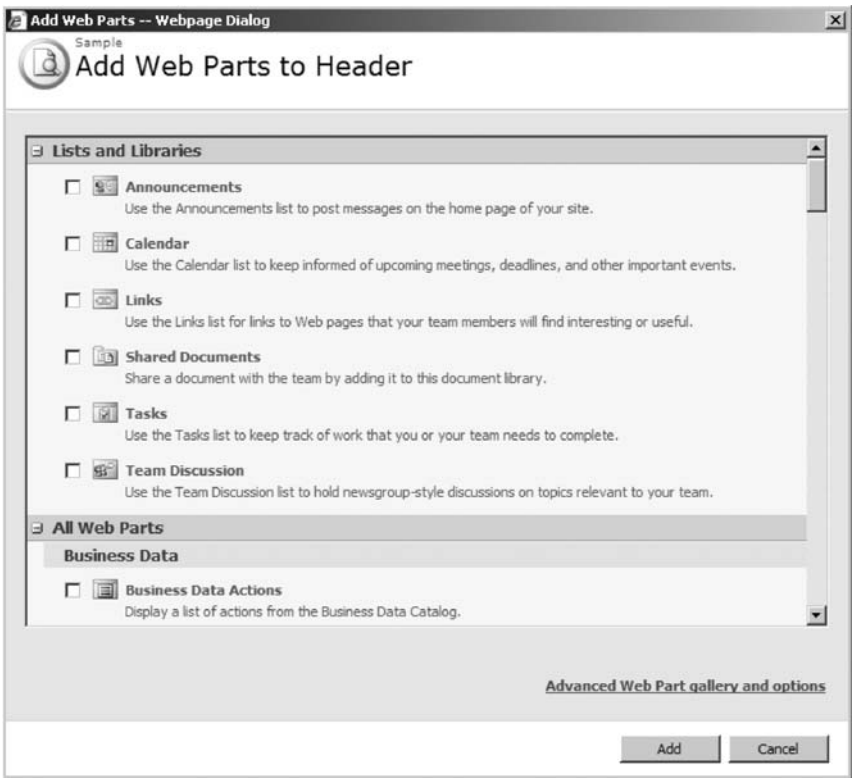


Figure 13.5
Add Web Parts Web page dialog.

Alternatively, if you would prefer to use the Add Web Parts task pane method that was available with the previous version of SharePoint, you can click the advanced Web Part gallery and options link (Figure 13.6). The Add Web Parts task pane lists the available Web Parts in alphabetical order, displaying 10 Web Parts at a time; there are links to navigate to the previous 10 and next 10 Web Parts. The task pane provides a filter option, allowing you to filter the Web Parts displayed by category. The advantage of using the task

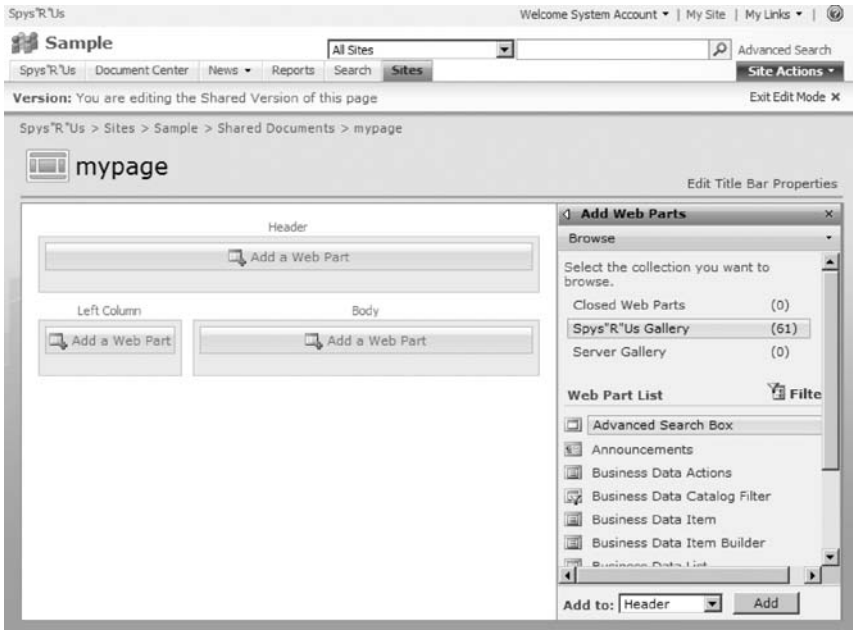


Figure 13.6
Add Web Parts task pane.

pane is that you can drag and drop Web Parts from the Add Web Parts task pane onto *any* of the zones on your Web Part page.

In addition to the browse capability, the Add Web Parts task pane provides two more options for adding Web Parts: Search and Import. The search option provides the ability to search all of the available Web Part galleries for a Web Part containing specific text; both the Web Part title and the Web Part description are included in the search. For example, if you search the galleries for the text "Table," the table of contents Web Part and the content editor Web Part are returned. The content editor Web Part description contains the word table. The search results appear in the task pane, sorted by gallery (Figure 13.7). Adding a Web Part to the page from this task pane is a simple matter of dragging and dropping the Web Part from the Search Results section to the desired zone on the Web Part Page.

The import option provides the ability to upload a Web Part definition file (.Webpart) to the Web Part page gallery and add it to the current Web Part page. Once the file is uploaded, the Web Part that it represents is accessible from the Browse task pane. Note that although the file extension for a Web Part definition file has changed from the previous version of SharePoint, from .dwp to .Webpart, you can still import .dwp files into SharePoint.

Once you have the desired Web Parts added to your page, you can begin to tailor them to suit your needs. For example, let's say that you added a content editor Web Part so that it would display some welcome informa-

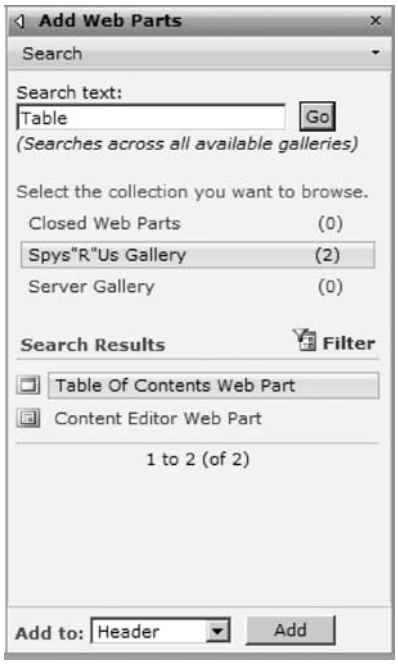


Figure 13.7 *Searching for a Web Part.*

Figure 13.8 *Content editor Web part.*

tion from the owner of the site, along with their photograph (Figure 13.8). We can modify the title of the Web Part in addition to adding content to the rich text field associated with the Web Part, by accessing the Web Part Editor task pane. Each Web Part has a set of base properties, such as height,

width, and any number of custom properties. Depending on how the Web Part was developed, these properties may appear in the Web Part Editor task pane for customization.

Removing a Web Part from a Web Part page is a simple process that involves accessing the Web Part menu and clicking Close. The Web Part is still available in the Closed Web Parts gallery and can be added again at any time. Web Parts can also be removed from a Web Part page by clicking Delete from the Web Part menu when in edit mode.

13.3 Microsoft Office SharePoint Server 2007 Web Parts

Both Windows SharePoint Services 3.0 and SharePoint Server 2007 come equipped with many useful Web Parts out of the box. Roll-up Web Parts that gathered information from multiple document libraries and lists, across multiple sites, were a common functionality request that was lacking in SharePoint Portal Server 2003. To attain this functionality, you either had to purchase a third-party tool or build your own home-grown Web Part. With Windows SharePoint Services 3.0 and SharePoint Server 2007, Microsoft offers a number of out-of-the-box Web Parts that provide roll-up capabilities enabling you to aggregate data across multiple site collections, sites, document libraries, and lists. In this next section we'll examine two such Web Parts, the data view Web Part that comes with SharePoint Designer 2007 and the content query Web Part that comes with SharePoint Server 2007, and show how you can use these tools to display information from a variety of sources.

13.3.1 SharePoint Designer Data View Web Part

Microsoft Office SharePoint Designer 2007 is the new version of FrontPage specifically targeted at customizing and extending Web sites built upon Windows SharePoint Services V3.0. SharePoint Designer 2007 provides a number of tools that make it easy to access data from a number of different sources, including SharePoint lists, document libraries, and XML files. The data view Web Part is a very powerful tool that was first introduced with FrontPage 2003. The FrontPage 2003 version allows data to be extracted from a variety of sources using the appropriate SharePoint data retrieval service adapter. The adapter returns data in the form of an XML document which the data view Web Part formats using extensible stylesheet transformations (XSLT) for final presentation in the browser. The architecture for retrieving and presenting data for viewing carries forward with the SharePoint Designer data view Web Part; however, additional capabilities include the ability to write data to a single data source and the ability to retrieve data

from multiple sources and combine them into a single XML document to be consumed by the data view Web Part. The latter enhancement is referred to as Linked Data Sources, and it allows a Web designer to aggregate views across multiple data sources and present the data to the user in a single view.

13.3.2 Data Source Library

The Data Source Catalog available with FrontPage 2003 has been renamed the Data Source Library in SharePoint Designer. SharePoint Designer provides a Data Source Library task pane from which you can access and manage data sources, whether they exist on your server or on an external server. You can define the sources of the data that you would like to leverage in your SharePoint site. As previously mentioned, these data sources can take a variety of forms including SharePoint lists and SharePoint libraries.

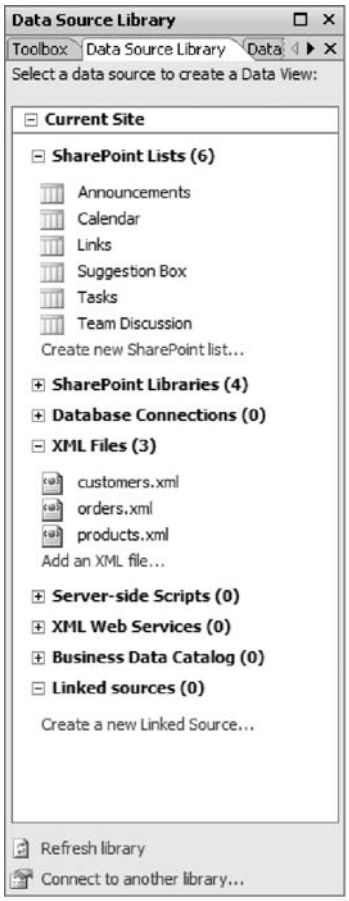


Figure 13.9 *Data source library task pane.*

The Data Source Library task pane groups each data source into a corresponding category. Figure 13.9 shows the Data Source Library for a typical SharePoint team site. By default, the catalog is divided into eight categories:

- *SharePoint Lists*—SharePoint Designer automatically creates entries for the SharePoint lists based on the WSS site contents. When you delete a list from the SharePoint site, it is automatically removed from the Data Source Catalog. You can create a new list to add to the Data Source Catalog using the Create new SharePoint List link. You can also add lists from other WSS sites, including sites in different site collections as content sources; you are not restricted to the current site.
- *SharePoint Libraries*—SharePoint Designer automatically creates entries for the SharePoint document libraries based on the WSS site contents. To remove theses entries from the Data Source Catalog, you must delete the actual document library from the SharePoint site. However, you can create a new document library to add to the Data Source Catalog using the Create new Document Library link. Once again, you are not restricted to the current site.
- *Database Connections*—The database connections category lists OLE DB database data sources, such as Access, SQL Server, and Oracle, that have been added to the Data Source Catalog. The Database Connections category is empty by default, but you can easily add connections by clicking the "Connect to Database" link associated with the data source.
- *XML Files*—SharePoint Designer automatically displays any XML files stored in any document library within your site under the XML files category. Figure 13.9 shows three XML files that we have loaded into a document library in a team site. The XML files category also lists the XML files that have been added as data sources to the Data Source Library. These files include Word, Excel and InfoPath documents that have been saved as XML documents.
- *Server-side Scripts*—The server-side scripts category lists the server-side scripts that have been added to the Data Source Library. These scripts should return data in the form of XML so that consumers, such as the data view Web Part, can use the Data Retrieval Service to work with the data. The server-side scripts category is empty by default, but you can easily add to it by clicking the Connect to a script or RSS feed link.
- *XML Web Services*—The XML Web services category lists the XML Web services that have been added to the Data Source Library. The

XML Web Services category is empty by default, but you can easily add to it by clicking the Connect to a Web Service link.

- *Business Data Catalog*—The business data catalog category lists the views associated with entities registered in the SharePoint business data catalog from which you can retrieve data. You can easily add additional views by clicking the Create a new view link.
- *Linked Data Sources*—The Linked Data Sources category lists the linked sources that have been added to the Data Source Library. Using the Create a new Linked Source link, you can merge or join multiple data sources into a combined data source so that you can easily drag and drop data onto the data view.

13.3.3 Linked Data Sources

The FrontPage 2003 data view Web Part was very powerful for retrieving data from a single data source, but it did not provide an easy way for you to retrieve data from multiple data sources into a single data view. The data view Web Part that ships with SharePoint Designer 2007 introduces the concept of linked data sources. Linked data sources allow you to merge or join interrelated data from different data sources into a single data source that appears as a source in the Data Source Library. These data sources can be any of the data sources supported by the data view Web Part, such as lists, document libraries, and external databases. Let's look at the simple example of two XML files containing related data. Customer.xml contains information about customers, such as their customer `ID`, their name, address, and so forth. Orders.xml contains information about the orders each customer has placed—for example, the customer's `ID`, the date of the order, the order number, the items in the order, the price, the quantity, and an overall order total. Each XML file has the customer `ID` in common, which relates the data from one source to the other. Using the Link Data Sources wizard, you can easily create a single linked data source that matches the customer information in Customer.XML to the order information in Orders.xml. The key field names do not have to have the same name; when you insert a joined subview, a wizard allows you to pick the matching field from each data source. If the customer `ID` field in Customer.XML was named `CustID` and the customer `ID` field in Orders.XML was named `ID`, the wizard allows you to match `CustID` to `ID` through its user interface.

When linking data sources you have two options: you can merge the data sources or join the data sources. If you have two data sources with the same set of fields and want to combine them into a single data source, you would choose the merge option. Figure 13.10 shows two tables containing product information. The first table contains information on products 1 through 5, whereas the second table contains information on products 6 thru

10. The merge option takes the data from both tables and combines it into a single table containing information on products 1 through 10. Note that the field data in both tables is the same.

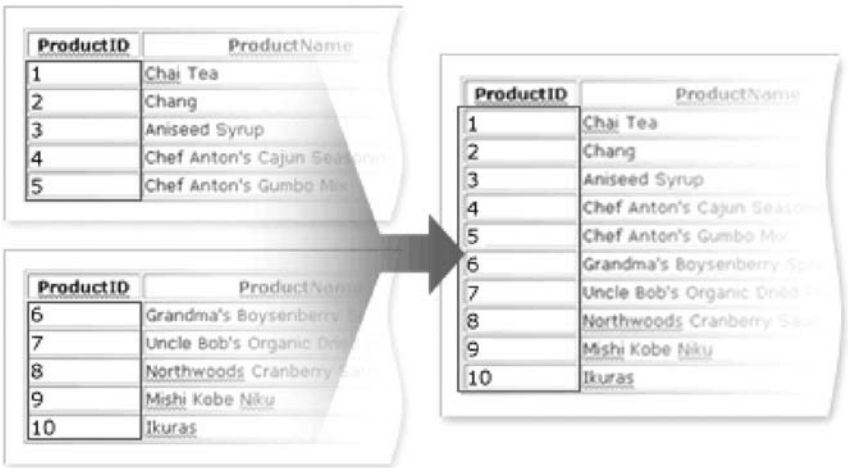


Figure 13.10
Merging related data sources.

If you have two separate data sources with a field in common and want to combine them into a single data source, you would choose the join option. Figure 13.11 shows two tables, a products table and a categories table. The field they share in common is the `CategoryID` field. The join option takes the data from both tables and joins them on `CategoryID`, producing a single table of information. Using the combined view, you can browse through each category and look at the associated product details.

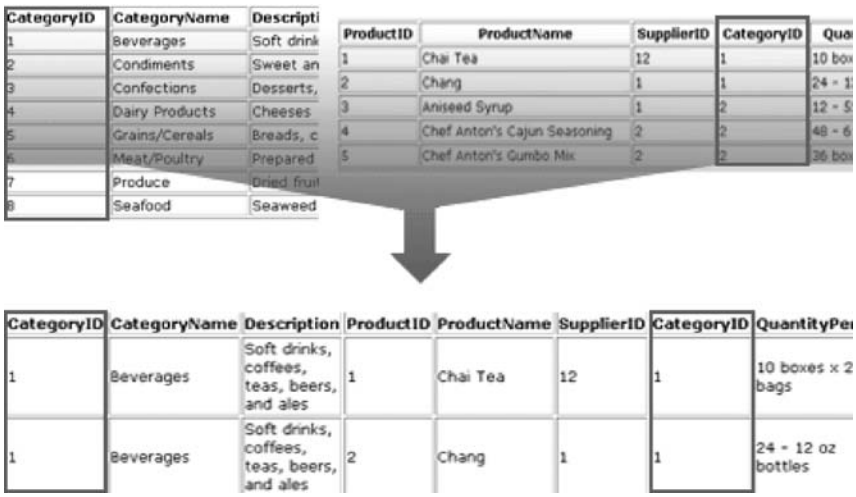


Figure 13.11
Joining related data sources.

13.3.4 Creating a Data View with a Linked Data Source

The following section walks you through the process of creating a data view from a linked data source. For this walkthrough, we will use the two XML files, Customers.XML and Orders.XML, mentioned earlier in this section. You can locate these sample files from the MSDN archives XML Order Application or create your own.

1. Upload your XML files to a document library in your WSS 3.0 site.
2. Create an empty Web Part page in your site. You can choose to create the Web Part page in the same document library that you uploaded the XML files or to create a new document library.
3. Navigate to the newly created Web Part page.
4. From the Internet Explorer File menu click Edit with Microsoft Office SharePoint Designer.
5. Click the zone on your page where you would like to insert your data view Web Part.
6. From the Data View menu, click Insert Data View. The Data Source Library task pane appears on the right.
7. From the Data Source Library task pane, click the Create a New Linked Source link. The Data Source Properties dialog appears.
8. Click Configure Linked Source. The Link Data Sources wizard appears.
9. From the XML Files section in the Available Data Sources column select the Customer.XML data source.
10. Click Add.
11. From the XML Files section in the Available Data Sources column, select the Orders.XML data source.
12. Click Add.
13. Click Next.
14. From the Link Data Source wizard, click the Join the contents of the data sources by using the Data Source Details to insert data views and joined subviews option.
15. Click Finish.
16. From the Data Source Properties dialog, click the General tab.
17. In the Name field, enter a name for the linked data source. You may also enter a description.

18. Click OK. The new data source appears in the linked data sources section of the Data Source Library.
19. Right-click the new data source and click Show Data. The data source details pane appears.
20. From the customer table select the fields you would like to insert into the data view—for example, select ID, first-name, and last-name.
21. From the Insert Selected Fields as menu at the top of the Data Source Details task pane, click Multiple Item View. The data view appears on the page with the selected fields displayed (Figure 13.12).

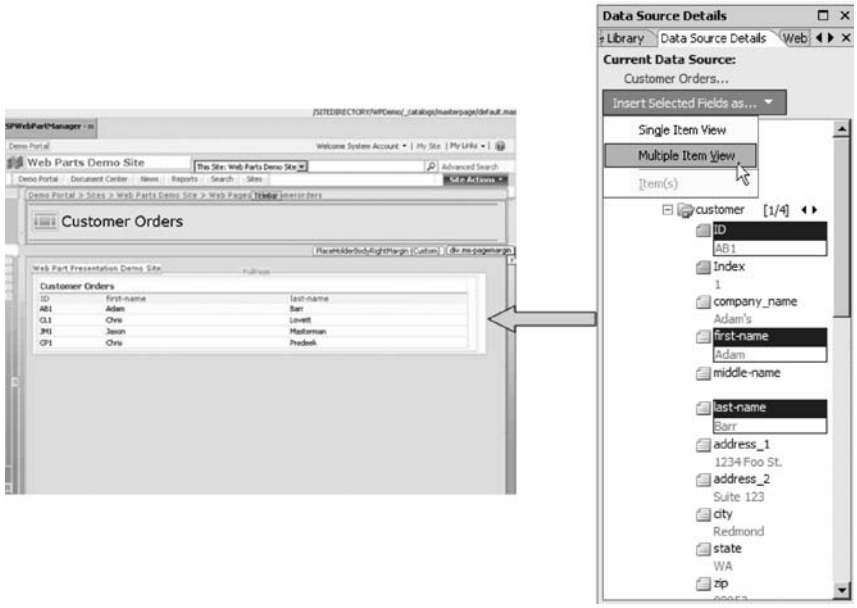


Figure 13.12 *Insert multiple item view.*

22. Insert an additional column to the right of the last name column.
23. Click the first data line (not the header) in the new column.
24. From the Data Source Details task pane, navigate to the orders table and select the fields you would like to insert into the data view—for example, Order_Number, Date, and Total Cost.
25. From the Insert Selected Fields as menu at the top of the Data Source Details task pane, click Joined Subview. The Join Subview dialog appears. This dialog allows you to match the fields that contain related data.

26. From the customer column select the ID field.
27. From the Order_Number column click more fields. The More Fields dialog appears.
28. Navigate and select the Cust_ID field from the Orders table.
29. Click OK.
30. Click OK again. The data view appears on the page with the selected fields displayed (Figure 13.13).

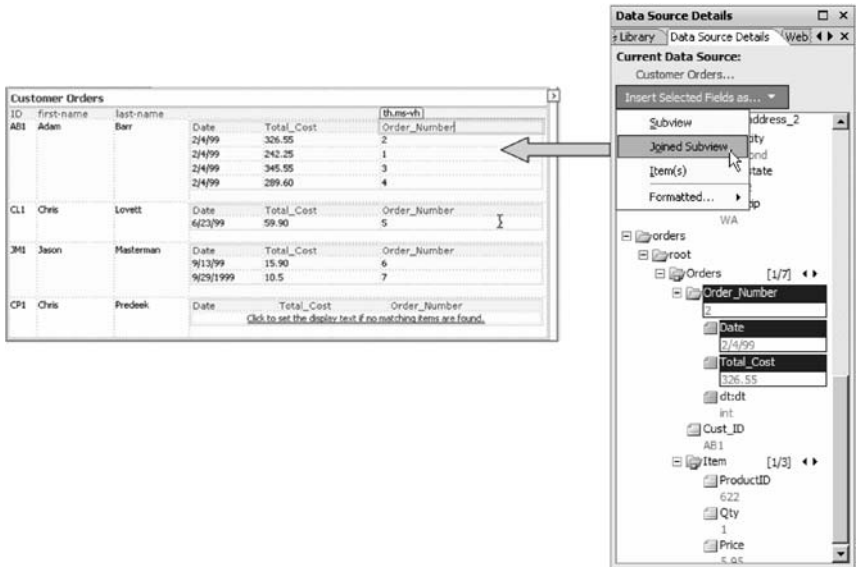


Figure 13.13 *Insert joined subview.*

31. Insert an additional column to the right of the `Order_Number` column.
32. Click the first data line (not the header) in the new column.
33. From the Data Source Details task pane, navigate to the orders table and select the item fields you would like to insert into the data view—for example `ProductID`, `Qty`, and `Price`.
34. From the Insert Selected Fields as menu at the top of the Data Source Details task pane, click Joined Subview. The Join Subview dialog appears. This dialog allows you to match the fields that contain related data.
35. From the `Order_Number` column, select the `Order_Number` field.
36. From the Item column click more fields. The More Fields dialog appears.

37. Navigate and select the same `Order_Number` field from the Orders table.
38. Click OK.
39. Click OK again. The data view appears on the page with the selected fields displayed (Figure 13.14).
40. Save your Web page and view the results in your browser.

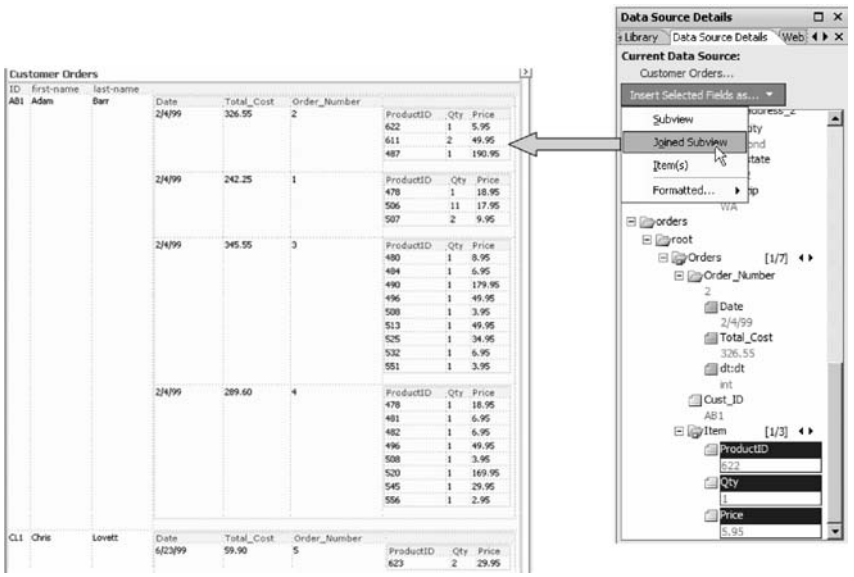


Figure 13.14 *Insert another joined subview.*

The architecture deployed with SharePoint 2007, in conjunction with the SharePoint Designer 2007 data view Web Part, provides an excellent vehicle for producing data-driven Web sites. The Data Source Library allows you to define multiple data providers and keep track of these providers regardless of where they live. With the new addition of linked data sources, you can now combine related data sources into a single data source for presentation to the user. This is a very powerful and welcome addition to the Web designer's toolkit.

13.3.5 SharePoint Server 2007 Content Query Web Part

The content query Web Part is a Web Part that displays a dynamic view of the content in your site. The properties of the Web Part determine the content that is returned and in what format it is presented. For example, you can set the scope of the query to the entire site collection, a specific site, or a specific list. Unfortunately, the scope cannot be set to retrieve content across multiple site collections; however, other Web Parts, such as the data view

Web Part or the search core results Web Part, can be configured to retrieve content from outside a single site collection. Once you have the scope defined, you can go on to specify a type of list, specify a content type, and even define your own custom filters to apply to the query. Using the properties of the content query Web Part, you can sort and group the results and select custom styles to control the presentation of the data retrieved from the query. Figure 13.15 shows the properties associated with the Web Part, and Figure 13.16 shows the resulting output in the content query Web Part. The presentation tool part contains an area for selecting the styles that you would like to associate with the Web Part. The Group Style drop-down box represents a list of XSL templates that are defined in the file Header.xsl, and the Item Style drop down box represents a list of XSL templates that are defined in the file ItemStyle.xsl. Both files are located in the root of the site collection in the XSL style sheets style library (..\`Style Library\XSL Style Sheets`). You can provide your own custom look and feel to the data retrieved from the content query Web Part by adding custom XSL templates to the Header.xsl and ItemStyle.xsl files directly, or by creating your own custom XSL style sheets and referencing those in the Web Part. SharePoint Designer 2007 provides a rich environment for editing these style sheets, while at the same time honoring the content management aspects of the style library.

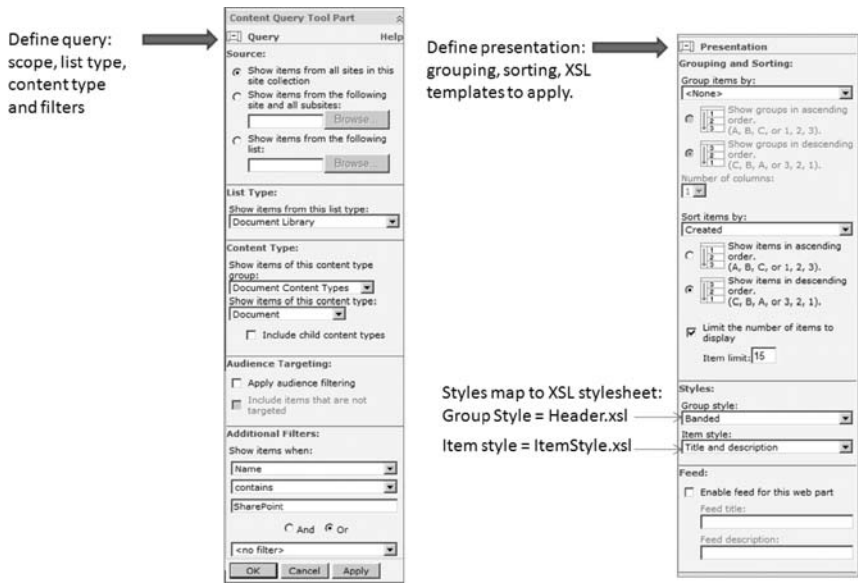


Figure 13.15 *Content query Web Part tool pane.*

At first glance the capabilities of this Web Part look fairly rudimentary. The fact that we can now finally roll up data from a variety of sources and present them through a single Web Part on the portal home page is great progress, but what if you want to do more than what's configurable through

the tool part properties? What if you want to display data from custom fields, or return data from multiple list *types* and content *types*? Are you back to writing your own custom Web Parts or pursuing third parties? Fortunately the answer to this question is a resounding "No." Upon closer inspection, the content query Web Part is extremely versatile and customizable; there are a host of configurable properties associated with the content query Web Part that are accessible from the .Webpart file. For example, you can list all the custom site columns, or fields, that you would like to be presented in the Web Part by configuring the `CommonViewFields` property. In fact, the output shown in Figure 13.16 displays data from a custom site column named Synopsis that has been added to the document libraries throughout the site collection. The .Webpart file contains the appropriate entry in the `CommonViewFields` property informing the content query Web Part to retrieve the data for the Synopsis field (Figure 13.17).

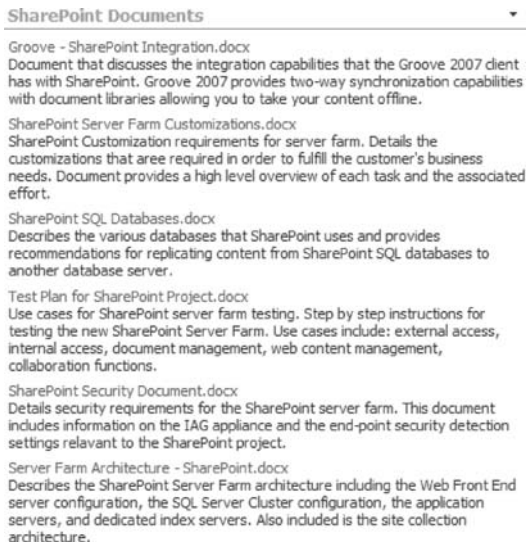


Figure 13.16 *Results of content query Web part.*

```
<property name="CommonViewFields" type="string">Synopsis, Note</property>
```

Figure 13.17 *CommonViewFields property.*

Table 13.1 lists the properties that can be configured in the .Webpart file. In order to customize these properties you must first add the default out of the box content query Web Part to your Web page, export the file to your desktop, edit the file and the relevant properties, save the changes, and then upload the customized .Webpart file to the Web Part gallery. We will be

Table 13.1 *content query Web Part Properties*

Property	Description
CommonViewFields	Requests additional fields, including custom fields. If you use a custom field in the XSL style sheet, you must first list it in the `CommonViewFields` property for it to display. The content query Web Part retrieves a limited set of fields by default; additional fields must be requested using this property. Fields are added using their internal name or a combination of their internal name plus the field type. You can add more than one field, separated with a semicolon. For example: ProductName, Text; ProductDetail, Text
DataColumnRenames	Allows the content query Web Part to rename fields before they are passed to the XSL style sheet for processing. Some internal field names are quite lengthy, or have characters such as `_x0020_` in them. You can use `DataColumnRename` to replace the internal name with a friendlier name for rendering. The internal name still must be listed in the `CommonViewFields` property.
QueryOverride	Allows you to override the query settings defined in the tool part properties with your own custom CAML query.
WebsOverride	Allows you to determine whether the content query Web Part retrieves data from subsites of the parent site. By default, the Web Part retrieves data from the parent site and all child sites.
ListsOverride	Allows you to determine the type of list, or a specific set of lists, from which to return data.
ViewFieldsOverride	Allows you to specify a list of fields that will override the default base properties returned by the content query Web Part. This property is used in conjunction with the `QueryOverride` property.
AdditionalFilterFields	Allows you to add fields to appear in the drop-down filter in the additional filters tool part. This property cannot be used in conjunction with the `QueryOverride` property.
AdditionalGroupAndSortFields	Allows you to add fields to appear in the drop down boxes in the grouping and sorting tool part. This property cannot be used in conjunction with the `QueryOverride` property.

Table 13.1 *content query Web Part Properties (continued)*

Property	Description
Filter Properties FilterField[n] FilterType[n] FilterValue[n] FilterOperator[n] Filter[n]ChainingOperator	A set of filter properties that allow you to further filter fields beyond the capabilities offered through the user interface. n represents the filter number, with numbers ranging from 1 to 3. These fields work in conjunction with one another based on the numbering. For example `FilterField1`, `FilterType1`, `FilterValue1`, `FilterOperator1` and `Filter1ChainingOperator` all work together to form the filter for the query. If you specify custom columns in the `FilterField`, you need to have the column defined in the `AdditionalFilterFields` property.

walking you through these steps shortly, but for now let's discuss the purpose of these properties.

By default, the content query Web Part retrieves only a subset of the fields, or columns associated with an item; fields that are not part of the default set returned, including custom fields, require explicit listing in the .Webpart file using the `CommonViewFields` property. It is understandable that *custom* columns require explicit listing, but what about columns that are defined out of the box with SharePoint? How do you know which of those columns are returned by default and which you have to explicitly list? During the initial design stages you can add a few simple lines of XML code to a style template in the ItemStyles.xsl file and use it to determine the fields returned. Figure 13.18 shows the XML code, and Figure 13.19 shows the resulting output. From this output we can figure out which fields are available for use without explicit listing in the `CommonViewFields` property; our example includes fields such as Author, Created, Comments, Description, and Editor. If we wanted to list the user that currently had the document checked out, we would need to explicitly list the `CheckoutUser` field in the `CommonViewFields` property; `CheckoutUser` is the internal name of the Checked Out To column in a document library. Fields listed for the `CommonViewFields` property should have a corresponding entry in the ItemStyle.xsl to handle the presentation of the data returned from that field. The reverse is also true; any fields referenced in the ItemStyle.xsl that are not returned as part of the default set of fields should also be listed in the `CommonViewFields` property of the .Webpart file.

Figure 13.18 *XML to return default fields.*

```
<xsl:for-each select="@*">
  Prop:<xsl:value-of select="name()" /> |
</xsl:for-each>
```

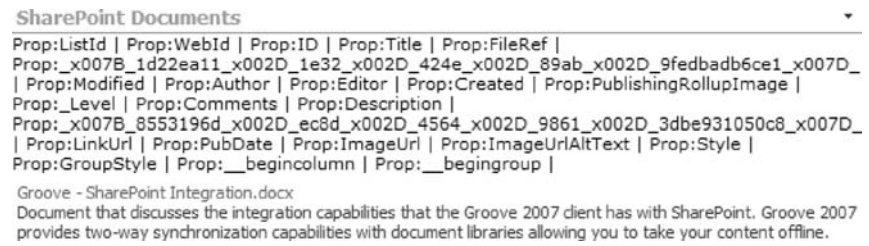


Figure 13.19 *Default fields displayed in content query Web Part.*

The `CommonViewFields` property requires the internal name of the column to be referenced. So our next question, inevitably, is how to determine the internal name of a site column. There are a couple of methods for retrieving this information, the most basic being through the browser user interface. Navigate to the location of the column definition; it could be a column associated with a content type, a column mapped to a document library via the document library settings page, or a site column. For the purposes of our example, we'll assume it's a column associated with a document library.

1. Navigate to the document library containing the desired column and click on the Document Library Settings link from the Settings menu.
2. From the Columns section, right-click the column that you wish to retrieve the internal name. For example, right-click Checked Out To.
3. Click Properties. A properties dialog box appears, as shown in Figure 13.20.
4. In the General section, scroll to the end of the name property until you see a string that equals `&Field=`. The text to the right of this string is the internal name of the column. Take note of the internal name for use in the .Webpart file and XSL style sheet.

Sometimes you may find that the internal name returned by the above technique does not return content as expected. In this case, it is worthwhile to double-check the internal name to ensure that you are using the correct name. In this particular situation, it is better to go to the actual schema files (`ONET.XML` and `Schema.xml`) that define the lists. The `ONET.XML` file associated with the `GLOBAL` site definition contains many of the base field definitions, such as `CheckOutUser`. Reference this file to check for the internal name of out-of-the-box SharePoint fields. The `GLOBAL` site definition is located in the `\Program Files\Common\Microsoft Shared\Web server extensions\12\TEMPLATE\GLOBAL` directory. If you do not find a reference to the field in this schema file, your next stop is the list definition, typically

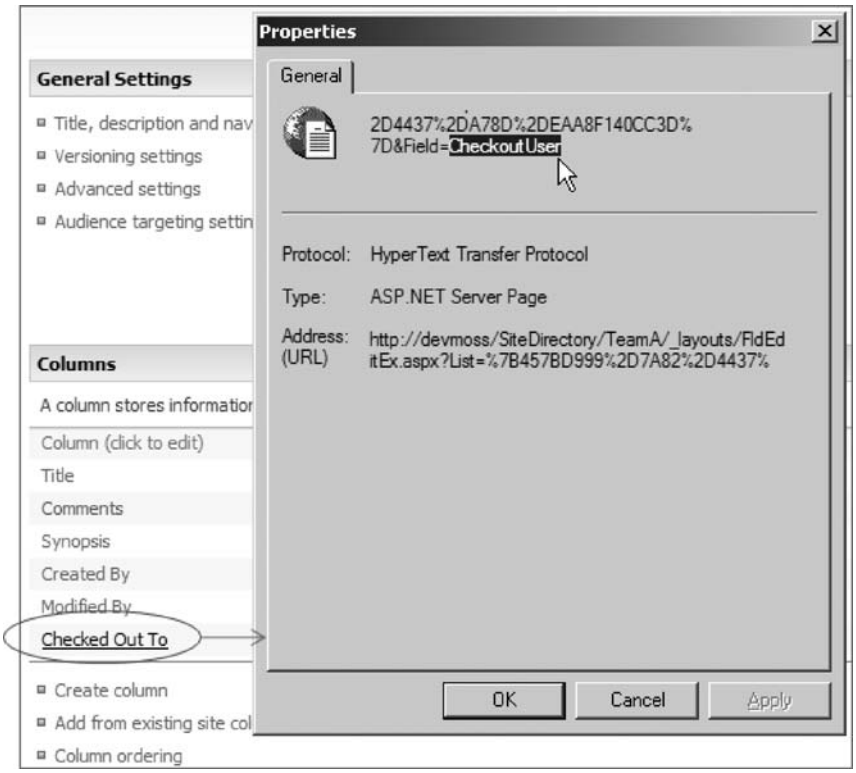


Figure 13.20 *Determining the internal field name.*

represented by a feature. Feature files are located in the `\Program Files\Common\Microsoft Shared\Web server extensions\12\TEMPLATE\FEATURES` directory. Navigate to this directory and search for a folder that would most likely represent the list or content type containing the field in which you are interested. For example, the folder name DocumentLibrary contains the feature definition for a document library in SharePoint. Examine the schema file associated with the list, for example Schema.xml, and search for the field reference. More information on features and the GLOBAL site definition can be found in Chapters 15 and 16.

When you list the custom field in the `CommonViewFields` property, you typically provide a corresponding entry in the ItemStyle.xsl to handle the presentation of the data returned from that field. We say "typically" because there are instances when you may not actually need to define a completely new style; instead, you could use a specific style from an existing data field instead. Let's revisit the Synopsis field that we defined in an earlier example. The Synopsis field is a custom site column that is applied to some document libraries within a site collection. The field is used to contain description information about the content of the document. In this particular scenario,

the Synopsis field behaves like the description field. The Title and Description style is ideally suited for displaying this data, but we would want the content query Web Part to replace the Description field with content from the Synopsis field. We could edit the ItemStyle.xsl and replace the reference to `@Description` with a reference to `@Synopsis` in the Title and Description template, but that would mean the Title and Description style would no longer render content from the Description field. An easier and less intrusive method involves leveraging the `DataColumnRenames` property of the .Webpart file (see Figure 13.21). The `DataColumnRenames` property allows you to rename fields before they are passed to the XSL style sheet for rendering. In our scenario, we want to rename the Synopsis field as the Description field so that, as far as the XSL style sheet is concerned, it is rendering content for the Description field. This means that we can use the Title and Description style when rendering content containing the Synopsis field; there is no need to create a separate template. Some internal field names are quite lengthy, or have characters such as `_x0020_` in them. Another great use of the `DataColumnRenames` property is that you can use it to replace the internal name of a field with a friendlier name for rendering purposes. Note that when using the `DataColumnRenames` property, the internal name still must be listed in the `CommonViewFields` property if it is not one of the default fields returned by the content query Web Part.

```
<property name="DataColumnRenames" type="string">Synopsis,Description</property>
```

Figure 13.21 *DataColumn Renames property.*

The tool part properties of the content query Web Part allow you to add additional filters to your query. For example, if you look back at Figure 13.15 you will notice that an additional query has been added to inform the content query Web Part to return all documents that contain the word "SharePoint" in their name. Using this same filter, you can also filter the content to a specific date. For example, you can add a filter that will only retrieve the documents that were created today, or retrieve the documents that were created before a specific date. However, you cannot specify a date range. For example, using the filter properties in the Web Part tool pane, you cannot add a filter that retrieves the documents created in the past five days. But just because you cannot easily achieve this functionality via the Web Part tool pane interface does not mean it cannot be done. The `QueryOverride` property allows you to apply additional query logic, such as adding a date range filter to the query. When the `QueryOverride` property is defined, certain settings such as the filter, grouping and sorting fields in the Web Part tool pane are dimmed, meaning that you can no longer apply these settings through the tool pane user interface. The `QueryOverride` property accepts a query in Collaborative Application Markup Language

(CAML) format. CAML is the XML based language used throughout many of the SharePoint schema files. The query should be enclosed in a `CDATA` block, which tells the XML processor to ignore the tag and entity references contained within the block. Figure 13.22 shows the CAML query required to retrieve items created within the last five days; the results are sorted so that the most recent creations appear at the top of the list. Note that you cannot have carriage returns or unwarranted spaces within the query.

```
<property name="QueryOverride" type="string"><![CDATA[<Where><Gt><FieldRef
Name="Created" Type="DateTime"/><Value Type="DateTime"><Today
OffsetDays="-5"/></Value></Gt></Where><OrderBy><FieldRef Name="Created" Type="DateTime"
Ascending="FALSE"/></OrderBy>]]></property>
```

Figure 13.22 *QueryOverride property.*

Using the `WebsOverride` property, you can determine whether the content query Web Part will traverse any subsites of the current site; the default behavior retrieves results from the parent site and all child sites. Figure 13.23 shows how to turn this behavior off and restrict the Web Part to retrieving content from the current site only.

```
<property name="WebsOverride" type="string"><![CDATA[<webs/>]]></property>
```

Figure 13.23 *WebsOverride property.*

The `ListsOverride` property allows you to define a specific set of lists or a specific list type from which to retrieve content. For example, Figure 13.24 tells the content query Web Part to only retrieve content from the list whose ID matches the specified list ID. Figure 13.25, on the other hand, tells the Web Part to only retrieve content from a specific list type; in this example, the list type is a document library.

```
<property name="ListsOverride" type="string"><![CDATA[<Lists><List
ID="457BD999-7A82-4437-A78D-EAA8F140CC3D"/></Lists>]]></property>
```

Figure 13.24 *ListsOverride property—defining a specific list.*

```
<property name="ListsOverride" type="string"><![CDATA[<Lists
ServerTemplate="101"</Lists>]]></property>
```

Figure 13.25 *ListsOverride property—defining a list type.*

As previously mentioned, the content query Web Part by default returns a base set of properties. Figure 13.19 shows the enumeration of these proper-

ties using an XSL style sheet. The `ViewFieldsOverride` property allows you to override the default fields returned by the Web Part and replace them with your own list of preferred fields. The `ViewFieldsOverride` property works in conjunction with the `QueryOverride` property.

The `AdditionalFilterFields` property allows you to specify additional fields to appear in the drop-down filter in the Additional Filters tool part. Note that this property cannot be used in conjunction with the `QueryOverride` property, because when the `QueryOverride` property is implemented, the Additional Filters tool part is disabled. Figure 13.26 shows how to add a custom field to the drop-down list and the corresponding output in the Additional Filters editor part of the Web Part tool pane.

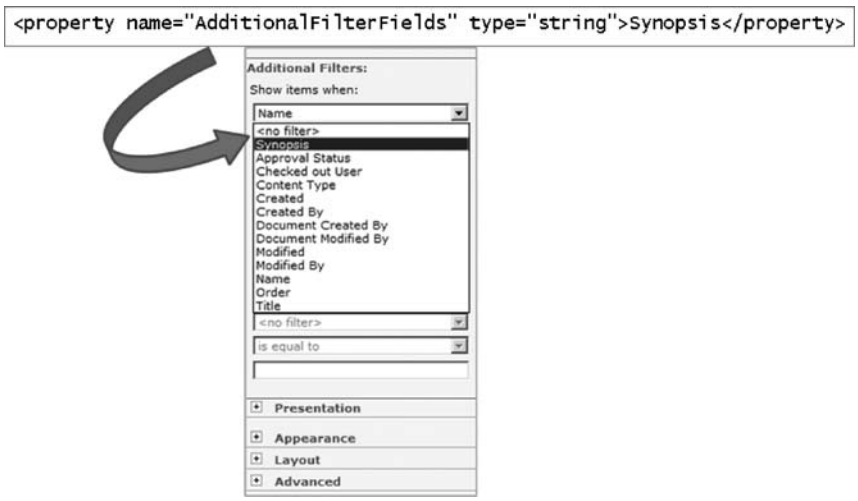


Figure 13.26 *AdditionalFilterFields Property and resulting output.*

The `AdditionalGroupAndSortFields` property allows you to add additional fields to appear in the drop-down boxes in the Grouping and Sorting tool part. Once again, this property cannot be used in conjunction with the `QueryOverride` property, because the `QueryOverride` property disables this portion of the Web Part tool pane interface. Figure 13.27 shows how to add a custom field to the drop-down list and the corresponding output in the Grouping and Sorting editor part.

In the Query section of the content query Web Part you will notice that there are three sets of possible filters that you can use to build your filter queries. The `AdditionalFilterFields` property allows you to add custom fields to the drop-down lists in each of the filters. We can further define and prepopulate this section of the query using a set of filter properties that work in unison to form the filter. The filter is typically comprised of four properties: `FilterField[n]`, `FilterType[n]`, `FilterOperator[n]`, `Filter-`

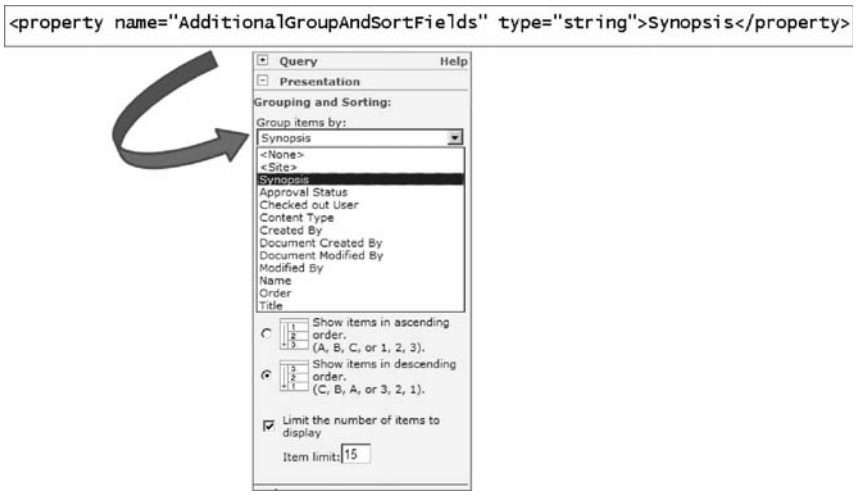


Figure 13.27 *AdditionalGroupAndSortFields Property and resulting output.*

`Value[n]`, and `Filter[n]ChainingOperator`, where `[n]` is replaced with the number 1, 2, or 3. `FilterField1` maps to the selected field in the first drop-down box of the Additional Filters tool part, `FilterField2` maps to the selected field in the second drop-down box, and `FilterField3` maps to the selected field in the third drop-down box. `Filter1Type` represents the type of the `FilterField1` field. For example, if `FilterField1` was set to Synopsis, then the `Filter1Type` would be set to Note. If you are using custom fields in the filters fields, the custom fields must also be listed in the `AdditionalFilterFields` property. `FilterOperator1` maps to the operator drop-down box that contains operators such as "is equal to" and "contains." Table 13.2 shows the possible values for these fields and their corresponding display in the Web Part tool pane. The `FilterValue1` property maps to the text box that appears below the operator drop down in the tool pane and is used to contain the value with which you want to filter your query. For example, if we wanted to filter our query so that it only displays those items in which the Synopsis contains the word SharePoint, then we would set the `FilterField1` property to Synopsis, the `FilterType1` property to Note, the `FilterOperator1` property to contains, and the `FilterValue1` property to SharePoint. If we then wanted to chain this filter with a second filter we would set the Filter1ChainingOperator to one of two values: And, or Or. Figure 13.28 shows the filter property settings in the .Webpart file and Figure 13.29 shows the resulting output in the Web Part tool part.

The presentation of data retrieved by the content query Web Part is ultimately controlled by two style settings: the group style and the item style. By default, the style templates presented in the group style drop-down list are provided by the Header.xsl style sheet stored in the site collection's

Table 13.2 *FilterOperator[n] Values*

Value	Tool Part Display
Eq	is equal to
Neq	is not equal to
Gt	is greater than
Geq	is greater than or equal to
Lt	is less than
Leq	is less than or equal to
BeginsWith	begins with
Contains	contains

Figure 13.28 *Filter property settings to pre-populate filter in tool pane.*

```
<property name="FilterField1" type="string">Synopsis</property>

<property name="FilterType1" type="string">Note</property>

<property name="FilterOperator1"
type="Microsoft.SharePoint.Publishing.WebControls.ContentByQueryWebPart
+FilterFieldQueryOperator, Microsoft.SharePoint.Publishing,
Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c">Contains</property>

<property name="FilterValue1" type="string">SharePoint</property>

<property name="Filter1ChainingOperator"
type="Microsoft.SharePoint.Publishing.WebControls.ContentByQueryWebPart
+FilterChainingOperator, Microsoft.SharePoint.Publishing,
Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c">And</property>

<property name="FilterField2" type="string">Created</property>

<property name="FilterType2" type="string">DateTime</property>

<property name="FilterOperator2"
type="Microsoft.SharePoint.Publishing.WebControls.ContentByQueryWebPart
+FilterFieldQueryOperator, Microsoft.SharePoint.Publishing,
Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c">Eq</property>

<property name="FilterValue2" type="string">[Today]</property>
```

style library. The style templates presented in the item style drop-down list are provided by the ItemStyle.xsl style sheet, also stored in the site collection's style library. If you want to provide your own style sheet templates, you can use SharePoint's document management capabilities to modify these files directly through SharePoint Designer. When you edit the files in this manner, you detach them from the site definition files, causing them to become customized files. This detachment means that if a future service

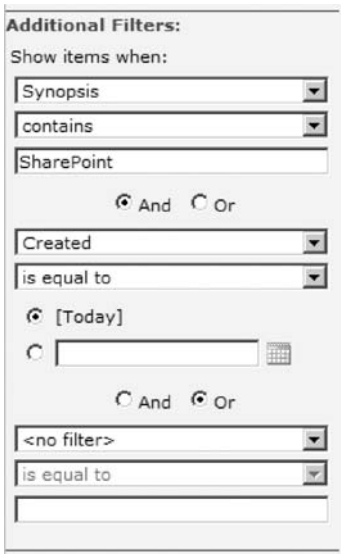


Figure 13.29 *Filters set in Additional Filters editor part.*

pack updates the ItemStyle.xsl or Header.xsl files by adding new templates, the updates will no longer be accessible to the sites within the site collection. A better method for applying custom style sheet templates is to create your own XSL style sheets and reference them directly in the content query Web Part by leveraging the `HeaderXslLink` and `ItemXslLink` properties. Figure 13.30 shows the property settings to override the ItemStyle.xsl file with a custom style sheet named MyCustom.xsl, also stored in the site collection's style library. Note that the custom style sheet *overrides* the default style sheet; if your custom style sheet contains one XSL template, then only one style will be presented through the style drop-down list in the Web Part tool pane. If you want to make the styles that appear in the default style sheet available as well as your own custom styles, then the easiest option is to copy and paste the content from the default style sheet file (e.g. ItemStyle.xsl) into your custom style sheet and add your style templates to this content.

```
<property name="ItemXslLink" type="string">/Style Library/XSL Style
Sheets/MyCustom.xsl</property>
```

Figure 13.30 *ItemXslLink property.*

Let's run through a brief exercise to show how easy it is to use the content query Web Part. To follow along with this exercise, you must perform a few preparatory steps before you use the Web Part.

13.3.6 Exercise Preparation

1. Navigate to the root of your site collection and create a custom site column named Synopsis, with the type set to Multiple Lines of Text.
2. Navigate to the Shared Documents document library of three different sites within the site collection and add the Synopsis column to the document library.
3. In each Shared Documents document library, upload a couple of documents and populate the Synopsis field with text. For some documents, make sure that the Synopsis field contains the word SharePoint; for others, make sure that just the name of the document contains the word SharePoint.

13.3.7 Content Query Web Part Exercise

1. Navigate to the home page of the root of your site collection.
2. From the Site Actions menu, click Edit Page.
3. From the middle left zone of the Web Part page, click Add a Web Part.
4. Select the content query Web Part, and click Add. The content query Web Part is added to the middle left zone on the Web Part Page, and is configured with the default settings.
5. From the Web Part Edit menu, click Export…
6. Export the .Webpart file to your desktop or another location of choice.
7. From the Web Part Edit menu, click Delete, to delete the newly added content query Web Part.
8. Open the newly exported .Webpart file in Notepad for editing.
9. Using the Find option, locate the Title property and set the property as shown in Figure 13.31.

```
<property name="Title" type="string">SharePoint Documents</property>
```

Figure 13.31
Title property.

10. Using the Find option, locate the `CommonViewFields` property and set the property as shown in Figure 13.17.

11. Using the Find option, locate the `DataColumnRenames` property and set the property as shown in Figure 13.21.
12. Using the Find option, locate the `AdditionalFilterFields` property and set the property as shown in Figure 13.26.
13. Using the Find option, locate the `ItemXSLLink` property and set the property as shown in Figure 13.30.
14. Save the changes to the .Webpart file.
15. Go back to your browser and navigate to the style library in the root of your site collection. You can get to the style library by selecting Manage Content and Structure from the Site Actions menu at the root of the site collection.
16. Navigate to the XSL Style Sheets folder within the style library.
17. Save a copy of the ItemStyles.xsl to your desktop and rename it to MyCustom.xsl
18. Using Notepad, edit the MyCustom.xsl file and insert the XSL template code shown in Figure 13.32. Insert the code so that it is the first XSL template definition in the file. This means that the SpysRUs style template will appear first in the Item Styles drop-down list. Notice that we have added a `ddwrt` reference to the V2 DataView runtime; the DataView runtime contains some handy date and time formatting tools that we leverage in our example.

Figure 13.32 *SpysRUs XSL template.*

```
xmlns:ddwrt="http://schemas.microsoft.com/WebParts/v2/DataView/runtime">
  <xsl:template name="SpysRUs" match="Row[@Style='SpysRus']" mode="itemstyle">
      <xsl:variable name="SafeLinkUrl">
          <xsl:call-template name="OuterTemplate.GetSafeLink">
              <xsl:with-param name="UrlColumnName" select="'LinkUrl'"/>
          </xsl:call-template>
      </xsl:variable>
      <xsl:variable name="DisplayTitle">
          <xsl:call-template name="OuterTemplate.GetTitle">
              <xsl:with-param name="Title" select="@Title"/>
              <xsl:with-param name="UrlColumnName" select="'LinkUrl'"/>
          </xsl:call-template>
      </xsl:variable>
      <xsl:variable name="LinkTarget">
          <xsl:if test="@OpenInNewWindow = 'True'" >_blank</xsl:if>
      </xsl:variable>
      <div class="mytitle-With-Background">
          <xsl:call-template name="OuterTemplate.CallPresenceStatusIconTemplate"/>
          <a href="{$SafeLinkUrl}" target="{$LinkTarget}" title="{@LinkToolTip}">
              <xsl:value-of select="$DisplayTitle"/>
          </a>
           <div class="synopsis-With-Background">
               <xsl:value-of select="@Synopsis"/>
               <div class="description">
                   Created <xsl:value-of select="ddwrt:FormatDateTime(string(@Created) ,1033 ,
'dddd, MMM d yyyy @ h:mm tt')"/> by <xsl:value-of select="@Author"/>
               </div>
           </div>
      </div>
  </xsl:template>
</xsl:stylesheet>
```

19. Save the changes to MyCustom.xsl.
20. Upload and publish MyCustom.xsl to the XSL Style Sheets folder in the style library.
21. Navigate back to the home page. The page should still be in edit mode; if not, edit the page again.
22. From the middle left zone of the Web Part page, click Add a Web Part.
23. From the bottom of the Add Web Parts Web page dialog, click the Advanced Web Part gallery and options link.
24. From the drop-down menu that appears next to the Browse link, click Import.
25. Click the Browse button and select the .Webpart file that you saved the changes to in step 13.
26. Click the Upload button.
27. Drag and drop the Web Part from the task pane onto the middle left zone, or another zone of your choice on the Web Part page.
28. From the Web Part Edit menu, click Modify Shared Web Part. The Web Part tool pane appears to the right.
29. Expand the Query section.
30. Set the list type equal to Document Library.
31. Set the content type to group by Document content types and show items of Document content type.
32. From the Additional Filters section, set the filters to show items when Synopsis contains SharePoint Or when Name contains SharePoint. This means that we set two filter fields with the filter chain set to Or.
33. Expand the Presentation section.
34. Set the Sort to sort items by Created, and show items in descending order.
35. From the Styles section, set the Item Style to the SpysRUs style.
36. Click OK. The resulting output should be similar to that shown in Figure 13.33.

If you find that the Synopsis field is not appearing for you, try adding a new managed property that maps the Synopsis property to the associated crawled property named ows_Synopsis property, and then rebuild the index by running a full crawl. You can access the metadata property mappings page

by navigating to the shared services administration page, which is accessible from the central administration page. From the shared services administration page, click the Search Settings link. From the search settings page, click the Metadata property mappings link and add a new managed property.

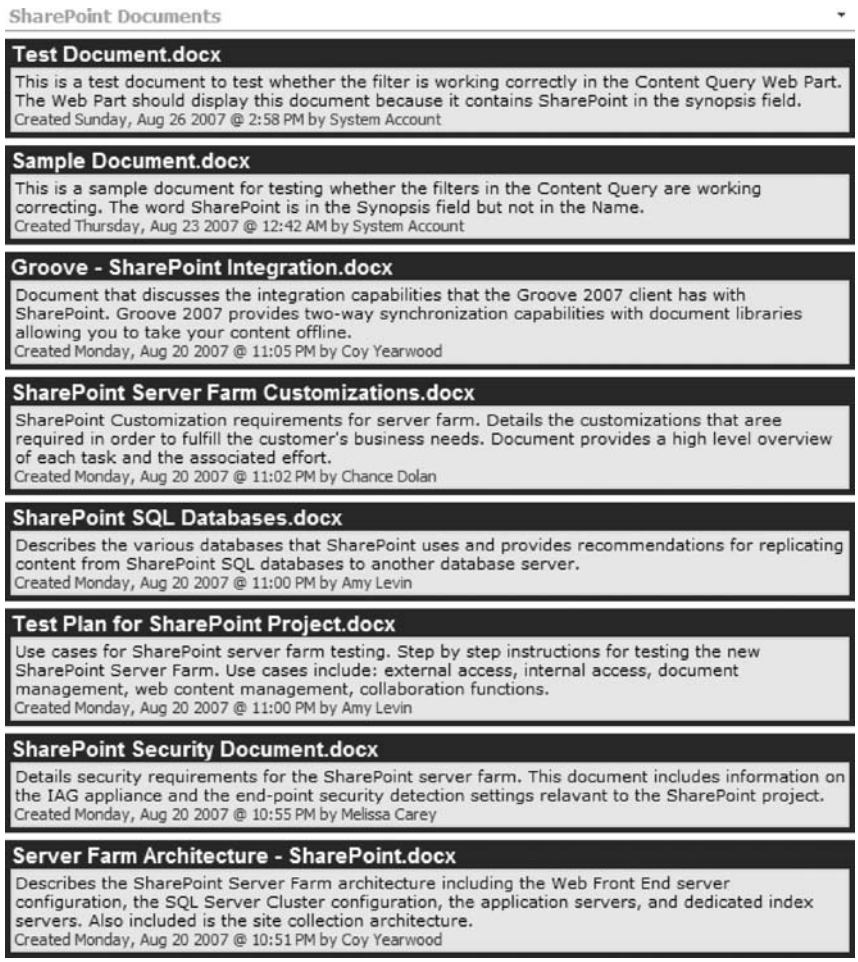


Figure 13.33 *Customized content query Web Part.*

13.4 Creating Custom Web Parts

As we can see, SharePoint comes equipped with many useful Web Parts out of the box, and many more are available from the Microsoft online gallery and through third-party vendors. However, there will be times when you need a Web Part that is not available from these sources and is very specific to your business needs. Fortunately, SharePoint is highly extensible and allows

you to develop your own custom Web Parts using industry-standard development tools such as Visual Studio 2005.

The following section steps you through the basics of how to create two simple Web Parts. The main purpose of this section is to walk you through creating your first Web Part as quickly as possible and to familiarize you with the basic steps involved in deploying a Web Part. Later in the chapter, we will discuss more advanced Web Part development techniques, such as creating custom tool parts and creating connectable Web Parts.

Since Windows SharePoint Services 3.0 and SharePoint Server 2007 are built upon ASP.NET 2.0, as developers you now have two methods of creating SharePoint Web Parts. The first method involves creating Web Parts that inherit from the ASP.NET 2.0 `System.Web.UI.WebControls.WebParts.WebPart` base class; this is the Microsoft-recommended method for Web Part development. The second method involves creating a Web Part that inherits from the Windows SharePoint Services 3.0 `WebPart` class. The WSS 3.0 `WebPart` class was carried forward from Windows SharePoint Services 2.0, and is mostly available for backward-compatibility purposes. While inheriting from the ASP.NET 2.0 is generally the best technique to adopt when developing Web Parts, there may be times that you need to use the WSS 3.0 `WebPart` class. For example, if you would like your Web Part to be able to connect with a Web Part on another page, you would inherit from the WSS 3.0 `WebPart` class because it supports cross-page connections. Scenarios that would dictate the use of the WSS 3.0 `WebPart` class rather than the preferred ASP.NET 2.0 `WebPart` class include:

- Cross-page connections
- Connections between Web Parts outside of a Web Part zone.
- Client-side connections that leverage the Web Part page services component
- Caching data to the content database

This chapter focuses on developing Web Parts using the ASP.NET 2.0 `WebPart` class. More information on developing Web Parts that use the WSS 3.0 `WebPart` class can be found in the Microsoft WSS 3.0 SDK.

13.4.1 Getting Started

Since Web Parts are basically ASP.NET Web controls, you can use Visual Studio 2005 to perform all Web Part development. To make development easier, Microsoft has provided a Web Part template that is available as part of the Visual Studio 2005 Extensions package, which can be downloaded from MSDN. This template provides skeleton code and the appropriate files and

references required for a basic Web Part; both VB.NET and C# versions are available. Once you have downloaded and installed the extensions, you can begin creating a new project using the Web Part Library template. If you would rather develop your Web Parts from scratch, you can create a new project using the Web Control Library template instead.

The following example leverages the C# Web Part template. We are about to create a simple Web Part that displays some text in a large font. As you will see, the code is very basic, with most of the work required in the setup and deployment phases. The steps assume that you are running Visual Studio 2005 on a server running Windows SharePoint Services 3.0 or SharePoint Server 2007.

Step 1: Create the project

1. Launch Visual Studio 2005.
2. From the File menu, click New and then click Project. The New Project dialog appears.
3. Select the C# language.
4. From the SharePoint section, click the Web Part template.
5. Enter the name of the project in the Name text box, for example, MyWebParts.
6. Enter the location where you want the project to be created and accept the default solution name.
7. Click OK.

Figure 13.34 shows the Web Part project created using the Web Part template. Notice that the Web Part template automatically adds the required references, such as `Microsoft.SharePoint.dll` and `System.Xml`, and also adds the appropriate namespace directives to the code, such as `Microsoft.SharePoint.WebPartPages`. If you are not using the Web Part template to create your Web Part, you must manually add the references and namespace directives.

Step 2: Specify the build output path

All the files that comprise a Web Part, or for that matter a .NET application, are packaged into an assembly. Assemblies appear to the user as a single dynamic link library (.dll) or executable file (.exe). For your Web Part or assembly to run, it must be physically located in one of two specific locations on the server: the Global Assembly Cache (GAC), or the Web application's \bin directory. The GAC is used to store assemblies that need to be

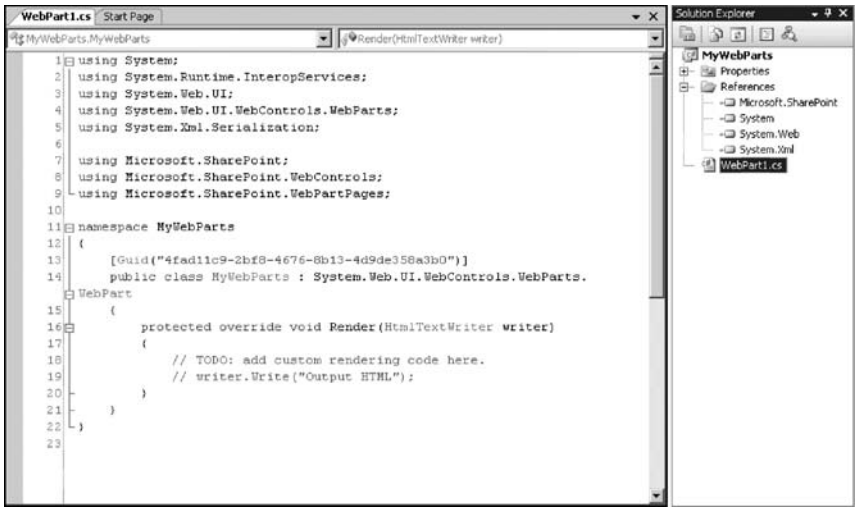


Figure 13.34
Web Part project created using template.

available to all applications on the server and is located in the \WINNT\ assembly directory. If you want to place your Web Part assembly in the GAC, you do not need to change the build output path of your project. Instead, manually copy the assembly to the GAC or use the Visual Studio GAC command line utility (gacutil.exe), located in the \bin directory of the Framework SDK. Code that runs in the \bin directory has the advantage of offering more granular security by being able to apply a lower level of code access security permissions. An additional benefit is the fact that it allows you to isolate a specific assembly to one or more select Web applications. A disadvantage of running code in the \bin directory is that in order to make your assembly available to all Web applications you need to deploy the Web Part to each application's \bin directory separately. Another disadvantage of using the \bin method is that inevitably your code will require more permissions than are permitted by the default code access security level, which means that these permissions must be elevated for your assembly to run. The recommended approach for elevating code access permissions is to create a trust policy file specific to your Web Part that grants the appropriate level. More information on code access security and how it applies to Windows SharePoint Services can be found on the Microsoft Web site. Deploying signed assemblies to the GAC means that they are automatically accessible to all Web applications. By default, assemblies in the GAC run with full trust, which means that you lose the granular security capabilities of the \bin method.

If you are developing your Web Part on the same computer that your SharePoint site is running, you can set the build output path to point to the Web application's \bin directory (e.g., \inetpub\wwwroot\wss\VirtualDirectories\80\bin), which is the same location from which the Web Part will run

on the server. If you are developing your Web Part remotely, you can simply deploy the Web Part assembly to the Web application's \bin directory after it is built.

Set the build output path as follows:

1. From the Project menu, click MyWebParts Properties. The property pages dialog appears.
2. Click the Build tab.
3. From the Outputs section, set the output path to the path of your Web application's bin directory. For example c:\inetpub\wwwroot\wss\VirtualDirectories\80\bin.
4. Close the property pages dialog.

Step 3: Add the AllowPartiallyTrustedCallers

If you are deploying your assembly to the \bin directory, you must include the `AllowPartiallyTrustedCallers` security attribute on your Web Part to allow it to run because the code used to execute your Web Part is granted only partial trust permissions. This attribute is added once for the entire assembly and is defined in the `AssemblyInfo` file.

Add the `AllowPartiallyTrustedCallers` attribute as follows:

1. From the Solution Explorer, expand the Properties folder, and open the `AssemblyInfo.cs` file.
2. Add the `AllowPartiallyTrustedCallers` by inserting the code shown in Figure 13.35. The `AllowPartiallyTrustedCallers` attributes are part of the `System.Security` assembly, and as such, this assembly needs to be referenced in the code file.
3. Save your changes to the `AssemblyInfo.cs` file

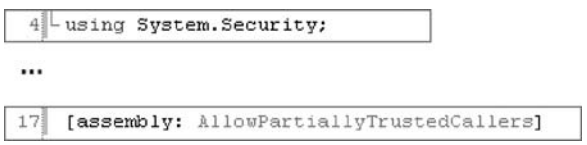


Figure 13.35 *Adding the AllowPartially TrustedCallers attribute.*

Step 4: Set the version number

In addition to classes and methods, assemblies also contain versioning information so that multiple versions of the same code can run side by side without conflicts. A Web Part page uses the version number specified in the `SafeControl` entry of the Web.config file to identify a Web Part. Therefore, you must ensure that when you recompile your Web Part, the `AssemblyInfo` contains the same version information specified in the Web.config file; otherwise, the Web Part will not run. The easiest way to do this is to manually control the version information stored in the AssemblyInfo.cs file. Setting the version to `1.0.*` means the project will automatically increment the version of your Web Part. Setting the version to 1.0.0.0 will ensure that the version numbering is static. If you want to release a new version, simply increment the version number (e.g., `1.0.0.1`) and update the Web.config file accordingly.

If you are using the Web Part template to create your Web Part, the version number is already static.

Set the version number as follows:

1. From the Solution Explorer, expand the Properties folder and open the AssemblyInfo.cs file.
2. Ensure that the assembly is set to manually increment. Figure 13.36 shows the appropriate C# code.
3. Save your changes to the AssemblyInfo.cs file.

```
[assembly: AssemblyVersion("1.0.0.0")]
```

Figure 13.36 *C#assembly version code.*

Step 5: Strongly name the assembly

A strong name consists of the assembly's identity, a public key, and a digital signature. For a Web Part to run in a SharePoint site, it needs to have a strong name, which basically informs SharePoint that the Web Part can be trusted. The strong name tool (sn.exe), available in the .NET Framework SDK, allows you to create a strong name key file; however, Visual Studio 2005 makes this process even easier for you by providing signing capabilities directly in the development environment. Once you have generated a strong name, either by using the tool or by using the signing tab in Visual Studio 2005, you add the strong name key file to your project. If you are using the Web Part template to create your Web Part, you will notice that a key pair file

already exists for your project, named Temporary.snk, so there is no need to follow the steps below.

The steps to strongly name the assembly are as follows:

1. From the Project menu, click MyWebParts Properties. The property pages dialog appears.
2. Click the Signing tab.
3. Check the box to Sign the Assembly.
4. From the drop down, select New to create a new strong name key file.
5. Enter the key file name and uncheck the setting to associate a password with the key file.
6. Click OK. A new strong name key file will appear in your solution explorer window. Move this file to the Properties folder of your project.
7. Save the project.

13.4.2 Coding the Web Parts

Now that we have the appropriate setup information defined, we can generate the code for our Web Parts.

Step 1: Name the assembly and the namespace

The first thing we need to do is specify the name of the assembly and the namespace that we want to use for our project. When naming your assembly, keep in mind that an assembly can house many Web Parts. In our example, the assembly will house two Web Parts, named WebPart1 and WebPart2. Consequently, it makes sense to apply a more generic name to the assembly, and it is good practice to provide a descriptive name that applies to all the classes in the assembly—for example, "BookWebParts". For illustrative purposes, our example uses the assembly name MyAssembly. While this name is not particularly descriptive, it does help illustrate the places where the assembly name appears during development. It is very easy to get confused between the assembly and the namespace when developing Web Parts. When naming your namespace, keep in mind that Web Parts within the same assembly can have different namespaces. For example, the SharePoint assembly is represented by the file Microsoft.SharePoint.dll and contains many classes that derive from many namespaces, such as `Microsoft.SharePoint.WebControls`, and `Microsoft.SharePoint.Administration`.

The following steps show how to rename your assembly and namespace using Visual C#:

1. Open your C# Web Part project in Visual Studio 2005.
2. From the Project menu, click MyWebParts Properties. The Property Pages dialog appears.
3. Click the Application tab.
4. In the Assembly field, enter the name of your assembly (for example, MyAssembly).
5. In the default namespace field, enter the name of your namespace (for example, MyNamespace).
6. Click the Assembly Information button.
7. Update the Title and Product fields to reflect the new assembly name.

Open the Web Part class file, WebPart1.cs, and update the namespace section of the code. Line 11 of Figure 13.37 shows the change that needs to occur.

Figure 13.37
WebPart1 C# code.

```
1  using System;
2  using System.Runtime.InteropServices;
3  using System.Web.UI;
4  using System.Web.UI.WebControls.WebParts;
5  using System.Xml.Serialization;
6
7  using Microsoft.SharePoint;
8  using Microsoft.SharePoint.WebControls;
9  using Microsoft.SharePoint.WebPartPages;
10
11 namespace MyNamespace
12 {
13     [Guid("4fad11c9-2bf8-4676-8b13-4d9de358a3b0")]
14     public class WebPart1 : System.Web.UI.WebControls.WebParts.WebPart
15     {
16         protected override void Render(HtmlTextWriter writer)
17         {
18             writer.Write("<h1>This is my first Web Part</h1>");
19         }
20     }
21 }
22
```

Step 2: Rename the Web Part

For the purposes of our example, we will keep the default Web Part name applied by the Web Part template: `WebPart1`. However, if you wish to change

the Web Part name, all you need to do is replace all occurrences of the text "WebPart1" with the new name. This can easily be achieved using the Find and Replace functionality in Visual Studio 2005, with the search scope set to current project.

Step 3: Insert the Web Part Code

Now that we have the environment configured correctly, we can add our Web Part code. The Web Part code is very basic and overrides the render method to display some HTML in the browser. The HTML code is a single line of text with the `H1` style applied. Line 18 of Figure 13.37 shows the required code; replace the contents of the render method with this code to display some sample text.

Step 4: Build the assembly

We are now ready to compile the assembly. This creates the assembly in the location specified in the output build path (\inetpub\wwwroot\wss\VirtualDirectories\80\bin).

1. Save your project to make sure you have the latest updates.
2. From the Build menu, click Build MyWebParts.

Step 5: Extract the public key

Once the assembly is compiled, you can extract the public key token from the strong name key file using the strong name tool. This value is needed in order to register your Web Part as a safe control for the site.

The steps to extract the public key token are as follows:

1. Open up a command prompt.
2. Navigate to the \bin directory of the .NET Framework SDK (Figure 13.38).

```
cd \Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin
```

Figure 13.38 *Changing directory to .NET Framework SDK.*

3. Extract the public key token by executing the command listed in Figure 13.39.

Figure 13.39
Extracting the public key token.

```
sn.exe -T c:\inetpub\wwwroot\wss\VirtualDirectories\80\bin\MyAssembly.dll
```

4. Copy the token from the command prompt window and paste it into Notepad for future use. An example of what a token looks like is: `9f4da00116c38ec5`.

13.4.3 Creating the Second Web Part

We'll create a second Web Part that performs the same function but displays different text.

The steps to create the second Web Part are as follows:

1. From the Visual Studio Solution Explorer, copy the Web Part class file: WebPart1.cs.
2. From the Solution Explorer, right-click the project name and click Paste.
3. Rename the newly created file as WebPart2.cs.
4. Open the newly created Web Part class file and replace the code to reflect the code shown in Figure 13.40.

Figure 13.40
WebPart2 C# code.

```
using System;
using System.Runtime.InteropServices;
using System.Web.UI;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Serialization;

using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;
using Microsoft.SharePoint.WebPartPages;

namespace MyNamespace
{
    [Guid("4fad11c9-2bf8-4676-8b13-4d9de358a3b0")]
    public class WebPart2 : System.Web.UI.WebControls.WebParts.WebPart
    {
        protected override void Render(HtmlTextWriter writer)
        {
            writer.Write("<h1>This is my second Web Part</h1>");
        }
    }
}
```

13.4.4 Deploying the Web Part

At last, we are close to seeing our newly created Web Parts in action. The inclusion of Features and the solution framework allows us to automate many of the manual steps involved in deploying a Web Part. However, at the end of the day, it is beneficial to understand the basic steps involved in manually deploying Web Parts. For this next step we will focus on the manual aspects of Web Part deployment; later in the chapter we will discuss how you can deploy Web Parts by leveraging Features and the solution framework.

Step 1: Register the Web Part control as a safe control

Since the Web Parts share a common assembly and namespace, only one entry in the Web.config is needed to register the Web Parts as safe controls. Figure 13.41 shows the entry to add to the `SafeControls` block of the Web.config file. The Web.config file is located in the root directory of the Web application (e.g., \inetpub\wwwroot\wss\VirtualDirectories\80\). For the Web Parts to run successfully, the assembly name and namespace name must be consistent between the Web Part class file, the Web Part definition file, the Web.config file, and the project settings with Visual Studio.

```
<SafeControl Assembly="MyAssembly, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=9f4da00116c38ec5" Namespace="MyNamespace" TypeName="*" Safe="True" />
```

Figure 13.41 *Registering the assembly as a safe control.*

Step 2: Add the Web Parts to the Web Part gallery

The Web Part definition file defines the properties associated with the Web Part, such as its height and width and whether the title bar is displayed. Windows SharePoint Services 3.0 and SharePoint Server 2007 accept two types of Web Part definition files: .dwp files and .Webpart files. The .dwp files represent Web Parts from Windows SharePoint Services 2.0 and SharePoint Portal Server 2003, and the .Webpart files represent the newer Windows SharePoint Services 3.0 and SharePoint Server 2007 Web Parts. Before you can use a Web Part on a Web Part page it must first be loaded into a Web Part gallery. In WSS 3.0 and SharePoint Server 2007 the Web Part gallery is merely a special document library used to store Web Parts. When you upload or import a Web Part into this library, the Web Part becomes available for use in SharePoint. For the purposes of this example, we will upload the Web Parts into the Web Part gallery at the root of the site collection. A convenient feature of Windows SharePoint Services 3.0 and SharePoint Server 2007 is that it has the capability to scan the contents of the safe controls section of

the Web.config file for new Web Parts and automatically creates the .Webpart file when adding new Web Parts to the Web Part gallery.

1. Using your browser, navigate to the home page of the root of the site collection.
2. From the Site Actions menu, click Site Settings and then click Modify All Site Settings.
3. From the Galleries list, click Web Parts. The Web Part gallery page appears.
4. From the Web Part gallery menu, click New.
5. The New Web Parts page appears.
6. Scroll to the bottom of the page and select the two new custom Web Parts, MyAssembly.WebPart1 and MyAssembly.WebPart2 (Figure 13.42).
7. At the top of the page, click the Populate button.

Figure 13.42 *Populate the Web Part gallery with new Web Parts.*

Step 3: Add the Web Parts to the Web Part Page

1. Using your browser, navigate to the home page of a team site in the site collection.
2. From the Site Actions menu, click Edit Page.
3. From the left Web Part zone, click Add a Web Part. The Add Web Parts Web page dialog appears.
4. Scroll down to the Miscellaneous section and select WebPart1 and WebPart2 (Figure 13.43).
5. Click the Add button.
6. In the upper right corner of the page, click the Exit edit mode link.

The Web Parts should appear on the Web Part page as shown in Figure 13.44. The Web Parts are very basic, displaying a simple line of text; how-

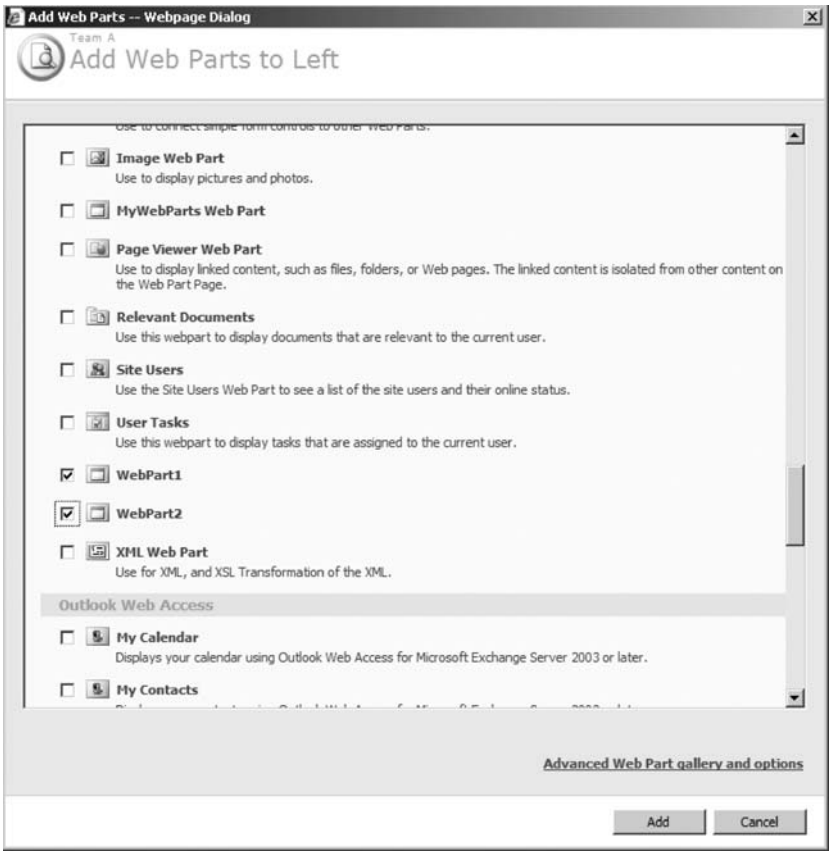


Figure 13.43
Adding the Web Parts to the page.

Figure 13.44
Web Parts added to page.

ever, when developing more complex Web Parts, the underlying development steps that you need to follow to create and deploy the Web Part remain the same. Now that we understand the basics of Web Part development, let's move on to building slightly more complex Web Parts.

13.5 Coding Web Parts

When working with Web Parts, it helps to understand how a Web Part is actually rendered. The ASP.NET 2.0 Web Part framework, and in particular the `SPWebPartManager` control, is at the heart of SharePoint's rendering process. The `SPWebPartManager` control overrides the standard ASP.NET 2.0 `WebPartManager` control and allows data to be stored in the SharePoint content database instead of the ASP.NET services database. When a request is made for a Web Part page, SharePoint retrieves the page source. If the code has not been customized using for example, SharePoint Designer 2007, it is considered an uncustomized (ghosted) page, and the source is initially retrieved directly from the SharePoint directory on the Web server. If the page has been customized, the source is retrieved from the content database.

13.5.1 Web Part Rendering Process

The rendering process for an individual Web Part can be broken down by examining the sequence of ASP.NET events executed throughout the process (Figure 13.45). When a Web Part page is requested, the `OnInit` event is triggered. The `OnInit` event launches the initialization phase for the Web Part. During the initialization phase, the Web Part framework retrieves the property values associated with the Web Part from the database, such as the title of the Web Part and its height and width. If your custom Web Part needs to perform any additional initialization steps, you can override the `OnInit` event. If this is the first time the Web Part is loaded, it proceeds to the `OnLoad` event, which sets the Web Part properties with the values retrieved from the initialization phase. If the Web Part has been loaded previously, the `LoadViewState` method is invoked to load the state information previously serialized into the Web Part, after which the `Onload` event proceeds and updates the properties. You can leverage the `OnLoad` event to retrieve information from various data sources, such as a database, that you want to display in the Web Part. If your Web Part uses any ASP.NET 2.0 controls, the `EnsureChildControls` and `CreateChildControls` events are launched to ensure that all the appropriate server controls are initialized and created in the control tree. Controls added by overriding the `CreateChildControls` method are automatically rendered by the framework and do not require further processing in the subsequent rendering events.

Once the Web Part has finished creating the required server controls and has handled user-generated events, it is ready to render the output to the browser. The rendering phase commences with the `OnPreRender` event. Developers can use the `OnPreRender` event to change Web Part properties before the output is rendered to the browser. On completion of the `OnPreRender` event, the view state information of the Web Part is serialized and saved to a hidden variable on the Web page for future load requests. At this stage of the rendering process, the output can actually be rendered to the browser. Two rendering methods, or events, are responsible for producing the final output: the `Render` event, and the `RenderContents` event. The `Render` event is responsible for rendering the entire Web Part, including the chrome, whereas the `RenderContents` event is only responsible for rendering the contents of the Web Part.

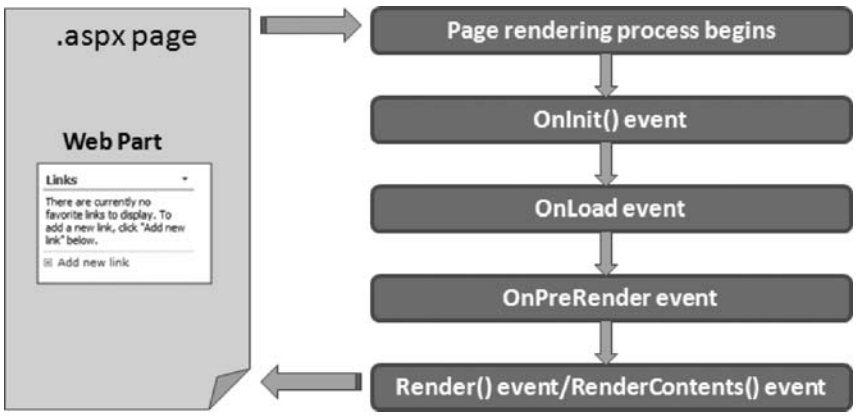


Figure 13.45 *Web Part rendering process.*

13.5.2 Web Part Properties and Methods

So far, we know that a Web Part is an ASP.NET control that inherits its properties and methods from the ASP.NET 2.0 `WebPart` base class. The properties associated with a Web Part determine the appearance and behavior of the Web Part, and each property typically has a get and set accessor method defined. The get accessor enables you to retrieve the value of a property, and the set accessor enables you to set the value of a property. Figure 13.46 shows sample C# code to define a custom Web Part property. If you have experience developing Web Parts for Windows SharePoint Services 2.0 and SharePoint Portal Server 2003, you will notice some similarities in the process of defining custom properties; you will also notice some differences in the attribute terminology. For example, the `WebBrowsable` attribute determines whether the Web Part property appears as a property within the Web Part tool pane; the same attribute was named `Browsable` in the previous version of SharePoint.

Figure 13.46
Defining a property using C#.

```
using System;
using System.Runtime.InteropServices;
using System.Web.UI;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Serialization;

using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;
using Microsoft.SharePoint.WebPartPages;

namespace MyNamespace
{
    [Guid("4fad11c9-2bf8-4676-8b13-4d9de358a3b0")]
    public class WebPart1 : System.Web.UI.WebControls.WebParts.WebPart
    {
        private string wpintro;

        [WebBrowsable(true)]
        [Personalizable(PersonalizationScope.Shared)]
        [WebDisplayName("Web Part Intro")]
        [WebDescription("Text to display at the top of your Web Part")]
        public string WPIntro
        {
            get
            {
                return wpintro;
            }
            set
            {
                wpintro = value;
            }
        }

        protected override void Render(HtmlTextWriter writer)
        {
            writer.Write("<h1>" + WPIntro + "</h1>");
        }
    }
}

```

When you use the `System.Web.UI.WebControls.WebParts.WebPart` class to create a Web Part, the Web Part automatically inherits all the base-level properties and methods from that class. If you examine the `WebPart` class using the Visual Studio object browser, you can see a list of the available properties (Figure 13.47).

The default settings for the Web Part properties are defined when the Web Part is loaded onto the page. If you want to control the settings for your Web Part using the object model, your changes should be applied using the `OnLoad` event. Figure 13.48 shows sample C# code used to set the Title and Description properties of the Web Part.

In addition to properties, Web Parts have a number of methods defined that provide functionality to the Web Part. For example, the base `WebPart` class

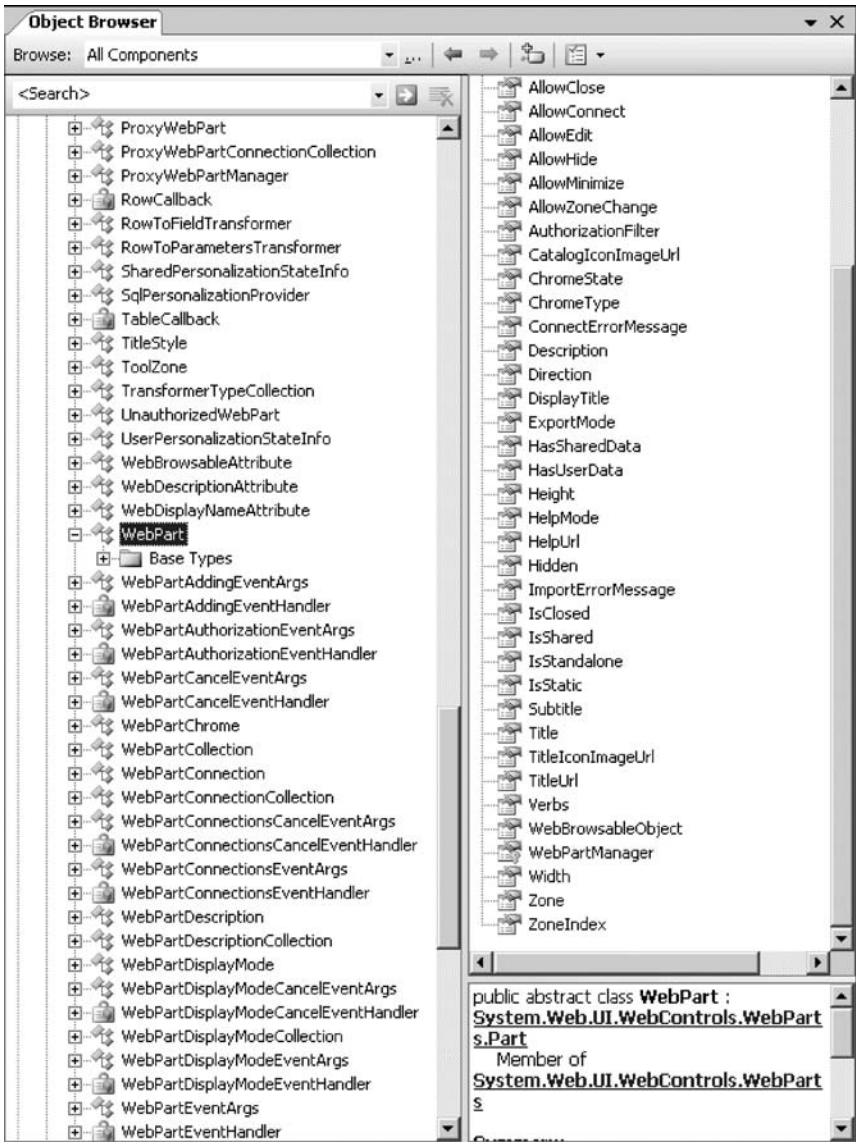


Figure 13.47 *WebPart class properties viewed using object browser.*

includes an `OnDeleting` method that is invoked when a Web Part control is deleted from a Web Part page. You can override this method to trigger some specific events when someone tries to delete the Web Part from the page. If you delve deeper into the inheritance model, you will find many more methods that you can leverage in your code. One such method that we have briefly touched upon is the `RenderContents` method of the `WebControl` base class. The `RenderContents` method is called by the Web Part framework to render the HTML for the body of the Web Part to the client. When building your

Web Parts, you will most likely override the `RenderContents` method when rendering custom HTML for the Web Part. More specifically, the `HtmlTextWriter` class provided by the `RenderWebPart` method allows you to create any HTML output you desire for your Web Part. Figure 13.49 lists some sample C# code that shows how to create a very basic Web Part that overrides the `RenderContents` method. Figure 13.50 shows the resultant output of this code. Lines 14 through 19 show how to override the `RenderContents` method using the `HtmlTextWriter` object named writer. The Write method of the `HtmlTextWriter` class, shown in line 18, writes a string to the HTML text stream. In this case, the string we are providing includes some special HTML tags, some text, and the string output of the `DateTime` ASP.NET function. Any HTML-embedded tags are rendered in the browser as HTML elements. If

Figure 13.48
Overriding the OnLoad event.

```
protected override void OnLoad(EventArgs e)
{
    this.Title = "My New Title";
    this.Description = "Setting Web Part Properties";
}
```

you want the HTML-embedded tags to render as text, or to ensure that any input properties are rendered as text, use `SPEncodeMethod.HtmlEncode`.

Figure 13.49
Implementing the RenderContents method.

```
1  using System;
2  using System.Runtime.InteropServices;
3  using System.Web.UI;
4  using System.Web.UI.WebControls.WebParts;
5  using System.Xml.Serialization;
6
7  using Microsoft.SharePoint;
8  using Microsoft.SharePoint.WebControls;
9  using Microsoft.SharePoint.WebPartPages;
10
11 namespace MyNamespace
12 {
13     [Guid("4fad11c9-2bf8-4676-8b13-4d9de358a3b0")]
14     public class BasicWebPart : System.Web.UI.WebControls.WebParts.WebPart
15     {
16         protected override void RenderContents(HtmlTextWriter writer)
17         {
18             writer.Write("<BR><B>Today's Date and time is: </B>" +
DateTime.Now.ToLongDateString());
19         }
20     }
21 }
22
```

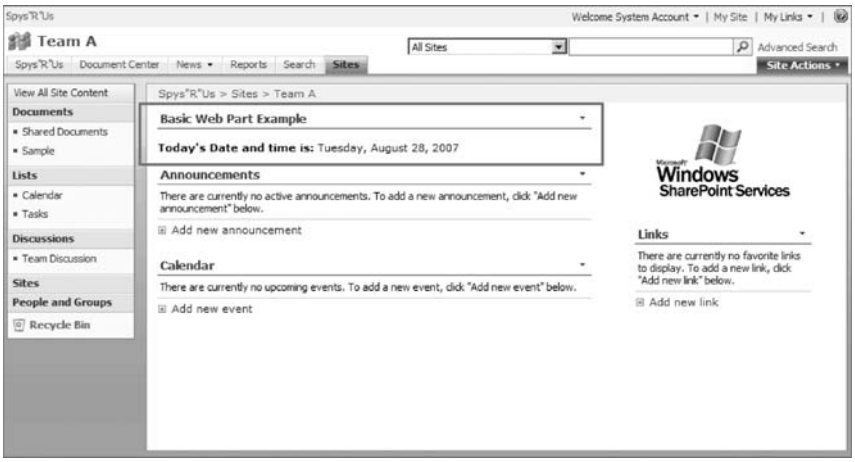


Figure 13.50 *Basic Web Part rendered in browser.*

13.5.3 Implementing Child Controls

When developing Web Parts using ASP.NET 2.0, you can create a user control (.ascx file) and place it into a Web Part zone, thereby converting the control into a Web Part. ASP.NET 2.0 automatically handles the conversion by wrapping the control in a `GenericWebPart` wrapper class at runtime. Unfortunately, this technique is not supported for WSS 3.0 and SharePoint Server 2007 Web Parts, and we are once again left to generate our Web Part interface through code. Note, there are a number of workarounds to this restriction available. Some involve creating a Web Part container that hosts the user controls; others rely on a separate virtual directory to host the user controls. The viability of such workarounds greatly depends on your production deployment and security policies.

From a user interface perspective, the typical sequence of events when developing a Web Part involves first creating the ASP.NET controls that you need in code, then adding these controls to the collection by overriding the `CreateChildControls` method of the `WebControls` base class. Figure 13.51 shows sample C# code that creates a Web Part with a `Calendar` control and a `Label` control. The basic premise of the Web Part is that when the user selects a date on the `Calendar` control, the `Label` control is populated with the selected date in the format of "Day, Month DD, YYYY"—for example, Monday, April 07, 2008. When the Web Part initially loads, we want both the `Calendar` control and the `Label` control to reflect the current date. As you can see from the sample code, the aforementioned functionality is easily achieved. Lines 17 and 18 declare the variables used for the ASP.NET 2.0 controls. The `Calendar` control comes equipped with an event, named `SelectionChanged`, which handles when a user selects a date on the control. When this event fires, we want to invoke a custom handler that updates the

text property of the Label control. Lines 20 through 23 represent the custom handler `mycalendar_SelectionChanged`.

The `CreateChildControls` method, represented by lines 25 through 34, is responsible for instantiating the controls and capturing the appropriate events. Lines 27 and 28 create a new instance of the `Calendar` control and set the currently selected date equal to today's date. Lines 29 and 30 create a new instance of the `Label` control and set the display text equal to today's date. Lines 31 and 32 add the controls to the controls collection. Finally, line 33 invokes the custom event handler, `mycalendar_SelectionChanged`, if the selection changes on the `Calendar` control, such as when the user selects a date. That is all the code that is required. Notice that our code does not override either the `Render` method or the `RenderContent` method; so how are our controls rendered? Remember, controls added by overriding the

Figure 13.51 *Creating Child Controls in a Web part (C#).*

```
1  using System;
2  using System.Runtime.InteropServices;
3  using System.Web.UI;
4  using System.Web.UI.WebControls;
5  using System.Web.UI.WebControls.WebParts;
6  using System.Xml.Serialization;
7
8  using Microsoft.SharePoint;
9  using Microsoft.SharePoint.WebControls;
10 using Microsoft.SharePoint.WebPartPages;
11
12 namespace MyNamespace
13 {
14     [Guid("4fad11c9-2bf8-4676-8b13-4d9de358a3b0")]
15     public class CalendarWP : System.Web.UI.WebControls.WebParts.WebPart
16     {
17         Calendar mycalendar;
18         Label myLabel;
19
20         public void mycalendar_SelectionChanged(object sender, EventArgs e)
21         {
22             myLabel.Text = mycalendar.SelectedDate.ToLongDateString();
23         }
24
25         protected override void CreateChildControls()
26         {
27             mycalendar = new Calendar();
28             mycalendar.SelectedDate = mycalendar.TodaysDate;
29             myLabel = new Label();
30             myLabel.Text = mycalendar.TodaysDate.ToLongDateString();
31             Controls.Add(myLabel);
32             Controls.Add(mycalendar);
33             mycalendar.SelectionChanged += new EventHandler
   (mycalendar_SelectionChanged);
34         }
35     }
36 }
37
```

`CreateChildControls` method are *automatically* rendered by the framework

and therefore do not require further processing in the subsequent rendering events. Figure 13.52 shows the Web Part displayed in a SharePoint site.

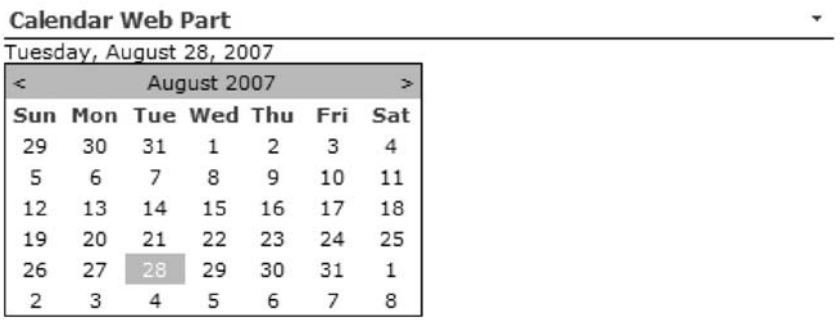


Figure 13.52 *Calendar Web part.*

13.5.4 Customizing the Web Part Menu

Each Web Part that you create automatically inherits a default Web Part chrome and menu that depending on the property settings of the Web Part, includes commands such as Minimize, Close, Delete, and Modify Shared Web Part. In Web Part programming terms, these menu items are known as `Verbs`. The ASP.NET 2.0 `WebPart` class has a `Verbs` property, of type `WebPartVerbCollection`, that predictably contains the collection of menu commands associated with the Web Part. You can add your own custom menu options to your Web Part by overriding the `Verbs` property and adding additional `Verbs` to the collection. Customization capabilities include the ability to add your own custom menu items, remove existing menu items, or even change the state of existing menu items.

In our example, two menu items have been added to the Web Part menu: Toggle Day Header and Toggle Grid Lines. When you click on the Toggle Day Header menu option, the day header that appears at the top of the `Calendar` control will appear or disappear, depending on the previous setting. If you click on the Toggle Grid Lines menu option, the grid lines will appear or disappear from the `Calendar` control, depending on the previous setting. Figure 13.53 shows the results of clicking the Toggle Grid Lines menu option when the grid lines were previously hidden; notice that the `Calendar` control now displays the grid lines.

The code required to implement the menus is very basic and simply involves creating the appropriate event handlers, retrieving the current `Verb` collection, instantiating the new menu items, and adding the new menu items to the `Verb` collection. Figure 13.54 shows sample C# code required to implement the custom Web Part menu. Each of the new menu items triggers an event upon selection. Therefore, a custom event handler is required for each menu item to handle the event when it is triggered. Lines 21 through 29

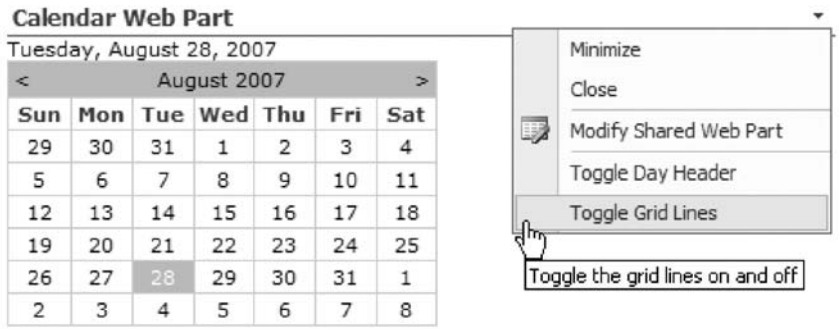


Figure 13.53 *Custom Web Part menu.*

define the event handlers for the menu options. The Toggle Day Header event handler (`verb1handler`) sets the value of the `Calendar` control's `ShowDayHeader` property to the opposite of its current value. For example, if the `ShowDayHeader` property is set to `true`, the `verb1handler` event resets the value to `false`, which in turn hides the day header from the top of the `Calendar` control. The Toggle Grid Lines event handler (`verb2handler`) sets the value of the `Calendar` control's `ShowGridLines` property to the opposite of its current value. For example, if the `ShowGridLines` property is set to `false`, the `verb2handler` event will reset the value to `true`, which in turn displays the grid in the Calendar control (Figure 13.53). Lines 31 through 46 represent the `WebPart Verbs` property that the Web Part overrides in order to add the new menu items.

13.5.5 Tool Panes and Editor Parts

If you refer back to Figure 13.46, notice that in addition to the get and set accessors that control the value of the property, there are attributes associated with the actual property itself, such as `WebBrowsableAttribute` and `PersonalizationScope` attribute. These attributes control how the Web Part property appears through the browser interface when in design mode. When you modify a Web Part in design mode using the browser interface, the Web Part tool pane appears to the right of the Web Part page (Figure 13.55). The Web Part tool pane allows you to modify the properties associated with the currently selected Web Part. The tool pane is divided into various sections, known as editor parts, determined by the Web Part itself. For example, the site image Web Part has an images editor part, an appearance editor part, a layout editor part and an advanced editor part. The editor part is analogous to the tool part in the previous version of SharePoint.

When defining custom Web Part properties using the method shown in Figure 13.46, the tool pane automatically generates the appropriate controls in the tool pane interface. The type of control generated is determined by the

Figure 13.54
Creating a Custom Web Part Menu

```
21          public void verb1handler(object sender, WebPartEventArgs e)
22          {
23              mycalendar.ShowDayHeader = !(mycalendar.ShowDayHeader);
24          }
25
26          public void verb2handler(object sender, WebPartEventArgs e)
27          {
28              mycalendar.ShowGridLines = !(mycalendar.ShowGridLines);
29          }
30
31          public override WebPartVerbCollection Verbs
32          {
33              get
34              {
35                  List<WebPartVerb> wpverbs = new List<WebPartVerb>();
36                  WebPartVerb verb1 = new WebPartVerb(this.ID + "_dayheader",
    new WebPartEventHandler(verb1handler));
37                  verb1.Description = "Toggle the Day Header on and off";
38                  verb1.Text="Toggle Day Header";
39                  wpverbs.Add(verb1);
40
41                  WebPartVerb verb2 = new WebPartVerb(this.ID + "_togglegrid",
    new WebPartEventHandler(verb2handler));
42                  verb2.Description = "Toggle the grid lines on and off";
43                  verb2.Text = "Toggle Grid Lines";
44                  wpverbs.Add(verb2);
45                  WebPartVerbCollection verbcoll = new WebPartVerbCollection
    (base.Verbs, wpverbs);
46                  return verbcoll;
47              }
48          }
49
```

property type. For example, if the Web Part property was of type `Boolean`, a `Check Box` control would be generated, and if the Web Part property was of type `String` then a `Text Box` control would be generated. Figure 13.56 shows the text box control added to represent the Web Part intro property. Properties added in this manner are assigned by default to the miscellaneous editor part.

There may be times when you would like more control over how your custom properties are presented through the tool pane—for example, you might like to apply special formatting. In this case, you would create a custom editor part that would group your custom Web Part properties into a separate section within the tool pane. Creating editor parts provides you with more control as to how your Web Parts are ultimately configured. Creating a custom editor part involves creating a new class that inherits from the `EditorPart` class and associating your Web Part with this new class. You associate a Web Part with a custom editor part by overriding the `CreateEditorParts` method of the Web Part.

To demonstrate the creation of a new editor part, we will modify our previous example by replacing the menu item functionality with two custom Boolean properties that represent the current setting of our `ShowDayHeader` and `ShowGridLines` properties.

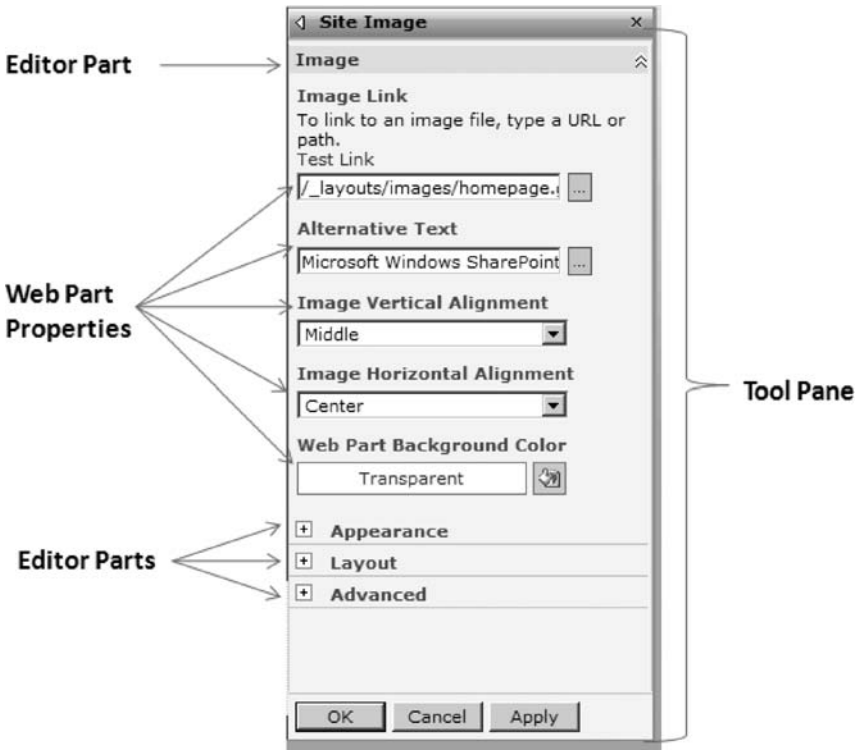


Figure 13.55 *Web Part tool pane.*

Figure 13.56 *Custom Web Part property added to tool pane.*

Step 1: Creating custom properties

The first step is to create our custom properties. Earlier in the chapter, we saw that the creation of a custom property involved creating get and set accessors

for the property. In addition to the accessors, we need to define the attributes that control how the Web Part property appears in design mode. The `WebBrowsableAttribute` determines whether the Web Part property appears in the tool pane. If it is set to `false`, then the custom property will not appear in the tool pane. The `WebDisplayName` attribute determines the text that appears for the custom property in the tool pane. If this attribute is not set, then the actual custom property name appears in the tool pane. The `WebDescription` attribute determines the content of the tool tip that appears when you hover the mouse pointer over the custom property in the tool pane. The `Personalizable` attribute represents that personalization setting for the Web Part property. Possible settings include `PersonalizationScope.Shared` and `PersonalizationScope.User`. `PersonalizationScope.Shared` indicates that changes to the Web Part property are shared by all users of the Web Part instance. `PersonalizationScope`.User indicates that changes to the Web Part property are specific to the individual user's instance of the Web Part. When the `Personalizable` attribute is set to `PersonalizationScope.User`, the Web Part properties appear in the tool pane when the user chooses to personalize the Web Part. If the `Personalizable` attribute is set to `PersonalizationScope.Shared`, the Web Part properties do not appear in the tool pane when the user chooses to personalize the Web Part. Figure 13.57 shows the C# code used to create our two custom properties. Lines 18 and 19 declare the private variables used in the get and set accessors later in code and also set the default values for these variables. Lines 20 through 28 define the `MyShowHeader` custom property. The fact that the `WebBrowsableAttribute` is set to false may seem strange at first, since you would expect that if you want to browse the attribute in your custom editor part, it would require a setting of true. However, setting `WebBrowsableAttribute` to false merely prevents the custom property from appearing in the default assigned editor part (i.e., miscellaneous editor part). This is an advisable practice, since it avoids confusing the user by having the custom property accessible from two places, and also avoids the developer having to synchronize the changes between the editor parts. Since we want the calendar Web Part to initially display with the day header enabled, we set the default value for our custom property to true. The `Personalizable` attribute is set to `PersonalizationScope.User`, which means that the custom property will appear in the design mode of both the shared view tool pane and the personal view tool pane; individual users will be able to personalize the settings of the custom property for their instance of the Web Part. The `WebDisplayName` attribute is set to Show Day Header, which appears in the tool pane and, since the property is of type Boolean, the Web Part infrastructure by default renders it as a check box. The `WebDescription` attribute defines the tool tip associated with the property. Lines 26 and 27 define the get and set accessors for the property. The get accessor allows you to retrieve the current value of the custom property, and the set accessor allows you to set its value. Lines 29 through 37 similarly define the `MyShowGridLines` custom property. Lines 49 through 53 shows the code added to the `RenderContents`

method that applies the changes made to the custom properties to the `Calendar` control. The code is very straightforward; it basically sets the `Calendar` control's Show Day Header field to the value stored in the `MyShowDayHeader` custom property and sets the `Calendar` control's Show Grid Lines field to the value stored in the `MyShowGridLines` custom property.

From a development and testing perspective, it is sometimes useful to set the `WebBrowsableAttribute` to true to test the functionality of your custom properties before you implement the custom editor part. Once you are happy that the custom properties are behaving as expected within your Web Part, you can then reset `WebBrowsableAttribute` to `false`. If you build the solution at this stage with the `WebBrowsableAttribute` set to `true`, the custom properties appear in the tool pane under the miscellaneous editor part, and you can check and uncheck the property values accordingly (Figure 13.58). Opening and closing the browser will show you that the values persist in the tool pane.

Step 2: Creating the custom editor part

Now that we have the appropriate functionality applied to our Web Part, all that remains is to create the custom editor part. The creation of the custom editor part is separated into two tasks. First, we need to create the class that we will use for our custom editor part, and then we must override the `CreateEditorParts` method in the main Web Part class. Figure 13.59 shows the code involved in creating the `EditorPart` class that hosts our custom properties. Lines 16 through 28 instantiates and creates the two checkbox controls required for the editor part. Lines 30 through 36 override the `SyncChanges` method; the `SyncChanges` method sets the initial values for the properties in the editor part. Lines 38 through 44 override the `ApplyChanges` method; the `ApplyChanges` method is self-explanatory in that it applies the changes triggered from clicking the Apply or OK buttons in the Web Part tool pane.

Figure 13.60 shows the code required to override the `CreateEditorParts` method. This method should be inserted before the `RenderContents` method of the sample code listed in Figure 13.57. The `CreateEditorParts` method retrieves the collection of editor parts associated with the Web Part, and adds an additional editor part titled Calendar Settings to the collection. Figure 13.61 shows the resultant output.

13.6 Connecting Web Parts

Web Parts themselves are excellent vehicles for providing users with relevant information, but their true potential lies in their ability to communicate with each other. A Web Part connection typically has a provider and a consumer.

Figure 13.57
Creating custom properties using C#.

```
14    public class CalendarWP : System.Web.UI.WebControls.WebParts.WebPart
15    {
16        Calendar mycalendar;
17        Label myLabel;
18        private bool myshowdayheader = true;
19        private bool myshowgridlines = false;
20        [WebBrowsable(false),
21        Personalizable(PersonalizationScope.User),
22        WebDisplayName("Show Day Header"),
23        WebDescription("Check to display day header on the calendar")]
24        public bool MyShowDayHeader
25        {
26            get{ return myshowdayheader;}
27            set{ myshowdayheader = value;}
28        }
29        [WebBrowsable(false),
30        Personalizable(PersonalizationScope.User),
31        WebDisplayName("Show Grid Lines"),
32        WebDescription("Check to display grid lines in calendar")]
33        public bool MyShowGridLines
34        {
35            get{ return myshowgridlines;}
36            set{ myshowgridlines = value;}
37        }
38
39        protected override void CreateChildControls()
40        {
41            mycalendar = new Calendar();
42            mycalendar.SelectedDate = mycalendar.TodaysDate;
43            myLabel = new Label();
44            myLabel.Text = mycalendar.TodaysDate.ToLongDateString();
45            Controls.Add(myLabel);
46            Controls.Add(mycalendar);
47        }
48
49        protected override void RenderContents(HtmlTextWriter writer)
50        {
51            mycalendar.ShowDayHeader = MyShowDayHeader;
52            mycalendar.ShowGridLines = MyShowGridLines;
53            base.RenderContents(writer);
54        }
55    }
56 }
```

The provider Web Part initiates the communication by providing some form of information that is of interest to a consumer Web Part. For example, let's suppose we have a simple form Web Part connected to a Shared Documents Web Part on the home page of a team site. The form Web Part has the role of the provider for the connection, so that when the user enters a name in the form, the text is sent to the consumer Web Part, which in this scenario is the Shared Documents Web Part. In the role of the consumer, the Shared Documents Web Part then consumes the name, and filters the Shared Documents document library to display only those documents modified by the user that match that name (Figure 13.62).

Figure 13.58 *Custom properties with WebBrowsable-Attribute set to true.*

Figure 13.59 *Creating Editor Part Class (C#).*

```
using System;
using System.Runtime.InteropServices;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Serialization;

using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;
using Microsoft.SharePoint.WebPartPages;

namespace MyNamespace
{
    public class CalEditorPart:EditorPart
    {
        CheckBox mycheckA;
        CheckBox mycheckB;

        protected override void CreateChildControls()
        {
            mycheckA = new CheckBox();
            mycheckB = new CheckBox();
            mycheckA.Text = "Show Day Header";
            mycheckB.Text = "Show Grid Lines";
            Controls.Add(mycheckA);
            Controls.Add(new LiteralControl("<br/>"));
            Controls.Add(mycheckB);
        }

        public override void SyncChanges()
        {
            this.EnsureChildControls();
            CalendarWP myCalWP = (CalendarWP)WebPartToEdit;
            mycheckA.Checked = myCalWP.MyShowDayHeader;
            mycheckB.Checked = myCalWP.MyShowGridLines;
        }

        public override bool ApplyChanges()
        {
            this.EnsureChildControls();
            CalendarWP myCalWP = (CalendarWP)WebPartToEdit;
            myCalWP.MyShowDayHeader = mycheckA.Checked;
            myCalWP.MyShowGridLines = mycheckB.Checked;
            return true;
        }
    }
}
```

```
50          public override EditorPartCollection CreateEditorParts()
51          {
52              List<EditorPart> myeditorParts = new List<EditorPart>(1);
53              CalEditorPart calPart = new CalEditorPart();
54              calPart.ID = this.ID + "CalEditorPart";
55              calPart.Title = "Calendar Settings";
56              myeditorParts.Add(calPart);
57              EditorPartCollection parts = new EditorPartCollection
    (base.CreateEditorParts(), myeditorParts);
58              return parts;
59          }
```

Figure 13.60 *Overriding the CreateEditorParts method (C#).*

Figure 13.61 *Custom Editor Part.*

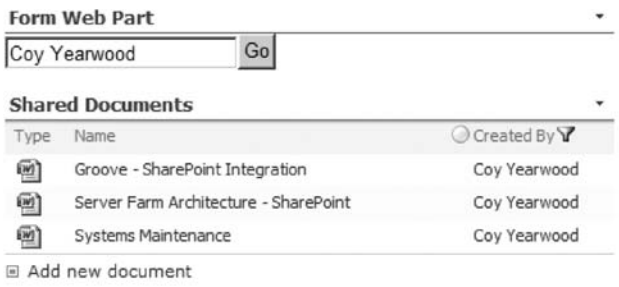


Figure 13.62 *Form Web Part filtering shared documents Web Part.*

When a Web Part is created, it does not automatically support connections; to provide this functionality, the Web Part itself has to implement a number of interfaces, which we'll cover shortly. You can determine whether an existing Web Part supports connections by navigating to the Web Part

Figure 13.63 *SendUrl IWebPartField provider Web Part.*

```
16    public class SendUrl: System.Web.UI.WebControls.WebParts.WebPart, IWebPartField
17    {
18        TextBox txtUrl;
19        Button btnSendUrl;
20        private string seturl=string.Empty;
21
22        [WebBrowsable(false),
23        Personalizable(PersonalizationScope.Shared),
24        WebDisplayName("Send URL"),
25        WebDescription("URL to send")]
26        public string SetUrl
27        {
28            get { return seturl; }
29            set { seturl = value; }
30        }
31
32        public void btnSendUrl_Click(object sender, EventArgs e)
33        {
34            SetUrl = txtUrl.Text;
35        }
36
37        protected override void CreateChildControls()
38        {
39            txtUrl = new TextBox();
40            btnSendUrl = new Button();
41            btnSendUrl.Text = "Send URL";
42            Controls.Add(txtUrl);
43            Controls.Add(btnSendUrl);
44            btnSendUrl.Click += new EventHandler(btnSendUrl_Click);
45        }
46
47        [ConnectionProvider("Send URL",AllowsMultipleConnections=true)]
48        public IWebPartField ConnectionInterface()
49        {
50            return this;
51        }
52
53        public void GetFieldValue(FieldCallback callback)
54        {
55            callback.Invoke(SetUrl);
56        }
57
58        public PropertyDescriptor Schema
59        {
60            get { return TypeDescriptor.GetProperties(this)["SetUrl"]; }
61        }
62    }
```

menu when in design mode. If the Connections menu is enabled, then the Web Part supports connections. The actual type of connection that a Web Part supports depends on a number of factors, including the type of Web Part it is (i.e., whether it's a server-side Web Part or a client-side Web Part), the location of the Web Part (i.e., whether it is on the same Web Part page as its connection Web Part), and whether it has exceeded the maximum number of connections permissible for the Web Part.

13.6.1 Connection Interfaces

The Web Part Framework that came with Windows SharePoint Services 2.0 and SharePoint Portal Server 2003 provided a standard set of connec-

tion interfaces that allowed Web Parts to communicate with one another at run time. The great thing about this was that the Web Parts could be developed independently of one another and could ultimately share information if they both supported the same interfaces. The interfaces provided with the previous version of SharePoint derived from the `Microsoft.SharePoint.WebPartPages.Communication` namespace and was implemented in a provider/consumer interface pair that was used to complete the communication sequence. For example, the `ICellProvider` interface allowed communication with other Web Parts that implemented the `ICellConsumer` interface. Keeping track of the supported connection interface pairs proved to be quite cumbersome, and the overall communication architecture was overly complex. With Windows SharePoint Services 3.0 and SharePoint Server 2007, Web Part connections still play a predominant role in Web Part development; however, the connection architecture has been revamped and now implements the standard ASP.NET 2.0 interface, thereby removing a lot of the complexity. The `Microsoft.SharePoint.WebPartPages.Communication` namespace is still supported, but it is only recommended for backward compatibility. When developing Web Part connections for Windows SharePoint Services 3.0 and SharePoint Server 2007, you should use the attributes and interfaces defined in the `System.Web.UI.WebControls.WebParts` namespace.

When developing connectable Web Parts, you can create a custom connection interface to communicate with other Web Parts, or you can leverage the standard interfaces provided by the `System.Web.UI.WebControls.WebParts` namespace. Creating your own custom connection interface has the advantage of being very easy to implement. However, the disadvantage is that your Web Part can only communicate with other Web Parts that implement the same custom interface; the Web Part will not be able to communicate with the default SharePoint Web Parts, such as the Shared Documents Web

Table 13.3 *Connection Interfaces*

Connection Interface	Description
IWebPartField	Defines a provider/consumer interface for connecting Web Parts using a single field of data.
IWebPartRow	Defines a provider/consumer interface for connecting Web Parts using a row of data.
IWebPartTable	Defines a provider/consumer interface for connecting Web Parts using a table of data.
IWebPartParameters	Defines a provider/consumer interface for connecting Web Parts by mapping a set of fields from one Web Part to another.

Part. Creating Web Parts that implement the standard connection interfaces are more complex to develop, however they have the added advantage of being able to communicate with both custom Web Parts and the default SharePoint Web Parts that implement the same interfaces. Table 13.3 lists the connection interfaces available for use when developing Web Part connections. A Web Part that implements any of these interfaces can communicate with other Web Parts that implement the same interface. For example, if you create a Web Part that implements the `IWebPartField` provider interface, then it can communicate with another Web Part on the page that implements the `IWebPartField` consumer interface.

Figure 13.63 shows an example of a Web Part, named `SendUrl`, which implements the `IWebPartField` provider connection interface to provide a URL to consuming Web Parts. It is up to the consuming Web Part how it intends to use the URL. For example, the Image Web Part can consume the URL provided by the `SendUrl` Web Part. Figure 13.64 shows the resulting output for the `SendUrl` Web Part; the Web Part has been connected to the image Web Part through the browser user interface. The `ConnectionProvider` and `ConnectionConsumer` attributes enable the connections offered by the Web Part. In our example, we use the `ConnectionProvider` attribute to enable the `SendUrl` Web Part to provide data to other consumer Web Parts. A Web Part can support both provider *and* consumer connections.

For Web Parts to communicate with one another, they must be able to understand one another, and generally this means that they should implement the same connection interface. The connection interface could be a custom interface or one of the standard interfaces provided by ASP.NET 2.0, such as the `IWebPartField` interface. However, there are times when you will want two Web Parts that implement different interfaces to be able to share information. In these situations, both ASP.NET 2.0 and SharePoint provide a number of extended connections, known as transformers. For example, ASP.NET 2.0 provides the `RowToFieldTransformer` and the `RowToParametersTransformer`, which allow a Web Part that has implemented the `IWebPartField` connection interface to communicate with a Web Part that has implemented the `IWebPartRow` connection interface, and vice versa. The `Microsoft.SharePoint.WebPartPages` namespace provides a transformer named `ITransformableFilterValues`, which allows you to create provider and consumer Web Parts for filtering purposes. For example, if you wanted to filter the contents of a list Web Part with a selection made in your custom provider Web Part, you could use the `ITransformableFilterValues` transformer to handle the filtering. The main point here is that even when you have Web Parts that implement different connection interfaces, there are ways to enable communication between them.

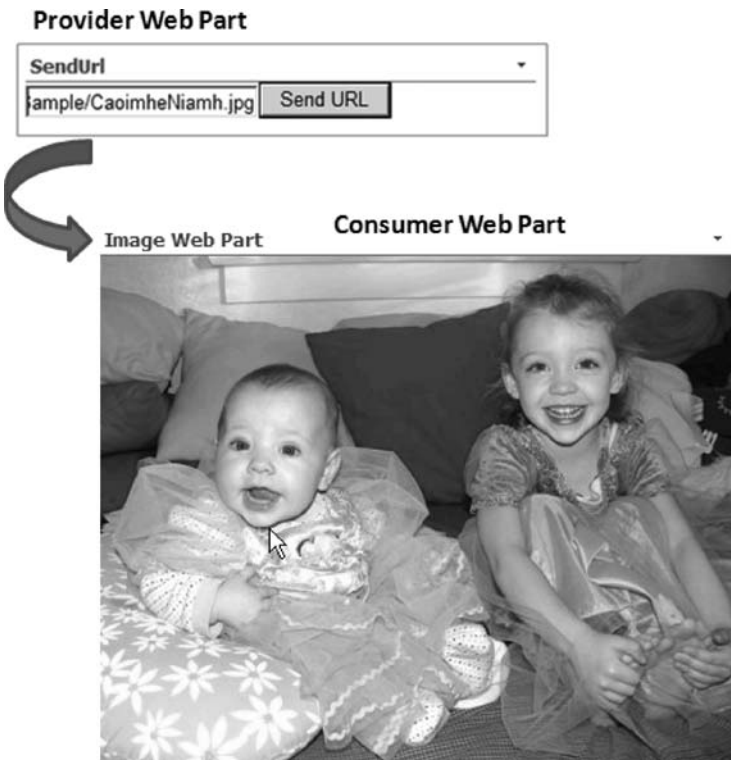


Figure 13.64
SendUrl provider Web Part.

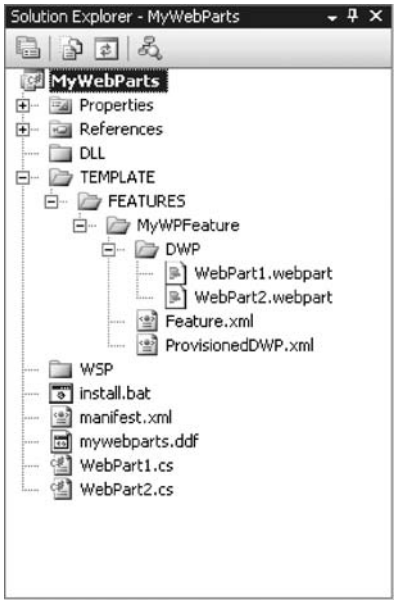


Figure 13.65
Solution folder hierarchy.

13.7 Debugging Web Parts

The method and techniques for debugging Windows SharePoint Services 3.0 and SharePoint Server 2007 Web Parts has not changed since the previous version of SharePoint. You are still able to step through your code line by line as the Web Part runs on the page. As before, debugging your Web Parts in this manner requires a number of preparatory steps before you can actually step through the code. At a high level, the steps include:

1. Ensure that the Web Part is registered as a safe control.
2. Ensure the appropriate level of trust, or code access permissions, is applied for the Web Part to run successfully.
3. The default trust level for SharePoint is `WSS_Minimal`, and `WSS_Minimal` does not allow debugging. Therefore, you will need to set the level of trust from `WSS_Minimal` to `WSS_Medium` or, alternately, create a specific policy file that applies the necessary permissions for debugging.
4. Configure the project debugging options in your development environment—for example, in Visual Studio 2005, enable ASP.NET debugging.
5. Turn debugging on by setting the debug attribute of the compilation element in the Web.config file to true.
6. Enable SharePoint to display the full error message by setting the mode attribute of the `customErrors` element to `Off`.
7. Allow stack traces to be displayed by setting the `CallStack` attribute of the `SafeMode` element to true.

It should be noted that when you are making the customizations to the Web.config file, you should first make a backup copy of the original so that you can revert back to it once you are finished debugging.

To debug your Web Parts directly from Visual Studio 2005, you can simply attach your project to the ASP.NET process (w3wp.exe); however using this method requires that you reattach to the process after you have stopped debugging. So if, for example, while debugging, you find an error that you want to correct, you need to terminate the process, modify and save the code, and then reattach the process to step through your changes once again. If you are developing Web Parts on a remote computer, you need to ensure that the server that hosts SharePoint is configured for debugging. In addition, you need to set the build path maps to the \bin directory of the SharePoint server, instead of the local machine.

13.8 Packaging and Deploying Web Parts

We have shown you how to develop a Web Part and manually deploy it to the site collection's Web Part gallery. The manual process can be convenient when working in a development environment, but when it comes to deploying your Web Parts in production, you should leverage the Windows SharePoint Services 3.0 solution framework. We introduced the solution framework earlier in the book, but let's provide a quick recap of its purpose. In essence, the solution framework provides a method for collating the various components that comprise a solution, packaging them together into a solution file, and deploying the solution in a consistent and efficient manner to the SharePoint Farm. The deployment process can be initiated using stsadm.exe or using the central administration browser interface via the operations solution management page. During deployment, the files contained in the solution package are examined and placed in their appropriate location on the server. If the solution package contains Web Parts, the Web Parts are placed in their appropriate location, either the \bin directory or the GAC, and a safe control entry is added to the Web application's Web.config file. It should be noted that the deployment process does not upload, or populate, the Web Part gallery with the new .Webpart files. To achieve this you need to include the .Webpart file in your solution package and use a feature to provision the .Webpart file into the appropriate Web Part gallery. Features are covered in detail in Chapter 16, Features; however, for completeness we will show you how to provision the .Webpart file when packaging your Web Parts.

There are three methods available for creating a solution file. The first involves using the command line tool makecab.exe, the second involves using a setup and deployment project template in Visual Studio 2005, and the third method involves using the Visual Studio 2005 Deploy Solution build menu option. Chapter 11, Introduction to SharePoint Development explains each method in detail. For the purposes of this chapter, we will focus on the first technique: building a solution file using makecab.exe.

13.8.1 Packaging the Web Part

When you are creating Web Parts in your development environment, you will inevitably go through various stages of testing before the final Web Part is ready for packaging. To test your Web Part, you will need to deploy the Web Part manually through the steps defined earlier in the chapter. Once you are happy that the Web Part runs successfully and is production-worthy, you can package your solution. The following section picks up from the development and manual deployment of our Web Parts—WebPart1 and WebPart2.

Step 1: Extracting the .webpart files

Earlier in the chapter we created two Web Parts, WebPart1 and WebPart2, and deployed them to the Web Part gallery at the root of the site collection. One of the first things that we will want to do is to export the .Webpart files associated with WebPart1 and WebPart2 so that we can include them in our solution file. The following steps show how to extract the .Webpart files from the Web Part gallery.

1. Using your browser, navigate to the Web Part gallery at the root of the site collection.
2. Locate the two Web Parts; they should be named WebPart1.Webpart and WebPart2.Webpart.
3. For each Web Part, click the Edit Properties icon. The Web Part properties page should appear.
4. From the menu bar, click Export and save the .Webpart files to your desktop.

13.8.2 Step 2: Package the solution

1. Launch Visual Studio 2005 and open the project that contains the code for WebPart1 and WebPart2.
2. Within the project, create the folder structure as shown in Figure 13.65.
3. Set the output build path to point to the .dll folder that you created in the previous step. We need to do this so that the makecab.exe tool can locate the assembly and package it into the .wsp file.
4. Import WebPart1.Webpart and WebPart2.Webpart into the TEMPLATE\FEATURES\MyWPFeature\DWP folder in the project folder hierarchy.
5. Navigate to the TEMPLATE\FEATURES\MyWPFeature folder and add two new text files. Rename them Feature.xml and ProvisionedDWP.xml.
6. Open Feature.xml and insert the code shown in Figure 13.66. Save the changes.
7. Open ProvisionedDWP.xml and insert the code shown in Figure 13.67. Save the changes.

Figure 13.66
MyWPFeature Feature.XML.

```
<Feature  Id="C3FC38A5-3E1C-4115-9FE0-1EAA073858B5"
          Title="Provision My Web Parts"
          Description="Feature adds the .webpart files to the gallery"
          AlwaysForceInstall="TRUE"
          Scope="Site"
          Hidden="true"
          xmlns="http://schemas.microsoft.com/sharepoint/" >
    <ElementManifests>
        <ElementManifest Location="ProvisionedDWP.xml"/>
    </ElementManifests>
</Feature>
```

Figure 13.67
Provisioned DWP.xml.

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
    <Module Name="AllMyWebParts" List="113" Url="_catalogs/wp" Path="DWP
" RootWebOnly="TRUE">
        <File Url="WebPart1.webpart" Type="GhostableInLibrary" />
        <File Url="WebPart2.webpart" Type="GhostableInLibrary" />
    </Module>
</Elements>
```

8. At the root of the project add three new text files and rename them as manifest.xml, myWebparts.ddf, and install.bat.

Figure 13.68
Manifest.xml.

```
<Solution SolutionId="65683FD5-DF27-4085-8F41-2EB6C2C8665D" xmlns="http:
//schemas.microsoft.com/sharepoint/">
    <FeatureManifests>
        <FeatureManifest Location="MyWPFeature\Feature.xml"/>
    </FeatureManifests>
    <Assemblies>
        <Assembly Location="MyAssembly.dll" DeploymentTarget=
"WebApplication" >
            <SafeControls>
                <SafeControl Assembly="MyAssembly, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=9f4da00116c38ec5" Namespace="MyNamespace
" TypeName="*" Safe="True" />
            </SafeControls>
        </Assembly>
    </Assemblies>
</Solution>
```

9. Open manifest.xml and insert the code shown in Figure 13.68. Save the changes.
10. Open myWebparts.ddf and insert the code shown in Figure 13.69. Save the changes. Note that the files used in the project should be referenced by stating both the source path and the destination path to the files.
11. Open install.bat and insert the code shown in Figure 13.70. Save the changes.

Figure 13.69
mywebparts.ddf.

```
;***Deploy Web Parts ***
.OPTION EXPLICIT
.set CabinetNameTemplate=MyWebParts.wsp
.set DiskDirectoryTemplate=CDROM
.set CompressionType=MSZIP
.set UniqueFiles="ON"
.set Cabinet=on
.set DiskDirectory1=WSP
manifest.xml
DLL\MyAssembly.dll
TEMPLATE\FEATURES\MyWPFeature\Feature.xml
MyWPFeature\Feature.xml
TEMPLATE\FEATURES\MyWPFeature\ProvisionedDWP.xml
MyWPFeature\ProvisionedDWP.xml
TEMPLATE\FEATURES\MyWPFeature\DWP\WebPart1.webpart
MyWPFeature\DWP\WebPart1.webpart
TEMPLATE\FEATURES\MyWPFeature\DWP\WebPart2.webpart
MyWPFeature\DWP\WebPart2.webpart
```

12. At this point your project should look very similar to that shown in Figure 13.65.
13. From the Project menu, click <project> Properties.
14. Navigate to the Build Events tab and enter the commands shown in Figure 13.71 in the Post-Build event command line.
15. Save and build the project.
16. Exit Visual Studio.

Figure 13.70
myWebparts.ddf

```
@SET TEMPLATEDIR="C:\Program Files\Common Files\Microsoft Shared\
web server extensions\12\TEMPLATE\"
xcopy /e /y TEMPLATE\* %TEMPLATEDIR%
makecab.exe /f mywebparts.ddf
stsadm.exe -o addsolution -filename WSP\MyWebParts.wsp

stsadm.exe -o activatefeature -name MyWPFeature -url http://devmoss
```

All that remains is to deploy the solution and activate the Web Part provisioning feature.

Step 3: Deploy the Solution

1. Navigate to the operations page in the SharePoint central administration Web interface of the farm in which you would like to deploy the solution.

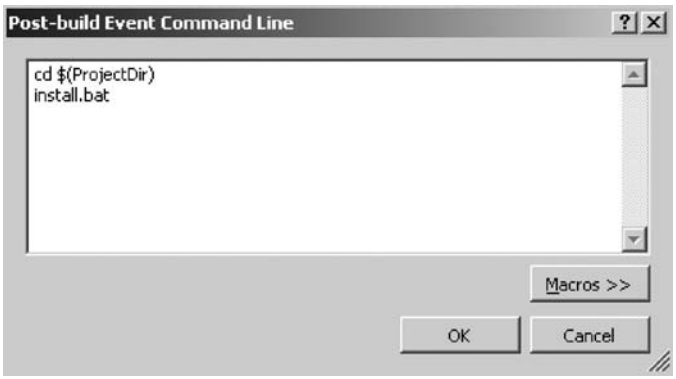


Figure 13.71 *Post build command.*

```
stsadm.exe -o activatefeature -name MyWPFeature -url http://devmoss
```

Figure 13.72 *Activating the Web Part provisioning feature.*

Figure 13.73 *Web Parts provisioned in the Web Part gallery.*

2. From the Global Configuration section, click Solution Management. The solution management page appears, and the myWebparts.wsp solution should be listed as an available solution in the farm.
3. Click the myWebparts.wsp link. The solution properties page appears.
4. From the solution properties page, click Deploy. The deploy solution page appears.
5. From the deploy solution page, choose to deploy the solution now, and select the Web application which you would like to deploy the Web Parts. You can also choose to deploy the solution to all Web applications within the farm. The solution management page shows the status of the deployment; once the solution is deployed successfully, the status will display "deployed."
6. Open the command prompt and enter the command shown in Figure 13.72 to activate the solution and provision the files.

When the solution is deployed, the file provisioning feature is installed in the Web application, but it is not activated. For the files to be provisioned, the feature must be activated, and since it is a hidden feature we cannot activate it from the Web interface. We could have added the commands to deploy the solution and activate the feature in our install.bat to streamline the process; to effectively demonstrate the process, we chose to instead perform these tasks manually. More information on features can be found in Chapter 16, Features.

Now all that remains is for us to see if the deployment was successful. There are a number of checks you can perform to verify that the solution was deployed successfully:

1. Check the Web.config file of the Web applications to which you deployed the solution for a Safe Control entry for the Web Part assembly file. (e.g. MyAssembly.dll)
2. Check the \bin directory of the Web application to see if the Web Part assembly is present.
3. Navigate to the Web Part gallery at the root of the site collection and check to make sure that the WebPart1.Webpart and WebPart2.Webpart files are present. Figure 13.73 shows the Web Parts successfully added to the Web Part gallery. Note that these files will only show up if the Web Part provisioning feature has been activated.

4. Import the Web Parts onto a page in a site.

Web Parts remain at the heart of SharePoint development. In this chapter, we really only touched the surface of the customization capabilities that Web Parts actually provide. SharePoint provides an extensible platform that integrates well with a Microsoft developer's tool set, allowing you to build and deploy your own custom Web Parts with relative ease. In addition, the product provides a set of feature-rich Web Parts out-of-the-box, which not only serve a real business need, such as aggregating data from multiple sources, but are also highly customizable.

This page intentionally left blank

14

Branding your SharePoint 2007 Sites

Both SharePoint Portal Server 2001 and SharePoint Portal Server 2003 left a lot to be desired when it came to the ease of applying corporate branding to sites. The introduction of master pages, combined with the enterprise Web content management Features of SharePoint Server 2007, helps alleviate many of the issues experienced with previous versions. In this chapter, we will examine some of the more common applications of branding, such as custom color schemes, banners, and navigation, and show how they are achieved in SharePoint Server 2007. The chapter is structured by first describing the various functional components that SharePoint Server 2007 offers to help apply a brand, such as master pages and page layouts. If you are already familiar with these concepts and just want to get to a branding exercise, you can skip to the end of the chapter, where we go through a couple of branding scenarios. It is also important to note that this chapter is really intended to familiarize you with the basics regarding customizing the look and feel of your site—more detailed aspects of branding are explained further in Chapter 15, Site Definitions and Templates, and Chapter 16, Features.

14.1 What is Branding?

One of the first requests that a developer or Web designer is likely to encounter when SharePoint is introduced to his or her company is, "Can I apply our company's look and feel to my SharePoint sites?" The *look and feel*, commonly referred to as branding, typically comprises customization of a site so that all pages rendered to the user are consistent in their presentation. For example, you could customize your site so that each page presented to the user contains the same header, footer, and navigational structure. Branding is important when deploying Web sites, as it not only provides a consistent look and feel across all the pages in a site but also reduces the learning curve from the end users' perspective. Once a user becomes familiar with the layout and usage of one page in your site, this knowledge becomes immediately transferable to other pages the user visits, increasing the usability of your site.

If a Web site is easy to use, there is a higher likelihood of a user returning to the site for information. Branding is also used to establish identity and ownership. For example, when you first install SharePoint Server 2007, the default branding is easily recognizable as Microsoft's branding (Figure 14.1). You have the default Microsoft color scheme, the default Microsoft navigational structure and the default Microsoft logo. Most companies will want to override the Microsoft identity with their own corporate identity, thereby making the site specifically their own. This factor is important, especially in an environment where users are collaborating and sharing data with each other. When a user visits a page that applies the corporate branding and adheres to all the corporate publishing standards they are much more likely to trust the source and provide information to that site.

Applications of branding are wide and varied, with the more common aesthetic applications relating to color and style, header/footer regions, navigation, and the structure of pages in the Web site. The remainder of this chapter focuses on these aspects of branding.

Figure 14.1 *Collaboration portal using default Microsoft branding.*

14.2 Branding Pain Points in SharePoint Portal Server 2003

SharePoint Portal Server 2001 had significant limitations when it came to applying custom color schemes, banners, layouts, and navigational structure. For example, to apply your corporate banner to the title area of the portal, it was necessary to customize some of the system dashboard template files. Customization of these system files took your SharePoint deployment into unsupported territory, and they were susceptible to being overwritten by ser-

vice packs and hot fixes. SharePoint Portal Server 2003 improved the customization capabilities of the SharePoint platform with the introduction of site definitions and custom templates, in addition to its tight integration with FrontPage 2003. A site definition, first introduced with SharePoint Team Services, is basically a set of files used when creating sites that provide information about the site's lists, views, Web Parts, pages, and navigational structure. Custom templates are created through the user interface or by using the SharePoint object model, and instead of residing on the file system, they are stored in the SharePoint content database. Both WSS v2.0 and SharePoint Portal Server 2003 support the use of multiple site definitions and templates, meaning that you can create sites that reflect the needs of the target audience by enabling the appropriate functionality up front through the use of custom lists, Web Parts, and custom pages. With respect to look and feel, site definitions provide the basic foundation for SharePoint Portal Server 2003 sites. Customization of a site definition often results in manipulation of the Collaborative Application Markup Language (CAML) code in the base XML files. The main XML file of a site definition, known as the ONET.XML, contains a number of significant elements that control various components of a SharePoint Portal Server 2003 site. For example, the ListTemplates element defines the lists included in the site definition, and the AllUsersWebPart element implements the Web Parts to appear on the page.

By now you are probably thinking that this doesn't sound so bad! Site definitions allow you to perform advanced customizations to SharePoint sites, and if you apply all your changes to *custom* site definitions and not the base system files, then you avoid your implementation being unsupported. However, WSS 2.0 and SharePoint Portal Server 2003 site definitions have significant disadvantages. If you examine the content of a SharePoint Portal Server 2003 ONET.XML file, you will see that it is poorly structured and that many of the site definitions share common element definitions. For example, site definitions that contain the same lists still require a separate copy of the list definition. In addition, working with site definitions requires a basic knowledge of CAML, which can prove to be quite challenging for even the most experienced programmer. Applying simple changes to a site definition, such as adding a new banner or footer, involves applying the same customization repeatedly to many files. For example, the AllUsersWebPart element in the ONET.XML file of a site definition can be used to implement a banner and footer on the default.aspx page. This element would run in conjunction with custom code inserted on the default.aspx page to render the custom banner Web Part. To apply the banner and footer to all pages, the default.aspx customization would then be repeated on every .aspx page in the site definition. Not only is this customization technique cumbersome, it is also difficult to keep track of *all* the changes made to your system. When a site has been instantiated from a custom site definition, it is difficult to apply changes to the site, and in fact, Microsoft recommends that you avoid apply-

ing customizations to site definitions once they have been used for site creation. A big factor for this recommendation is that the customization that you make could actually break existing sites, causing them to be irreparable.

If you only wanted to make a minor change to some header code on your .aspx pages and you knew that this change would not break your existing sites, you then faced the dreaded ghosting and unghosting issue. To recap, in SharePoint Portal Server 2003 the pages and list schema of a SharePoint site are read from the site definition files and cached at IIS process startup. The pages appear to be physically part of the site; however, when a page is requested, it is not in the actual database but on the file system and is pulled from the cache at run time. This concept is known as ghosting, and Microsoft uses ghosting for both pages and metadata in SharePoint Portal Server 2003. So, for example, if you examine the CAML code associated with a view definition, you will notice that there is a lot of XML generated to inform SharePoint how to render specific views within a Web Part page, such as allitems.aspx. Giving everybody a copy of the same XML would waste resources and affect the scalability of the system; therefore, ghosting tells SharePoint to use the page that is already available in the site definition. So you could have 10,000 sites, but as far as ASP.NET is concerned, it is only serving up one page as opposed to 10,000 pages. ASP.NET dynamically renders different content into that one page depending on the page request—but it is still only one page. From a performance perspective, ghosting significantly reduces the memory footprint and the I/O hit on SQL Server to serve these pages. Using FrontPage 2003 to customize a page in a SharePoint Portal Server 2003 site effectively unghosts the page, resulting in its entire contents being stored in the database. This means that each time the customized page is requested, instead of retrieving it directly from disk, it is retrieved from the database, which, in turn, reduces the site's overall performance. In addition to affecting the overall performance, any future updates made to the site definition files, such as a updating some header code on your .aspx pages, will not apply to unghosted pages.

SharePoint Server 2007 still has the concept of ghosted and unghosted pages (now more appropriately referred to as uncustomized and customized pages, respectively) however, the new architecture embraced by SharePoint Server 2007 and highlighted throughout this chapter means that it no longer has the adverse impact of its predecessor. For example, you now have the ability to reset a page to its site definition, thereby clearing the copy of the page stored in the database. This functionality is also known as reghosting, and is accessible through the browser (Figure 14.2) or through SharePoint Designer 2007 (Figure 14.3). Once a page is reset to its site definition it will resume the ability to receive updates made to the site definition files.

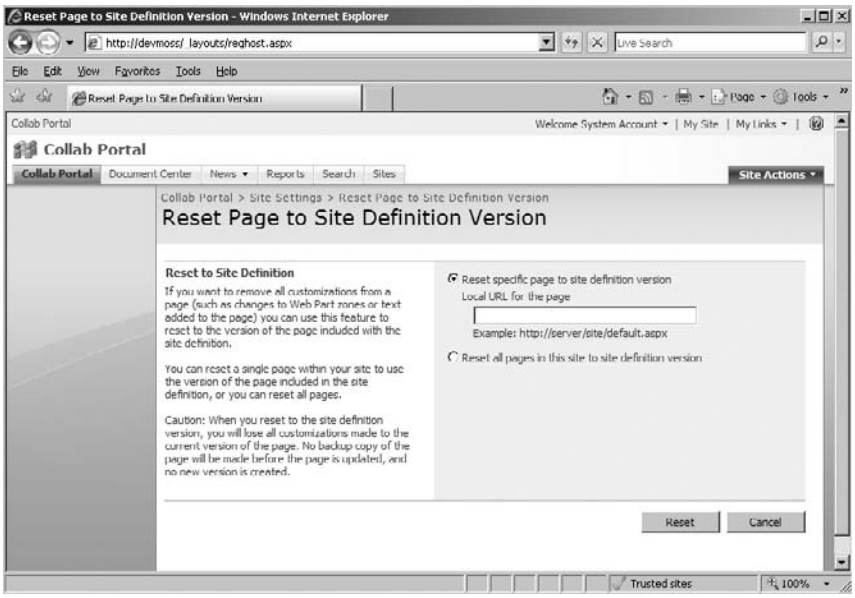


Figure 14.2
Resetting page to site definition version—browser UI.

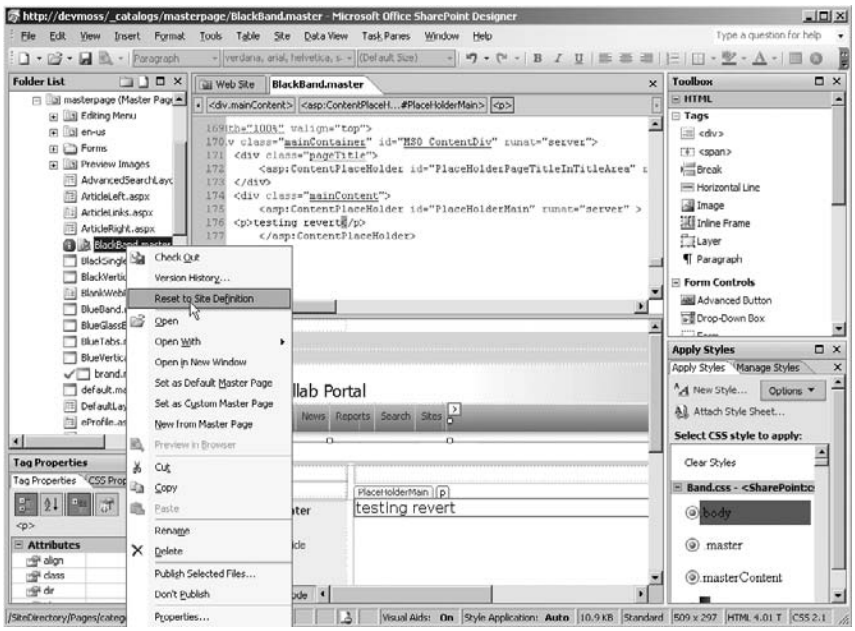


Figure 14.3
Resetting page to site definition version—SharePoint designer UI.

The Reset to site definition option can be found on the site settings page through the browser. When you customize a ghosted page using SharePoint Designer 2007, you receive a message indicating that saving your

changes will detach the page from its original site definition. If you proceed to save the changes then, as with the previous version of SharePoint, a copy of the page is stored in the content database and changes to the original site definition page no longer affect the customized page. In the context of SharePoint Designer 2007, you can reset a customized page to its uncustomized version using the Reset to site definition menu option (Figure 14.3). When resetting a customized page, SharePoint Designer replaces the content of the customized page with the original content from the site definition and also creates a backup of the customized page in the same folder. This allows you to keep a copy of your customizations without actually applying them to the original page.

14.3 SharePoint 2007 Enhancements

SharePoint Server 2007 presents a whole set of welcome additions to its architecture that drastically improve the customization story, particularly when it comes to branding your SharePoint sites. The adoption of master pages coupled with enterprise content management capabilities and tight integration with SharePoint Designer 2007 mean that branding SharePoint sites has never been easier. Let's take a look at the key players in this area and how they simplify the branding process.

14.3.1 Enterprise Content Management

As discussed in Chapter 1, 2007 Microsoft Office System and Chapter 5, Enterprise Content Management, the enterprise content management capabilities of SharePoint Server 2007 cover a broad spectrum of activities, from the authoring and publishing of documents to the creation, management, and deployment of Web content for internal and Internet-facing Web sites. The latter activity is especially important when it comes to branding, as it allows us to use the content management features of SharePoint Server 2007 to manage and deploy the appropriate master pages, page layouts, and style sheets to achieve our brand. Master pages, page layouts, and style sheets are simply files stored in document libraries at the root of your site collection, and as such they inherit all of the document management and content management capabilities that a regular document would have, such as per-item level permissions, two-stage delete, versioning, check-in/check-out, and the ability to create content types. For example, if you would like to make a change to a master page, you can simply check out the file, make the necessary modifications, save the changes, and check the file back into the system to get a preview of what the branding changes would look like in production. If you are happy with the changes, you can publish the master pages, which would in turn launch an approval process. When the changes are approved, the updated master page is published, thereby deploying the brand to your

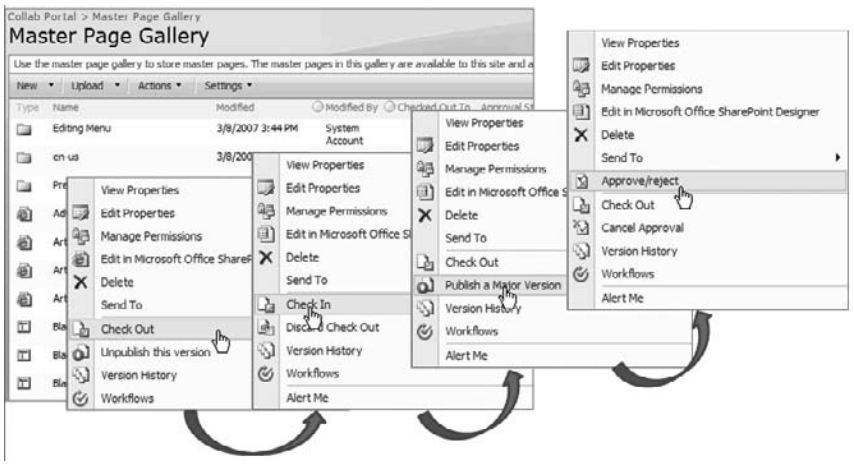


Figure 14.4
Enterprise content management options in master page gallery.

site. If you are not satisfied with the changes, you can simply roll back the file to the previous version. Figure 14.4 shows the enterprise content management menu options available to files stored in the master page gallery of a publishing site collection.

So what does all this really mean? All the files used to apply your customizations now adopt the flexibility and functionality provided by the enterprise content management capabilities of SharePoint Server 2007. This means that any custom master pages or page layouts that you deploy can be processed through a content management lifecycle before the changes are actually visible to your end users. It means that you can actually test the changes before they go live and, more importantly, you can roll back to a prior version should you need to quickly revert your changes.

Note that the publishing and approval stages of the content management lifecycle are only applicable to SharePoint Server 2007 site collections that have the Office SharePoint Server publishing infrastructure Feature activated. WSS 3.0 on its own is not equipped with the publishing Feature and as such does not inherit the publishing and approval capabilities. Chapter 16, Features, discusses the concept of Features in great detail, but for now you can see whether your site collection has the Office SharePoint Server publishing infrastructure Feature activated by navigating to the site collection features page (Figure 14.5); this page is accessible from the site settings page.

Activating the Office SharePoint Server publishing infrastructure Feature causes SharePoint to provision some specific libraries, content types, and master pages and enables publishing functionality for the site collection. For example, if you create a site collection based on the Document Center site template, the publishing infrastructure Feature is not active. Examining the master page gallery for this site collection shows only a small set of files,

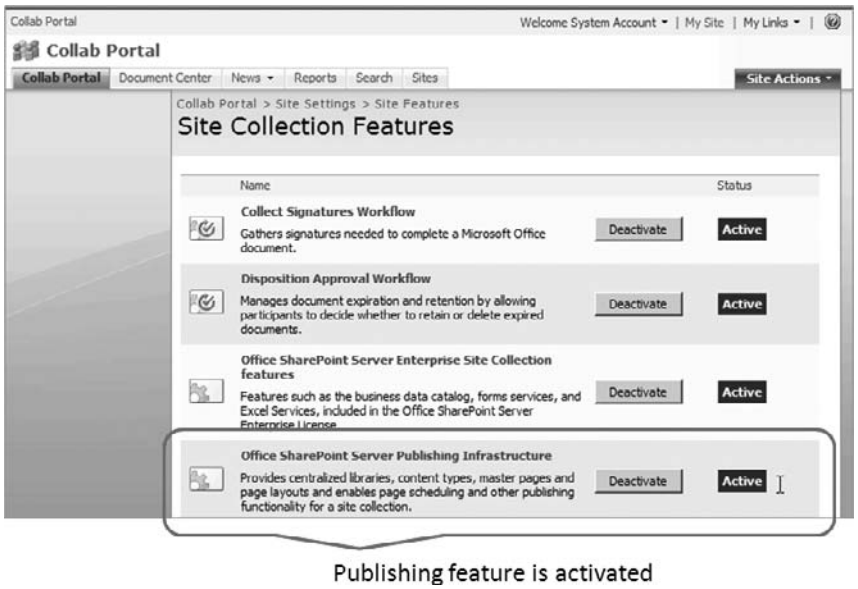


Figure 14.5 *Office SharePoint Server publishing infrastructure feature.*

including *one* master page named default.master. Activating the Feature populates the master page gallery with additional page layouts and master pages. Figure 14.6 shows a before and after picture of this scenario.

14.3.2 Master Pages

As discussed in previous chapters, master pages are an integral part of the ASP.NET 2.0 architecture embraced by SharePoint Server 2007. A master page represents the chrome, or brand, of a page, providing a separation between the presentation and the actual content of the page. The chrome typically consists of elements common to all pages in a site, such as the banner, navigation, search controls and login controls. Content pages inheriting a specific master page adopt the chrome of the master. All pages in SharePoint Server 2007 and WSS 3.0 inherit a master page. This is particularly good news for the ghosting scenario discussed earlier in the chapter. Although customized pages are stored in the content database and consequently no longer receive updates from the site definition, they can still pick up changes made to the master page because their page contains a reference to the master page and not the actual master page code.

So what makes a page a master page? Figure 14.7 shows a new ASP.NET 2.0 master page generated by SharePoint Designer 2007 and named test.master. The master page does not contain references to SharePoint at this point, but it gives you an idea of what comprises a master page. A master page has several characteristics that differentiate it from a regular ASP.NET page. It

Figure 14.6
Document Center with publishing infrastructure feature activated.

uses the .master file extension, indicating that the file is a master page. If you examine the master page gallery of a SharePoint publishing site collection you will see a number of files with the .master extension, such as BlackBand.master and default.master. A master page uses the new ASP.NET 2.0 Master directive to identify the page as a master page and defines content placeholder regions, using `ContentPlaceHolder` controls to effectively map out the specific areas on a page where content can be displayed. Pages that inherit from the master may only define content for the designated placeholder regions, thus ensuring a consistent look and feel to the pages throughout a site. Additionally a page may only reference placeholder controls defined in the inherited master page, otherwise the page fails to render.

The test.master code shown in Figure 14.7 includes some HTML style tags that basically tell the master to set the background color and font to a specific setting. Content pages that inherit test.master adopt the style settings defined in the master page, and subsequent updates made to test.master, such as changing the background color to green, are automatically reflected in the inheriting content pages. Figure 14.8 shows an example of a content page (hello_world.aspx) that inherits the test.master master page. The inheritance relationship is defined in the page directive by setting the masterpagefile attribute to the location of the master page, which in our example is set to "../_catalogs/masterpage/test.master". To render content on the page, all we need to do is insert an ASP.NET `Content` control and link it to the content

Figure 14.7 *ASP.NET 2.0 test.master master page.*

```
<%@ Master Language="C#" %>
<html dir="ltr">

<head runat="server">
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Test</title>
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
</head>

<body style="color: #FFFFFF; background-color: #800000; font-
family:Verdana;font-size:20pt;font-style:italic">

<form id="form1" runat="server">
    <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
    </asp:ContentPlaceHolder>
</form>

</body>

</html>
```

Figure 14.8 *Content page inheriting test.master master page.*

```
<%@ Page masterpagefile="../_catalogs/masterpage/test.master"
language="C#" title="Hello World" %>

<asp:Content ContentPlaceholderID="ContentPlaceHolder1"
runat="server">
    Hello World
</asp:Content>
```

placeholder defined in the master page. For example, in hello_world.aspx we have linked the `Content` control to the content placeholder named `ContentPlaceHolder1`, which is defined in the master page. If the master

page had named its content placeholder `EmerPageContentHolder`, the `ContentPlaceholderID` associated with the `Content` control would reference `EmerPageContentHolder`. Notice that hello_world.aspx does not contain any specific style references; the color scheme and font style are inherited directly from the master page, illustrating the separation of the page presentation from the page content.

14.3.3 SharePoint Master Pages

SharePoint Server 2007 and WSS 3.0 master pages reside in the master page gallery located in the _catalogs/masterpage folder of a site collection. The master page gallery link is accessible from the site settings page in the browser user interface. Any new master pages that you generate should be saved to this gallery. Master pages stored in the master page gallery appear in the drop-down list box on the master page settings page (Figure 14.9). WSS 3.0 ships with one master page (default.master) in the master page gallery, and SharePoint Server 2007 ships with additional master pages accessible by enabling the publishing infrastructure Feature mentioned earlier in this chapter. You can set the master page for the root level of a site collection and push the selected master page to all subsites; alternatively, you can set the master page at the individual site level. Note that once you add a new master page to the master page gallery, it cannot be deleted. Deletion attempts result in an error message stating: *This item cannot be deleted because it is still referenced by other pages*, and the error message appears even if the new master page is not referenced by other pages. This anomaly is documented in the Microsoft knowledge base under KB article 926812 and may be fixed by subsequent service packs. In the interim, the recommendation is to edit the content type properties of the master page you want to delete and set the page to hidden. Even though you do not have the ability to delete the master page, you can overwrite or rename the file.

The master page for a site, or site collection, can be changed using the master page settings page, accessible from the Look and Feel menu on the site settings page. Figure 14.9 shows the master page settings page; notice that there are two types of pages for which the master page can be set: site pages and system pages. Site pages refer to all custom application pages, such as the publishing pages in SharePoint Server 2007, and system pages refer to SharePoint application pages, such as the forms and view pages in a site. An easy way to familiarize yourself with which pages map to which master page setting is to set the site master and the system master to different master pages and then traverse your site to review which pages pick up which master page. The system master is often referred to as the *default master*—note, we are not talking about the default.master file, but rather a specific terminology for describing a type of master page. The site master is often referred to as the *custom master*.

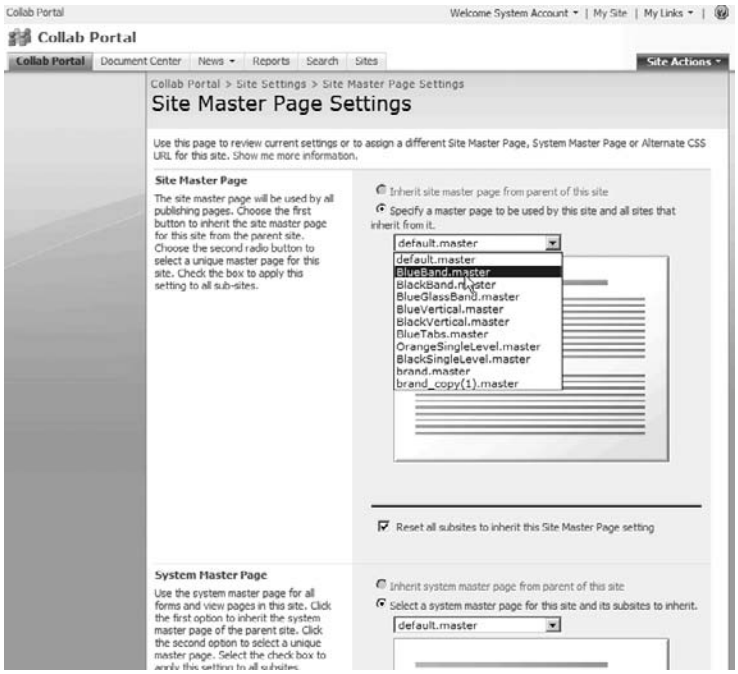


Figure 14.9
Master page settings.

SharePoint Designer 2007 also offers options for setting both the site master and the system master for a site. In fact, SharePoint Designer uses the custom master and default master terminology. Using SharePoint Designer 2007, if you right-mouse-click a master page in the _catalogs/masterpage folder, you will see two menu options: Set as Default Master Page, and Set as Custom Master Page (Figure 14.10). The options are made available depending on the role currently assumed by the master page. For example, if the selected master page is already operating as the default master but not operating as the custom master, the Set as Default Master Page option is grayed out. Setting master pages in this way only applies to the current site; there is no option to push the master page settings to child sites from within SharePoint Designer 2007.

When generating a SharePoint content page, one of the first things you need to do is reference the master page, you would like your page to inherit. You can directly reference the master page as shown in Figure 14.8; however, your content page will not reflect the master page settings specified through the browser user interface or through the SharePoint Designer 2007 menu. So how do we tell a content page to use the default master or custom master settings? WSS 3.0 provides several tokens that map to reference a master page. The tokens are separated into dynamic tokens and static tokens. The dynamic tokens include `~masterurl/default.master` and `~masterurl/custom.master`, and are used in conjunction with the `MasterPageFile`

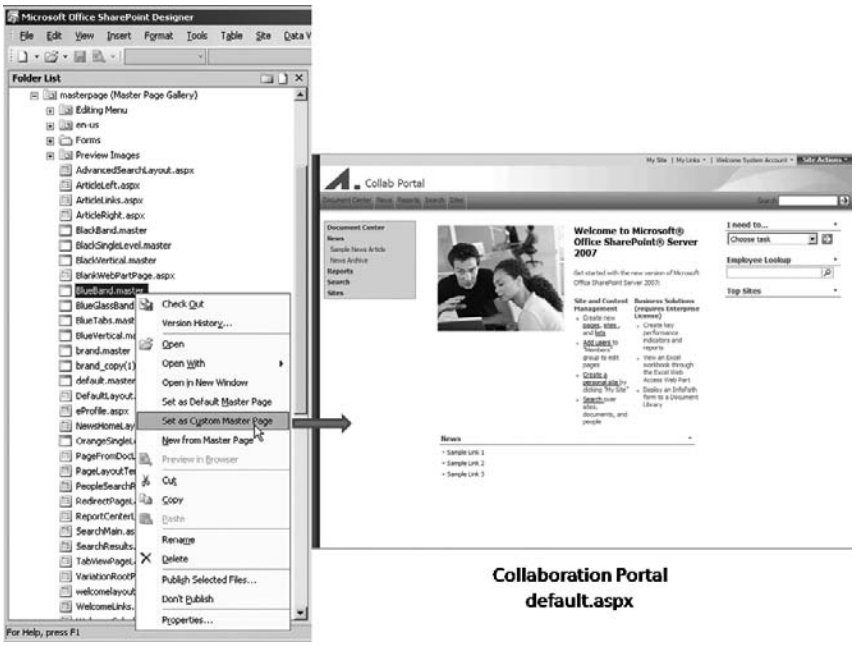


Figure 14.10
Setting Custom Master using SharePoint Designer 2007

attribute specified in your `@ Page` directive. If you would like your content page to inherit the custom master setting, you would simply reference the `~masterurl/custom.master` dynamic token in the `@ Page` directive. The dynamic tokens are replaced at runtime by the values of two specific properties of the current site. The `~masterurl/custom.master` token is replaced with the value of the `CustomMasterUrl` property, which defines the custom master, or site master setting for the site. The `~masterurl/default.master` token is replaced with the value of the `MasterUrl` property, which defines the default master, or system master for the site. Figure 14.11 shows examples of the `@ Page` directive settings for both the default master and the custom master.

```
<%@ Page language="C#" MasterPageFile="~masterurl/default.master"
<%@ Page language="C#" MasterPageFile="~masterurl/custom.master"
```

Figure 14.11
Page directives for settings the default master and the custom master.

Static tokens allow you to link your content page to a site relative or site collection relative master page. The static tokens include `~site/yourmasterpage.master` and `~sitecollection/yourmasterpage.master`, and are referred to as *static* because the master page setting does not change at runtime.

Note that new sites do not automatically inherit the `MasterUrl` and `CustomMasterUrl` properties of their parent site. However, should you need this functionality, there are several techniques to choose from, which will be discussed in Chapter 15, Site Definitions and Templates, and Chapter 16, Features.

Master pages created for SharePoint Server 2007 and WSS 3.0 *must* contain a specific set of placeholder controls for the default SharePoint pages to render correctly. For example, the master page shown in Figure 14.7 would result in a page error when applied to a default SharePoint page, because it does not contain the required placeholder regions referenced in files such as the default.aspx. The Microsoft MSDN Web site provides a base-level master page, known as a minimal master, containing all the necessary placeholder regions required to create a basic master page. You can use the minimal master as the template from which to create your own master pages and, by doing so, ensure that your master page honors all SharePoint pages. Figure 14.12 shows a subset of the HTML used to generate a minimal master, and Figure 14.13 shows a document center site with the minimal master applied. By examining the HTML, you can see that there are approximately 17 placeholder regions defined. Notice, too, that the rendered page has little in the way of actual branding. The reason for this is that the code supplied with the minimal master contains no tables, layouts, structuring, or style settings; it is simply a file containing the required components for a SharePoint page to render successfully. The only styles inherited by the minimal master originate from the default SharePoint style sheet referenced by the SharePoint `CssLink` control. If you are using the minimal master as the foundation for your master page, it is up to you to provide the additional HTML to meet your branding needs.

If you find that one of the default master pages that ships with SharePoint Server 2007 is similar in layout to the master page you would like to create, you can use it as the template from which to create your master pages. To do this you would simply use SharePoint Designer 2007 to create a new master page, open the master page that you would like to use as the template (taking care *not* to actually check out the file), and then copy and paste the code into your new master page. You are then free to customize the master page in any way you please, and once you save the file to the master page gallery, it can be used to brand your site. We'll step through the process of creating a master page using this method in the branding exercise at the end of this chapter. Note that if you check out, or customize an out-of-the-box master page, you effectively unghost, or customize, the master page file in that specific master page gallery. Consequently, the master page no longer receives updates applied to the original master page on the file system, and since the master page no longer receives updates, the content pages that reference the customized master page no longer receive the updates, regardless of whether they use dynamic tokens. You can reset a customized master page to its original version on the file system using

Figure 14.12 *HTML for minimal master.*

```
<html>
  <WebPartPages:SPWebPartManager runat="server"/>
  <SharePoint:RobotsMetaTag runat="server"/>
  <head runat="server">
  <title><asp:ContentPlaceHolder id="PlaceHolderPageTitle" runat="server" /></title>
      <Sharepoint:CssLink runat="server"/>
      <asp:ContentPlaceHolder id="PlaceHolderAdditionalPageHead" runat="server" />
  </head>
  <body onload="javascript: _spBodyOnLoadWrapper();">
    <form runat="server" onsubmit="return _spFormOnSubmitWrapper();">
      <wssuc:Welcome id="explitLogout" runat="server"/>
      <PublishingSiteAction:SiteActionMenu runat="server"/>
      <PublishingWebControls:AuthoringContainer id="authoringcontrols" runat="server">
        <PublishingConsole:Console runat="server" />
      </PublishingWebControls:AuthoringContainer>
      <asp:ContentPlaceHolder id="PlaceHolderSearchArea" runat="server"/>
      <asp:ContentPlaceHolder id="PlaceHolderMain" runat="server" />
      <asp:Panel Visible="false" runat="server">
          <asp:ContentPlaceHolder id="PlaceHolderTitleBreadcrumb" runat="server"/>
          <asp:ContentPlaceHolder id="PlaceHolderPageTitleInTitleArea"  runat="server"/>
          <asp:ContentPlaceHolder id="PlaceHolderLeftNavBar" runat="server"/>
          <asp:ContentPlaceHolder ID="PlaceHolderPageImage" runat="server"/>
          <asp:ContentPlaceHolder ID="PlaceHolderBodyLeftBorder" runat="server"/>
          <asp:ContentPlaceHolder ID="PlaceHolderNavSpacer" runat="server"/>
          <asp:ContentPlaceHolder ID="PlaceHolderTitleLeftBorder" runat="server"/>
          <asp:ContentPlaceHolder ID="PlaceHolderTitleAreaSeparator" runat="server"/>
          <asp:ContentPlaceHolder ID="PlaceHolderMiniConsole" runat="server"/>
          <asp:ContentPlaceHolder id="PlaceHolderCalendarNavigator" runat ="server" />
          <asp:ContentPlaceHolder id="PlaceHolderLeftActions" runat ="server"/>
          <asp:ContentPlaceHolder id="PlaceHolderPageDescription" runat ="server"/>
          <asp:ContentPlaceHolder id="PlaceHolderBodyAreaClass" runat ="server"/>
          <asp:ContentPlaceHolder id="PlaceHolderTitleAreaClass" runat ="server"/>
     </asp:Panel>
    </form>
  </body>
</html>
```

the Reset to site definition options available through the browser user interface and through SharePoint Designer 2007.

SharePoint Designer 2007 provides a reporting function that allows you to easily identify the customized pages within your site. You can set filters on the report to show a subset of data. For example, if you want an easy way to identify all the master pages that have been customized in your site so that you can reset them back to the file system version, you can set the filters on the report to show pages of type master where the customized value is equal to yes (Figure 14.14).

So far we have discussed the various components that form a master page and how these components are linked to content pages, but we haven't really mentioned how a page is rendered in the browser. SharePoint Server 2007 publishing pages are rendered differently than base WSS 3.0 content pages, as they introduce another layer of separation known as page layouts. We'll discuss page layouts in the next section, but for now let's focus on how a regular content page is rendered in SharePoint. Figure 14.15 provides an overview of the rendering process, and as you can see, it is quite straightforward. When a user requests a content page in SharePoint, ASP.NET examines the page directive of the content page and retrieves the master page

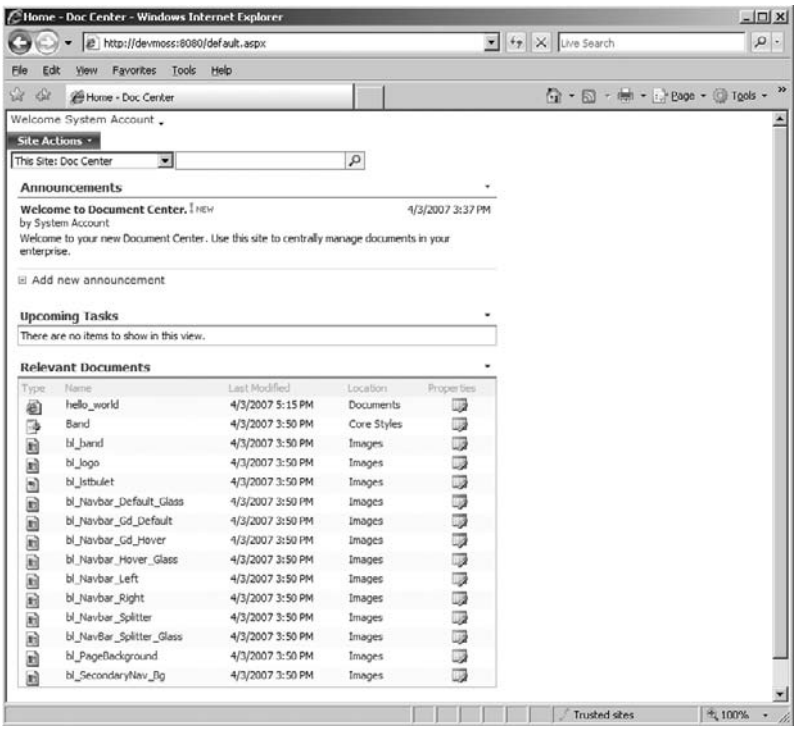


Figure 14.13 *Minimal master applied to a document center site.*

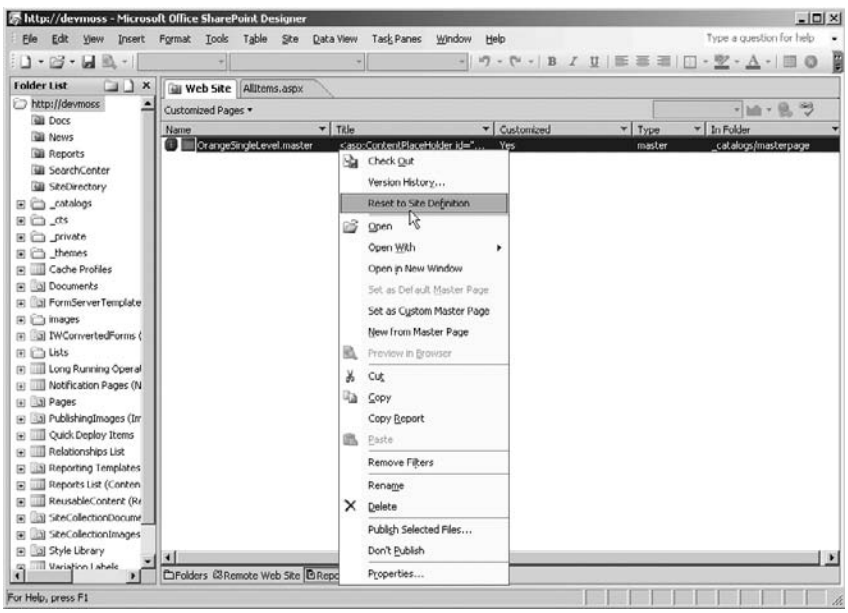


Figure 14.14 *Identifying customized master pages in SharePoint Designer 2007.*

referenced in the `MasterPageFile` attribute. The content in the content page is matched up with the appropriate `ContentPlaceHolder`, and the `Content` controls are rendered in the browser along with the chrome defined in the master page.

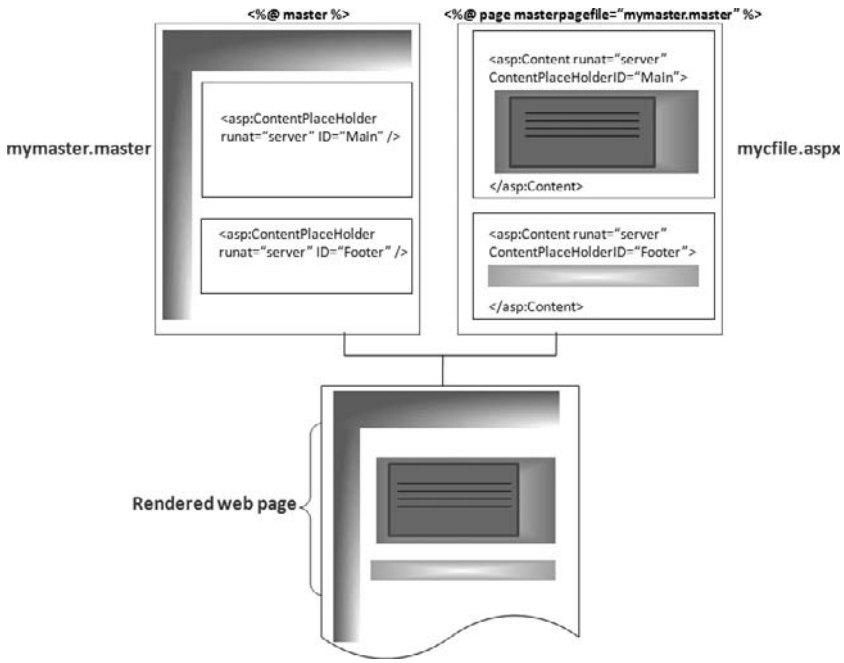


Figure 14.15
WSS 3.0 Page Rendering

14.3.4 Application.master—the Nemesis of Branding!

Changing the site master and system master settings to reference your own master pages gets you well on the way to a fully branded site; however, it will not get you all the way there. There are other content pages in SharePoint that actually contain static references to master pages that cannot officially be modified. For example, most of the files in the _layouts directory reference ~/_layouts/application.master. Now at first glance, you may not think this is a big deal, since the majority of the files in the _layouts directory require administrative-type privileges. However, there are files such as Upload.aspx that are visible to many users. Upload.aspx is located in the _layouts directory and references application.master as its master page file. This means that Upload.aspx will display the default Microsoft look and feel, regardless of what master page settings you have specified for the site master and the system master. This is both frustrating and disappointing; the whole premise of the master page architecture is to easily provide a consistent look and feel to all the pages throughout your site. It is difficult to achieve this nirvana when

you cannot customize *all* master page settings. We can only hope that Microsoft corrects this in the next major release of SharePoint. So why not simply modify the application.master in the _layouts directory? Customizing any of the default Microsoft files renders your deployment unsupported, and future service packs or updates have the potential of overriding your changes. If you have multiple Web applications and multiple site collections with different branding requirements, by default they all share the same application.master. If you apply the branding from one site collection to the application.master, the other site collections will automatically inherit that brand. So how do you customize the application.master? There are a couple of options out there, none of which are officially blessed by Microsoft at the time of writing. One method involves writing a custom httpmodule that basically remaps the master pages used by the pages in your site. A more common method involves making a copy of the _layouts directory in a different location on your server, for example c:\custom_layouts and modifying the application.master using Visual Studio 2005, or notepad. Do *not* use SharePoint Designer to modify your application.master file. SharePoint Designer automatically makes changes to tokens and references in the master page that will actually break the page. Once you have modified the application.master and saved your changes, all that remains is to set the path of the _layouts folder of your Web application in IIS to point to the new custom _layouts folder. This technique preserves the default system files and also allows you to specify different _layouts folders for each Web application. The disadvantages to this technique are that you need to have multiple copies of the _layouts directory, and depending on your implementation, this can become difficult to maintain. The custom _layouts folders will not receive future service packs and hotfixes, and although this has the positive effect of maintaining your customizations, the downside is that you may not receive critical fixes. Additionally, an update in a different system folder may require a specific update in the _layouts folder resulting in problems on your site. When creating a customized application.master, be sure to include all the content placeholders required by that specific master page, otherwise your pages will not render correctly in the browser.

14.3.5 Page Layouts

SharePoint Server 2007 introduces the concept of *page layouts*, which in conjunction with master pages add an extra layer of separation between the page presentation and the actual content that appears on the page. All page layouts reference the custom master page and will therefore inherit the settings that you apply to this master page. A page layout has an associated content type that determines the type of content stored on pages instantiated with that page layout. The content type defines the columns representing the fields that can appear on a page. For example, you may have a content type named

Profile that contains site columns such as Contact Name, Contact E-Mail Address, Job Title, Hobbies, Contact Picture, and Page Content. The actual content of a page is stored as a SharePoint list item in the Pages document library (Figure 14.16). If you examine the properties of a page instantiated from a specific page layout, you will see that the actual content is stored in columns associated with the page item and not physically on the page. The columns defined in the content type associated with a page are mapped to field controls on the page layout. When you create a page layout, you add the appropriate field controls to the page either by using an editor such as SharePoint Designer 2007 or by authoring the markup directly on the page.

Figure 14.16
Page instantiated from page layout.

Creating page layouts is a straightforward process. Once you have your content type defined or have chosen the content type to associate with the page layout, you can then create a new page layout using SharePoint Designer 2007. You can associate multiple page layouts with the same content type, allowing you to present the same information in multiple formats. SharePoint Designer 2007 has a special tab on the New dialog for creating SharePoint content. The Page Layout option resides in the SharePoint Publishing section of the SharePoint Content (Figure 14.17).

During the initial page-generation process SharePoint Designer 2007 retrieves the content type information specific to the page layout and displays the corresponding field controls in the SharePoint Controls tool. Once the page is created and the controls are loaded in the toolbox, you are free to

design your page by adding tables, field controls, and any desired HTML markup. Like master pages, page layouts are template files also stored in the _catlogs\masterpagegallery. When you create a page layout in SharePoint Designer 2007, it is automatically saved in the master page gallery, regardless of where you are in the site hierarchy.

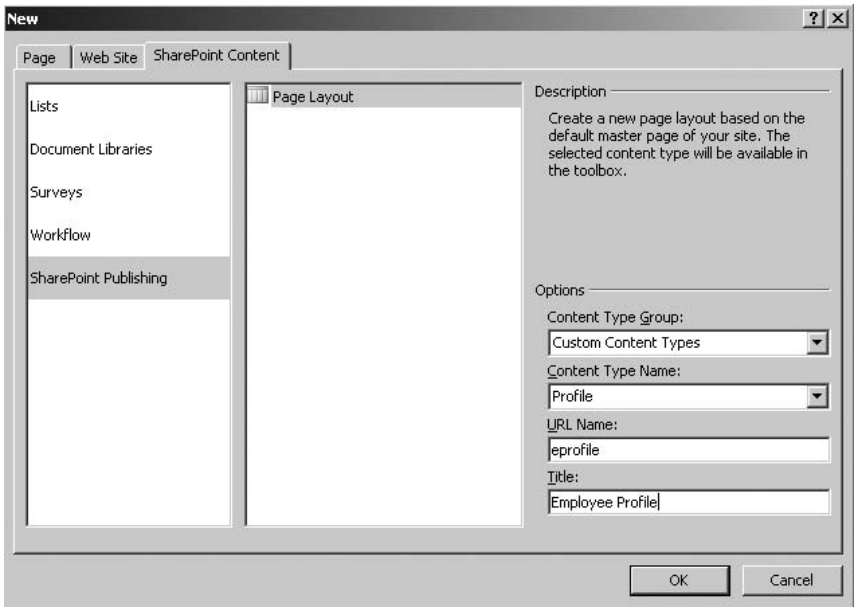


Figure 14.17 *Creating a new page layout.*

14.3.6 Publishing Page Rendering

Pages created from a page layout go through a more involved rendering process than regular WSS 3.0 content pages. When a user requests a content page, the page layout associated with the page is retrieved and the custom master associated with the site is retrieved rendering the chrome of the page. Then, finally, the field controls in the page layout render content from the page's fields into the page. Figure 14.18 illustrates the page rendering associated with publishing pages.

14.3.7 Navigation

When it comes to navigation, SharePoint Portal Server 2003 provides little in the way of customization. Web designers and administrators often want to change the navigation on a site to at the very least add custom links. To achieve this in SharePoint Portal Server 2003, you either had to plan in advance and customize the actual site definitions or edit the site using FrontPage 2003, add the appropriate links, and in the process risk unghost-

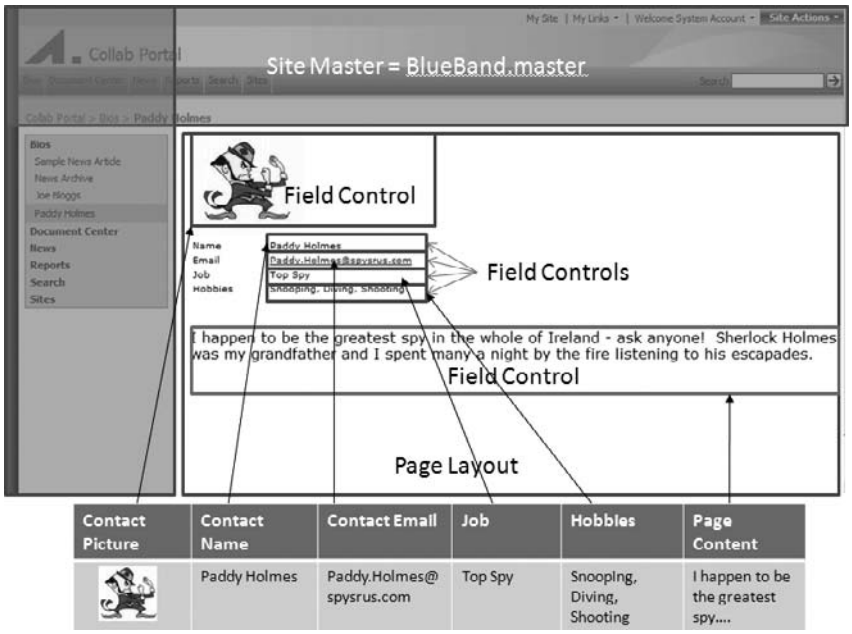


Figure 14.18
Publishing page rendering.

ing your pages. Other options existed, including being creative with the content editor Web Part and writing custom code, but none provided a flexible, convenient method for customization. WSS 3.0 is built on top of ASP.NET 2.0, and as such leverages much of the functionality that ASP.NET 2.0 brings to the table, including various provider models and associated controls that greatly improve overall navigation. The default navigation for SharePoint Server 2007 is based on the site hierarchy. A typical collaboration portal consists of the parent site and a collection of subsites (Document Center, News, Reports, Search, and Sites), and the navigational structure provides access to each site through the horizontal navigation bar that appears towards the top of the page and the vertical navigation bar that appears on the left of the page. Each of the navigation bars is customizable through the browser, through SharePoint Designer 2007, and through custom code.

If you examine the navigation areas of one of the default master pages, such as default.master, you will notice that the navigation is provided by ASP.NET 2.0 navigation Web controls. The navigation controls include the `Menu`, `TreeView`, and `SiteMapPath` controls. The `Menu` control displays a menu that can be aligned either vertically or horizontally, the `TreeView` control displays the entire site map hierarchy in a collapsible vertical tree, and the `SiteMapPath` displays a breadcrumb. Figure 14.19 shows how the `Menu` control is used to define the top navigation bar in the default.master. You can easily customize the `Menu` control and apply your own styles or reference a

custom style sheet, thereby changing the look and feel of the navigation on all pages that inherit your master page. Settings such as `StaticDisplayLevels` and the `MaximumDynamicDisplayLevels` control how many levels of the hierarchy should appear in the navigation and whether they should appear as flyouts. The `Orientation` setting allows you to define whether your menu should appear as a horizontal menu or a vertical menu. The `Orientation` of the top navigation bar in a typical SharePoint site is set to `Horizontal`, whereas the `Orientation` of the Quick Launch menu, or vertical navigation bar, is set to `Vertical`.

```
<SharePoint:AspMenu
  ID="TopNavigationMenu"
  Runat="server"
  DataSourceID="topSiteMap"
  EnableViewState="false"
  AccessKey="<%$Resources:wss,navigation_accesskey%>"
  Orientation="Horizontal"
  StaticDisplayLevels="2"
  MaximumDynamicDisplayLevels="1"
  DynamicHorizontalOffset="0"
  StaticPopoutImageUrl="/_layouts/images/menudark.gif"
  StaticPopoutImageTextFormatString=""
  DynamicHoverStyle-BackColor="#CBE3F0"
  SkipLinkText=""
  StaticSubMenuIndent="0"
  CssClass="ms-topNavContainer">
    <StaticMenuStyle/>
    <StaticMenuItemStyle CssClass="ms-topnav" ItemSpacing="0px"/>
    <StaticSelectedStyle CssClass="ms-topnavselected" />
    <StaticHoverStyle CssClass="ms-topNavHover" />
    <DynamicMenuStyle  BackColor="#F2F3F4" BorderColor="#A7B4CE" BorderWidth="1px"/>
    <DynamicMenuItemStyle CssClass="ms-topNavFlyOuts"/>
    <DynamicHoverStyle CssClass="ms-topNavFlyOutsHover"/>
    <DynamicSelectedStyle CssClass="ms-topNavFlyOutsSelected"/>
</SharePoint:AspMenu>

  <SharePoint:DelegateControl runat="server" ControlId="TopNavigationDataSource">
      <Template_Controls>
          <asp:SiteMapDataSource
            ShowStartingNode="False"
            SiteMapProvider="SPNavigationProvider"
            id="topSiteMap"
            runat="server"
            StartingNodeUrl="sid:1002"/>
      </Template_Controls>
  </SharePoint:DelegateControl>
```

Figure 14.19
AspMenu control displaying the top navigation bar.

The `DataSourceID` attribute of the `Menu` control maps to the data source control used to provide the hierarchical data for the menu. Figure 14.19 shows that the `DataSourceID` maps to a `SiteMapDataSource` control. The SiteMapDataSource control is an ASP.NET 2.0 control. SharePoint has its own data source control (`PublishingNavigation:PortalSiteMapDataSource`) that knows how to retrieve data from specific site map provider objects. The data source control uses the `SiteMapProvider` property to determine the provider to use to retrieve data. In our example, the `SiteMapProvider` referenced is the `SPNavigationProvider`. The `SPNavigationProvider` is one of several SharePoint-specific site map provider objects used to retrieve hierarchical data from SharePoint. You can discover the list of

Figure 14.20 *Named site map providers for SharePoint Server 2007.*

named providers that comes with SharePoint Server 2007 by examining the `siteMap` element in the application's Web.config (Figure 14.20).

You can register your own custom site map providers by placing an entry in the Web.config, and once they are registered, you can reference the provider in the `SiteMapProvider` property of the data source control. Figure 14.21 shows a sample site map entry in Web.config.

```
<add name="MyCustomNavigationProvider"
     type="MyCustomNav.Navigation, MyCustomNav,
     Version=1.0.0.0,
     Culture=Neutral,
     PublicKeyToken=898a5ad19ca9d788"
     NavigationType="Current"
/>
```

Figure 14.21 *Code to declare a Custom Site Map Provider*

The data source control has a number of other properties that define settings such as where in the site hierarchy the data source should start, and

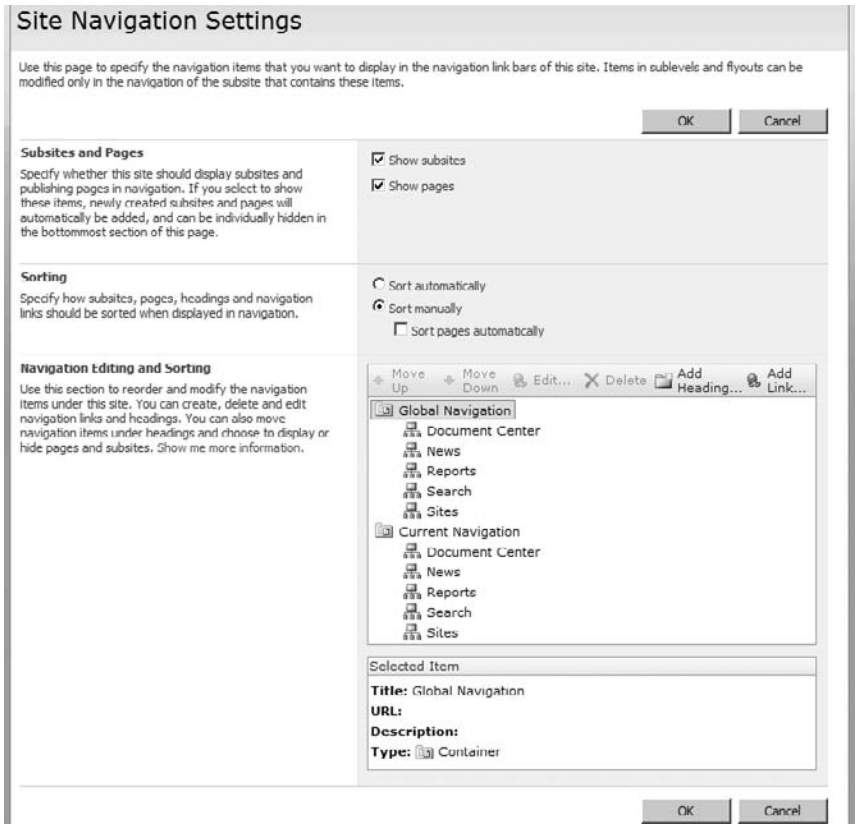


Figure 14.22
Site navigation settings page.

what type of nodes should be included in the hierarchy—for example, whether the navigation should be restricted to Web sites only and exclude all pages, headings and links. For more information on the data source control properties and how to create a custom data source, reference the WSS 3.0 SDK.

Having the ability to create your own custom data sources for navigation provides a lot of flexibility when it comes to designing your sites. However, you don't have to get that detailed if, for example, you simply want to add a few extra links to your vertical menu. The browser-based customization capabilities with respect to navigation are a welcome and much-needed addition to SharePoint. The Look and Feel menu on the Site Settings page contains a link to a page from which you can modify the navigation (Figure 14.22). The horizontal navigation that appears at the top of a SharePoint Web page is referred to as the global navigation, and the vertical navigation that appears on the left of the page is referred to as the current navigation. You can determine whether the site should display child sites and pages in the navigation, and whether to sort the sites and pages appearing in the nav-

igation automatically or manually, and finally you can modify the actual entries in the navigational structure. Using the Add Heading functionality, you can add specific headings in your navigational structure and choose whether the headings actually link you to a page. Additionally, you can add your own custom links to the navigational hierarchy.

The Look and Feel menu on the Site Settings page also includes a link to the Tree View Settings page, which allows you to enable and disable both the Quick Launch menu and the Tree View menu. The Quick Launch menu is the vertical menu, or current navigation, that appears on the left hand side of an uncustomized SharePoint site. Figure 14.23 shows the home page of a regular collaboration portal with the Tree View setting enabled and the Quick Launch setting disabled.

Figure 14.23 *Collaboration portal with Tree View enabled.*

14.4 Customizing the Color Scheme

As with previous versions, SharePoint Server 2007 and WSS 3.0 make extensive use of cascading style sheets (.css) to define the color scheme and layout of their pages. Cascading style sheets allow you to assign several properties at once to all of the elements on an HTML page marked with a particular tag. The term cascading style sheets implies that two or more style sheets can cascade together to produce the final look of the page. The language used for creating cascading style sheets is CSS, a simple declarative language that allows you to apply stylistic information, such as the font and color, to documents written in HTML or XML. Elements on the page are associated with

properties and values in the style sheet. For example, Figure 14.24 declares that BODY elements are 10-point blue text.

Figure 14.24
Sample CSS Code

```
Body
{
   Font-size: 10pt;
   Color: blue;
}
```

The main style sheet that controls the color scheme and layout for SharePoint Server 2007 and WSS 3.0 is core.css, which can be found in the: `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\LAYOUTS\<localid>\STYLES` directory. Core.css contains numerous classes that are used to define specific elements on a page. For example, the `ms-bannerframe` class controls the background color of the navigation bar that appears at the top of the page. How do you know which class controls which element on the page? The WSS 3.0 SDK provides a table that describes each class defined in core.css. The SDK also provides sample code (Figure 14.25) that you can add to a page, which applies a tool tip to each element, describing the class that controls that element.

Figure 14.25
Script code to identify CSS classes.

```
<script language="jscript">
function ClassInfo()
{
  if (window.event.srcElement.className != null)
  {
    stsclass.innerText = window.event.srcElement.className;
  }
  else
  {
    stsclass.innerText = "";
  }
}

window.document.body.onmouseover  = ClassInfo;</script>

<div style="border-style:solid;border-width:1px; width: 281px; height: 34px;
position: absolute; left: 286px; top: 41px; z-index:15; padding-left:4px;
padding-right:4px; padding-top:2px; padding-bottom:2px; background-color:#EEEEF4">
 <p id="stsclasstitle">
   <font face="Tahoma" id="stsclasstitle">Classname: </font>
   <font face="Tahoma"id="stsclass"> </font>
 </p>
</div>
```

The code can be inserted into a content editor Web Part to show the classes specific to that page. Alternatively, you can include the code within the `BODY` element of your master page, which will allow you to see the classes across all the pages that inherit that master page. The code basically adds an area on your page that displays the title of the class applied to the region over which your mouse is hovering (Figure 14.26). Notice that the mouse pointer is hovering over the title area of the page, and the style that is being used by the title area is `ms-sitetitle`.

Figure 14.26 *Home page with CSS script applied to master page.*

An excellent tool for anyone customizing the look and feel of SharePoint is the Internet Explorer Developer Toolbar, which is available as a free download on the Microsoft Web site. Once installed, it provides tools that allow you to explore the document object model (DOM) of a Web page, view the source of HTML and CSS, and view HTML object class names and IDs.

14.4.1 Creating Custom Style Sheets

You should not customize core.css directly; instead, you should create a custom style sheet that contains all the classes that you would like to override. You could then store this file in the same directory as core.css (`\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\LAYOUTS\<localid>\STYLES`) and reference your new custom style sheet. Alternatively, if you wanted to take advantage of the enterprise content management capabilities of SharePoint Server 2007, instead of storing the file directly on the file system you would upload your custom style sheet to a library such as the style library, and benefit from the version control and publishing model.

There are several options when it comes to referencing custom style sheets in SharePoint, and the option you select depends on what you want to

achieve. Core.css by default is applied last when rendering a page. One way to ensure that your custom style sheet is always called last is to set the alternate CSS URL for the site to reference your custom style sheet. The alternate CSS URL setting can be controlled by setting the AlternateCSS attribute in a site definition, by populating the `AlternateCSSUrl` property of the `SPWeb` object through code, or by populating the alternate CSS URL setting through the browser user interface. We will revisit the site definition and object model options in subsequent chapters, but for now let's focus our attention on the browser option. The alternate CSS URL property can be set from the Master Page Settings page, accessible from the Look and Feel menu on the Site Settings page. If you scroll down to the very bottom of the Master Page Settings page you will notice a section specific to the setting of the alternate CSS URL (Figure 14.27). By default, SharePoint sites use the WSS default core.css styles. To set an alternate CSS, all you need to do is insert the URL of the CSS file to be used by the site into the text field in the Alternate CSS URL section of the Master Page Settings page. Check the box if you would like to push the style sheet setting down to all subsites of the current site.

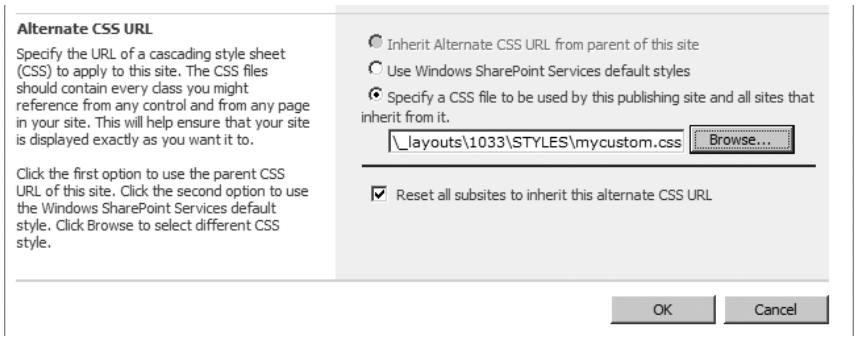


Figure 14.27 *Setting the AlternateCSSURL property through the browser UI.*

There may be occasions when you would like to have different sites within a site collection, each referencing a different custom style sheet. This is easily achieved through the same mechanism as above by setting the alternate CSS URL setting for each site to the URL of the appropriate custom style sheet. If you have a large number of sites requiring different style sheets, this method could quite quickly pose problems from an administrative perspective; however you could easily write a utility that would traverse the sites in your site collection and report the style sheet and master page each site is currently using. The style sheet setting specified in the alternate CSS URL property is guaranteed to override all other styles, since it is called last when rendering a page. Additionally, it also affects _layouts pages such as Settings.aspx and Upload.aspx.

You can also reference your custom style sheet from your master page by using the SharePoint `CssRegistration` control. In a master page, SharePoint calls core.css using the SharePoint `CssLink` control. Any references to custom style sheets should appear after this tag. Using the `CssRegistration` control has its drawbacks, namely the fact that the control does not actually guarantee the order in which the specified style sheet is loaded. This is not really a problem if your custom style sheet references styles that do not exist in core.css, but if you are using your custom style sheet to override the styles in core.css, you should look to using another method such as setting the alternate CSS URL property. Alternatively, you can use a standard ASP.NET `link` control to reference your custom style sheet directly from your master page. If you place the `link` control after the SharePoint `CssLink` control, your custom styles will be loaded after core.css. Figure 14.28 shows the HTML for referencing a custom style sheet using both the `CssRegistration` control and the ASP.NET `link` control. The drawback of using the `link` control method relative to the alternate CSS URL method is that the style sheet only affects pages that inherit the master page containing the reference. As we mentioned earlier in the chapter, the majority of the pages in the _layouts directory are controlled by the application.master, which means that the application.master would have to include the `link` control. Microsoft does not support the modification of system pages such as application.master. From a best practice standpoint, you should try to perform as much customization as possible through the use of custom style sheets and then leverage the alternate CSS URL setting to achieving a consistent look and feel with respect to style settings throughout your site collection.

```
<SharePoint:CssRegistration name="<% $SPUrl:~SiteCollection/Style Library/demo.css%>" runat="server"/>

<link rel="stylesheet" type="text/css" href="<% $SPUrl:~SiteCollection/Style Library/demo.css%>" />"
```

Figure 14.28 *SharePoint: CssRegistration and ASP link tags.*

It is important to note that, as with master page settings, new sites do not inherit the alternate CSS URL setting from the parent. As with master pages there are several techniques to choose from should you want newly created sites to reference a specific alternate CSS file upon site creation. These techniques are discussed in Chapter 15, Site Definitions nad Templates and Chapter 16, Features.

14.5 Themes

Designing a Web page so that it is both visually appealing and effective is of paramount importance to the ultimate success of the site. Not all Web sites are created by professional Web designers who know all of the intricacies and

design techniques such a post entails. Microsoft helps to produce a professional look and feel to a site quickly, not only through the use of master pages, page layouts, and style sheets, but also through the use of themes. Themes are professionally designed style packages that define the color, fonts, and graphical page elements of Web pages within a site.

Microsoft ships a number of themes with SharePoint Server 2007 (Figure 14.29) that you can apply to your SharePoint sites or use as the template for generating your own custom theme. When you apply a theme through the browser user interface, it applies only to the current site; there is no out-of-the-box mechanism for pushing this theme to all subsites within a given site collection. This lack of functionality is unfortunate and defeats the purpose of using themes to provide a consistent look and feel. Applying the same theme to all the sites in your site collection using the browser user interface involves visiting the Site Theme settings page in every site and setting the appropriate theme, although we can't see many organizations implementing this method to provide consistency. The good news is that you can implement this functionality programmatically using the `ApplyTheme` method of the `SPWeb` object. In Chapter 16, Features, we show the code and process to achieve this and it is relatively straightforward; makes you wonder why Microsoft didn't provide this option through the browser user interface in the first place! Another piece of good news with respect to themes is that they affect the _layouts pages. Where you place your style sheet references in your master page relative to the theme reference and its corresponding `meta` tag ultimately determines which style sheets will prevail (see Figure 14.30). Themes override style sheet references made in the master page, such as those set by the `CssLink` control, the `CssRegistration` control and the ASP.NET `link` control, they also override the alternate CSS URL .

So based on the information we have so far, you might conclude that as far as branding is concerned, themes are the way to go to if you want get a consistent look and feel. However, there are significant limitations with the way themes are actually applied behind the scenes that make them undesirable as far as branding is concerned. When a theme is applied to a site, a folder named _themes is created in the site hierarchy containing all the necessary files that comprise the theme, such as the image and style sheet files. This folder is almost a replica of the matching theme folder stored on the file system in the `\Program Files\Common\Microsoft Shared\Web server extensions\12\TEMPLATE\THEMES` directory. Why *almost* a replica? It contains an additional .css file that is basically a merger of the theme.css and the mossextensions.css files for the theme. The file, typically named something similar to obsid1011-65001.css, is generated when you apply the theme to your site; it is from this merged file that the styles are applied to your site. This design causes problems should you want to update the styles in the theme and have that update apply to all sites that have the theme applied and to future applications of the theme to sites. Simply

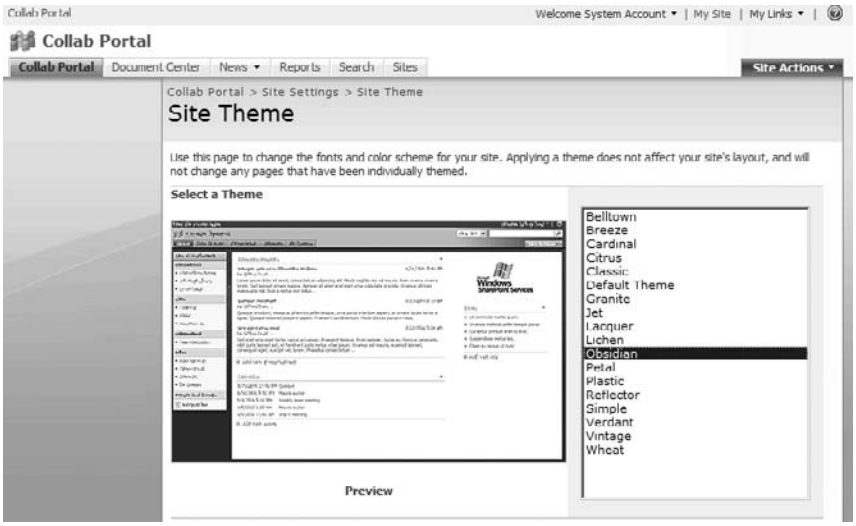


Figure 14.29
Site theme settings page.

updating the style files on the file system will not suffice, since once the theme has been applied to the site the site no longer references these files; instead it references the merged copy stored in the content database. Effectively, it's the whole unghosting/ghosting issue again, only this time the unghosting happens when you apply a theme. What are your options in this scenario? In addition to editing the themes file system files, you could apply your edits to the merged .css file in each site, but this would be incredibly cumbersome and would cause additional unghosting issues should you need to reapply the theme to the same site at a later date. Another alternative would be to reapply the theme, causing SharePoint to merge the updated style files; you would need to do this for each site that had the theme applied either through the browser or programmatically. Note it is not just a matter of navigating to the site page and clicking the apply button—you have to select a different theme, apply it, and then reselect the theme that you updated and click apply once more. If you apply a theme to the *root* of the site collection, and the root site references custom master pages that were loaded into the system using the enterprise content management capabilities, then a theme related `meta` tag is inserted into *all* the custom master pages in the master page gallery directly connecting these master pages to the site theme. A custom master page in this particular scenario refers to any master page that was not provided by SharePoint Server 2007, WSS 3.0, or any customized SharePoint Server 2007/WSS3.0 master page; in other words, any master page that resides in the content database.

All subsites that reference a custom master page automatically inherit the site theme applied at the root of the site collection. The subsite does not have to inherit the same custom master page as the parent for this behavior to

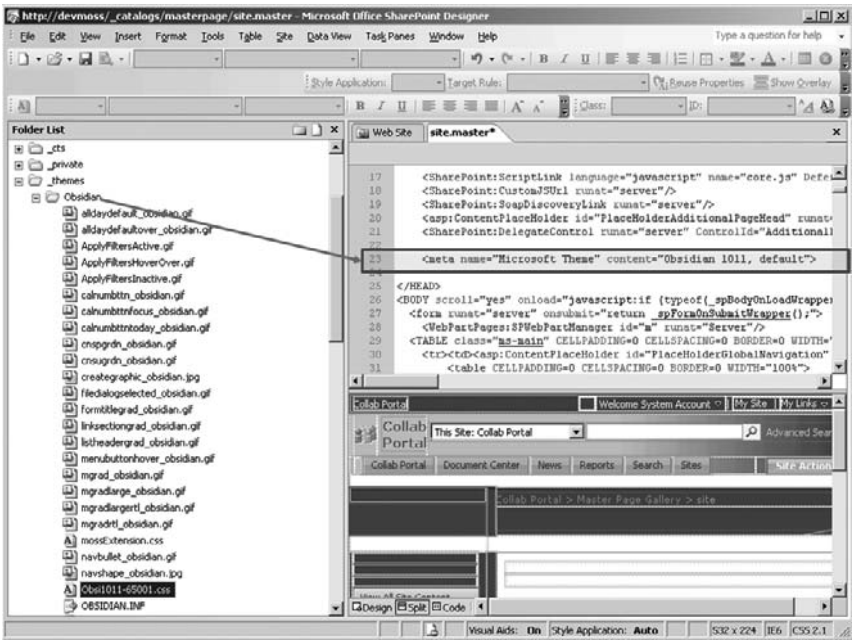


Figure 14.30
Theme folder and meta tag added.

occur. But we said earlier that themes were applied on a site-per-site basis and could not be pushed to other sites through the browser user interface. Doesn't the behavior we're now describing contradict our earlier statement? The answer to both these questions is yes, and the behavior described is not something that you should rely on to brand your sites; it is more of an anomaly than anything. If all of the sites in your site collection inherit any of the default out-of-the-box uncustomized master pages, such as default.master or BlueBand.master, this behavior does not occur. It only happens with *custom* master pages that have been processed through the content management lifecycle. So what is the issue with this behavior anyway? Things become problematic when you want to apply a *different* site theme to a subsite that inherits a custom master page. When a site theme is applied in this scenario, a folder named _themes is created in the site directory of that specific site. Once again, the _themes folder contains the files that comprise the theme; however, this time no changes are applied to the custom master pages that the site references, since the custom master pages are stored at the root of the site collection. This means that the custom master pages for the site still reference the site theme applied at the root site, and therefore the site will not pick up the newly applied theme. However, all _layouts pages in the site *will* adopt the new theme, resulting in a very inconsistent branding story for the site. Additionally, if the theme applied at the subsite level contains styles not referenced by the theme at the root level but referenced in the custom master

pages, the pages that reference the custom master pages will adopt some of their style settings from the subsite theme.

Let's look at an example. The root site of the collaboration portal site collection is set to customa.master. The Documents Center site, which is a subsite of the root site, is set to customb.master and the Sample team site is set to customc.master. The Obsidian site theme is applied to the root site, the Petal site theme is applied to the Documents Center site, and the Simple site theme is applied to the Sample team site. Consequently, each site contains a _themes folder, with the files specific to the theme applied to the site: the root site contains a _themes folder with the Obsidian theme files; the Document Center site contains a _themes folder with the Petal theme files and the Sample team site contains a _themes folder with the Simple theme files. If you inspect the contents of each custom master page, you will find a `meta` tag that references the Obsidian theme, because it is applied at the root of the site collection, but no references to the Petal or Simple themes. This `meta` tag causes the pages of each subsite that references a custom master page to inherit the styles from the theme applied at the root site of the site collection, thus overriding the styles from each site-specific theme; the _layouts pages for each site inherit the styles from the site theme (see Figure 14.31). Figure 14.32 shows the scenario in which the Obsidian theme has been applied at the root site, custom master pages are being used and have been pushed to all subsites, and the subsite has the Reflections theme applied. Notice the home page now adopts style settings from both themes, creating an ungainly look and feel. If you set the site theme at the root of the site collection back to the default theme, the theme edits are removed from the custom master pages, and the subsite will now adopt the site theme setting on its own site. Furthermore, if you change the master page settings at the root of the site collection to point to different master pages, edits made to the custom master pages when applying the site theme are not removed. The next time you apply the custom master pages to your site, they will still contain the same theme references regardless of whether you have themes applied to your site.

So is there any more bad news to share? Unfortunately, yes, and it's the same problem that we mentioned when discussing the master page and style sheet settings: new SharePoint sites do not automatically inherit the site theme settings from the parent site. Once again, all is not lost and Chapter 16, Features, shows how we can avoid this problem with the use of Features and some custom code that implements the `ApplyTheme` method of the `SPWeb` object setting the newly created Web to that of the parent.

14.5.1 Creating a Custom Theme

If you would like to leverage themes in your deployment and find that the default themes that SharePoint provides do not meet your needs, you can

Home page of Team Site with Obsidian theme applied at root and Verdant theme applied at Site level.

Home page picks up root site theme: Obsidian

_layouts page of Team Site with Obsidian theme applied at root and Simple theme applied at Site level.

_layouts pages pick up current site theme: Simple

Figure 14.31 *Inconsistent branding using themes.*

Figure 14.32 *Colliding themes.*

create a new theme or customize an existing theme. As previously mentioned, SharePoint themes are stored on the file system in `\Program Files\Common\Microsoft Shared\Web server extensions\12\TEMPLATE\`

THEMES folder. In this folder, SharePoint retrieves the list of themes from the theme template definition file (SPTHEMES.XML), which is located in the \Program Files\Common\Microsoft Shared\Web server extensions\12\TEMPLATE\LAYOUTS\<localid> folder. When you create a new custom theme, you must add a template definition for the custom theme in spthemes.xml for your theme to appear as a selection in the browser user interface. To help users identify the branding that your theme applies, you should provide thumbnail and preview image files for your theme. The easiest way to create a custom theme is to make a copy of the theme that most resembles the style you are trying to achieve, and modify the styles and associated images until you achieve the desired effect. The following steps show how to create a new custom theme and display it as an option in the Site Theme settings page.

1. From the file system of your SharePoint server, navigate to the \Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\THEMES directory.
2. Make a copy of the theme that most closely matches your needs, and rename the folder with a unique name that describes your custom theme—for example, SpysRUsTheme.
3. Open the new folder and locate the .INF file. Rename this file to match the name of the new folder—for example, SpysRUsTheme.INF.
4. Edit the .INF file and assign the same name to the title attribute in the [info] section of the file.
5. Customize the styles defined in the cascading style sheets associated with this theme (theme.css, mossextensions.css). Include any SharePoint Server 2007 and WSS 3.0 classes that may be missing from the theme and add any new styles that you would like to reference in your custom master pages.
6. Customize the image files in the new folder as desired.
7. Create a thumbnail and preview image for your theme, and save the files to the \Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\TEMPLATE\IMAGES directory.
8. Navigate to the \Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\TEMPLATE\LAYOUTS\<localid> directory and make a backup copy of the SPTHEMES.XML file before you edit the file.

9. Edit the file SPTHEMES.XML, and add a definition for your new theme. Themes appear in alphabetical order on the Site Theme settings page, regardless of the order they appear in the SPTHEMES.XML file. Figure 14.33 shows the code required for the template definition and the final result of the modification, with the custom template appearing at the top of the theme list in the Apply Theme to Web Site page.

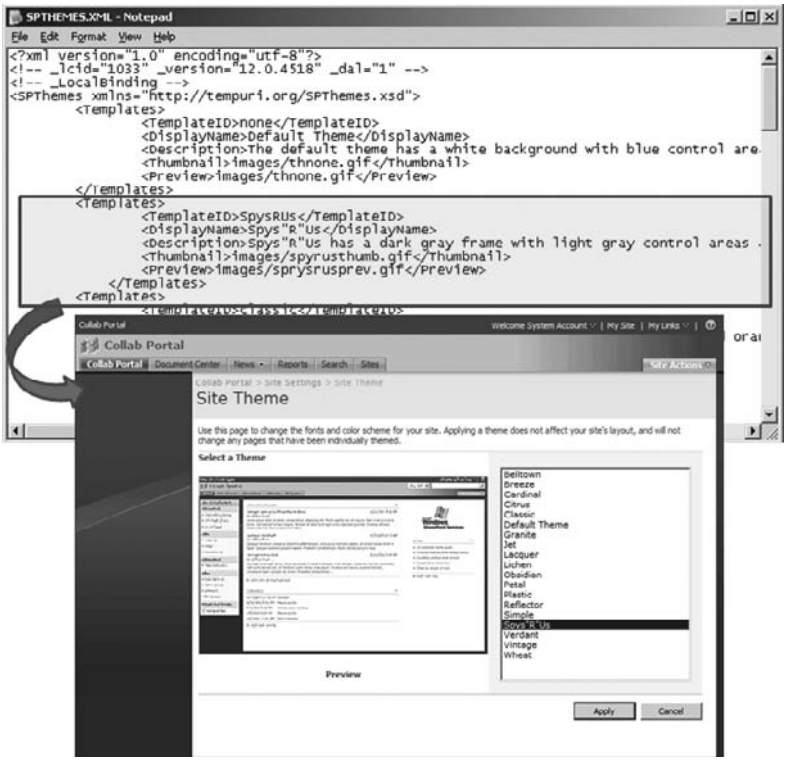


Figure 14.33 *Custom theme template definition.*

14.5.2 To Theme or Not to Theme

So after all is said and done, is it a good idea to use themes? Themes are great for packaging together a brand, but as we have seen in this section, they often behave unpredictably making it difficult to achieve and maintain a consistent brand across all sites in a site collection and across multiple site collections. The most reliable method to date for ensuring consistency across all sites is the use of the alternate CSS URL, in conjunction with themes. For example, if you are particularly fond of the layout and graphics of a specific SharePoint theme and would really like to apply a custom version of the theme across all the sites in a site collection, you could do the following:

1. Create a custom theme using the steps mentioned in the previous section, although you do not have to make an entry in the SPTHEMES.XML file in this instance. You could also remove all entries except your new custom entry, thus preventing users from overriding your customizations (Figure 14.34). You could even leave the entire list blank, preventing users from selecting a theme at all. Just be sure to make a backup copy of SPSTHEMES.XML before any editing occurs.
2. Create an empty .css file in your theme folder and copy and paste your customized contents from the both theme.css and mossextensions.css into your new style sheet.
3. Copy and paste your new theme folder into the _layouts directory, making your files accessible via a URL.
4. Using the browser user interface, set the alternate CSS URL at the root of the site collection to map to your new style sheet in the _layouts directory and push this setting down to all subsites.

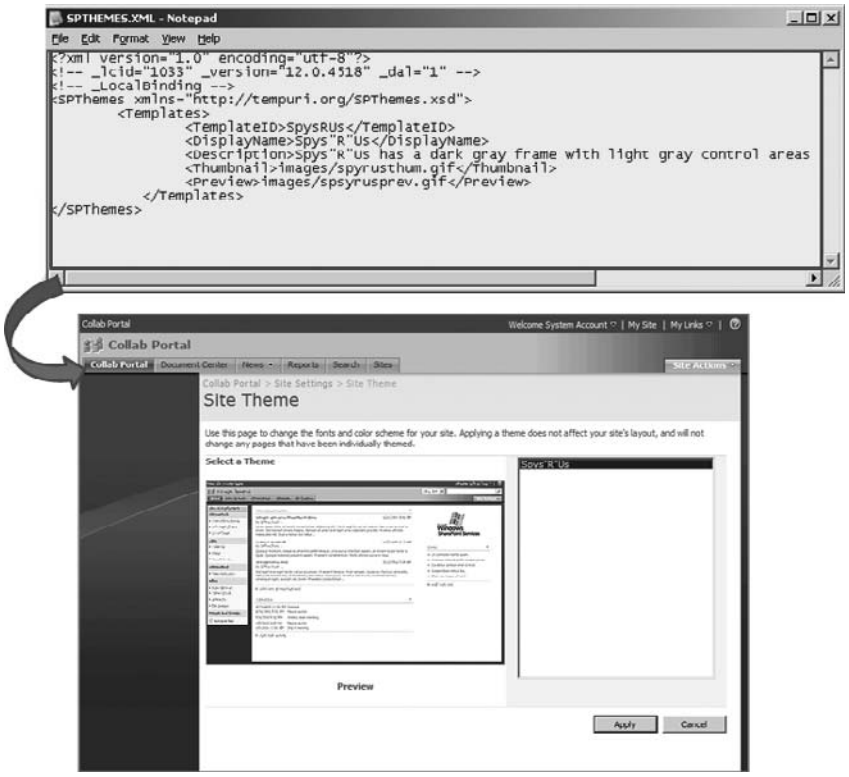


Figure 14.34
Only custom theme visible.

This method allows you to keep all your branding files together in one place, making future branding updates much simpler. Removal of *all* the templates from site theme list would prevent users from applying a theme and ultimately customizing/unghosting theme files through SharePoint Designer 2007. But what if you want to use multiple themes throughout various sites in your site collection? You can use the same method, but apply different alternate CSS URL settings to different sites. Also, instead of storing the theme files on the file system of the Web server, upload all the files to a folder in the styles library and take advantage of all the versioning features of SharePoint, in addition to centralized management of the files in the case where you have multiple front-end Web servers in your deployment.

14.6 Site Title, Description, and Icon

By default, the site title and site description are populated by settings specified during site creation. The default logo of the site is not definable during this process, and for most sites the logo is typically set to /_layouts/images/titlegraphic.gif, which is the default Microsoft site logo. After site creation, you can change the title of your site, the description of your site and the icon associated with your site. Figure 14.35 shows the settings page for these properties in the root site of our collaboration portal, and Figure 14.36 shows the result of applying these settings.

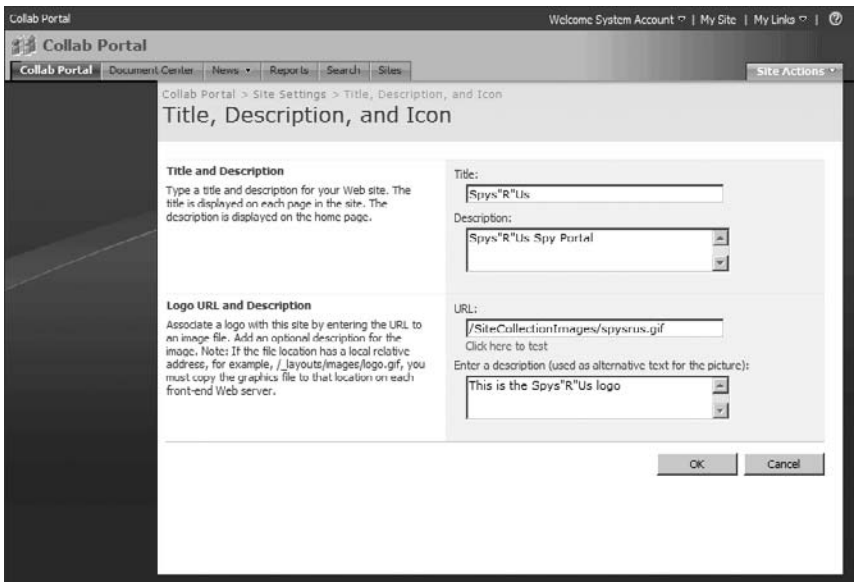


Figure 14.35 *Setting the title, description and site logo.*

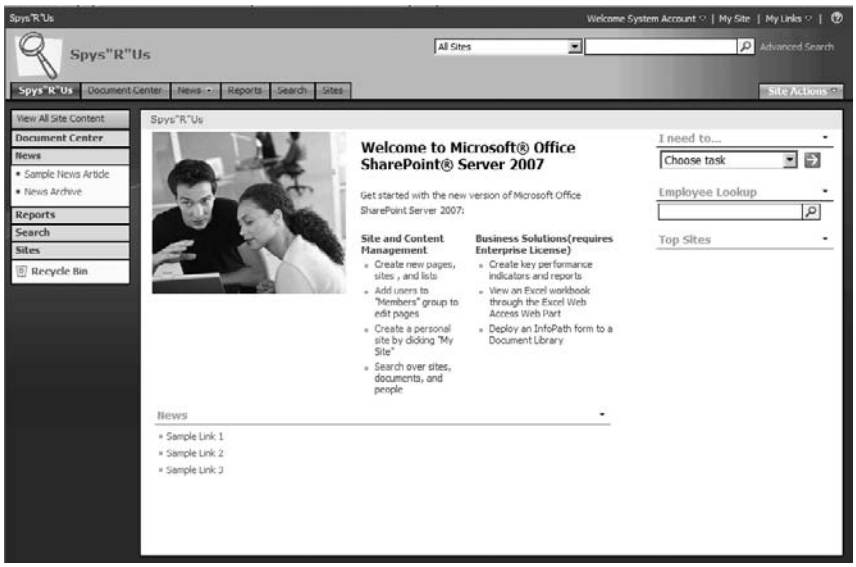


Figure 14.36 *SpysRUs collaboration portal with settings applied.*

These settings apply to all pages in the site, even the _layouts pages, which means that even the illusive _layouts pages adopt your custom logo. The bad news is that, through the browser user interface, there is no mechanism for pushing the logo down to subsites. It makes sense that you wouldn't want the Title and Description to be pushed down, but the logo?! At the very least we should be given the option! Interestingly enough, when you create a new site, it actually inherits the site logo setting from the parent site. So we know that at least one developer in Microsoft considered the concept of inheriting branding-type settings from parent sites at site creation. It is a pity that the application of master page settings, alternate CSS URL settings, and theme settings were not handled in the same way. You could go the old-fashioned route and replace the titlegraphic.gif file with your own logo directly on the file system; however, this method has the usual caveats with respect to modifying out-of-the-box files—they could be overwritten by a future service pack. In Chapter 16, Features, we look at how to change the site logo for all sites across all site collections in your deployment and how to ensure that new sites inherit all the appropriate branding settings from their parent site.

14.7 Basic Branding Exercises

SharePoint Server 2007 provides three levels of customization when it comes to branding: Browser-based customizations; SharePoint Designer 2007 based customizations; and finally more advanced customizations applied through the use of site definitions and Features. In this final section we will step through some exercises that focus on customizations that can be performed

through the browser and through SharePoint Designer. The more advanced customizations using site definitions and Features are covered in Chapter 15, Site Definitions and Templates, and Chapter 16, Features respectively.

Note that the following exercises reference a site collection created from the collaboration portal site template.

14.7.1 Browser Customizations

1. Using your browser, navigate to the Site Settings page at the root of your site collection.
2. From the Look and Feel section, click the Title, description, and icon link.
3. Change that title, description and icon of the root Web site.
4. View the home page and verify that the settings have been applied.
5. From the vertical navigation bar, click the View all Site Content link; your custom logo appears on the All Site Content page, even though this page is located in the _layouts directory.
6. From the horizontal navigation bar, click the Documents Center link. The Documents Center site does not pick up the logo from the parent site. This is because the site logo is a per site setting.
7. Navigate back to the Site Settings page for the root site.
8. From the Look and Feel section, click the master page link.
9. Set the site master page to BlueBand.master and check the box to reset all subsites to inherit this site master page setting.
10. Set the system master page to OrangeSingleLevel.master and check the box to reset all subsites to inherit this system master page setting.
11. Click OK.
12. Navigate to the root site; the home page now reflects the Blue-Band look and feel.
13. Navigate to the News site; the News site reflects the BlueBand design.
14. Navigate to the Document Center site; the Document Center reflects the OrangeSingleLevel.master look and feel. This happens because the pages in the Document Center reference the default master as opposed to the custom master.

15. From the vertical menu on the Document Center home page, click the second-level Documents link. This should bring you to the Documents page. The Documents page reflects the OrangeSingleLevel look and feel because it is a system page. Remember system pages include forms and view pages.
16. Click the Upload link from the Documents menu. The Upload.aspx page appears, but you will notice that it still has the default Microsoft look and feel. The reason for this is that upload.aspx is a _layouts page that references the application.master.
17. Navigate to the Site Settings page at the root of your site and click the Site Themes link.
18. Select the Obsidian theme and click Apply. You should immediately notice that the _layouts pages are affected by themes.
19. Navigate to the home page of the site; the home page still gets its branding from the BlueBand master, not from the theme. This happens because the BlueBand.master file does not reference themes and uses its own custom styles stored in a separate style sheet.
20. Navigate to the Documents Center site, click the second-level Documents link, and click the Upload link to bring you to the Upload.aspx page again. The page still has a Microsoft look and feel because the theme we applied was at the root site and themes are generally site-specific.
21. Navigate back to the Site Settings page for the root site.
22. From the Look and Feel section, click the Master Page link.
23. Set the site master page to default.master and check the box to reset all subsites to inherit this site master page setting.
24. Set the system master page to default.master and check the box to reset all Subsites to inherit this system master page setting.
25. Click OK. The theme is applied when default.master is selected because that specific master page honors themes whereas the Orange and BlueBand themes don't, and default.master uses the same style references in its .css file that are specified in default.master.

14.7.2 Branding the entire site (continues from previous exercise)

1. Upload an image to replace the Microsoft logo to the Site Collection Images library of your site.
2. Navigate to the root site and from the Internet Explorer file menu, click Edit With Microsoft Office SharePoint Designer. Click Cancel when prompted to edit the page layout.
3. The folder list appears on the left, showing the hierarchy of your site. You should notice a new folder name, _themes containing the Obsidian theme files.
4. Navigate to and expand the _catalogs/masterpages folder.
5. From the File menu, click New, then click Page.
6. From the list of pages, click master page.
7. Click OK. A new untitled master page will appear in SharePoint Designer.
8. Delete all the code in the new master page.
9. From the _catalogs/masterpages folder, open the default.master file. Click No when prompted to check out the file.
10. Copy and paste the code from default.master into your untitled master page.
11. From the File menu, click Save and save the untitled master page into the _catalogs/masterpages folder. Give the file a meaningful name, such as mybrand.master.
12. On the preview pane of your master page file, select the Microsoft default logo and click delete.
13. Drag and drop your logo from the site collection images folder to where the Microsoft logo was previously.
14. Locate the vertical Quick Launch menu by searching for AspMenu control: <SharePoint:AspMenu id="QuickLaunchMenu". Change the code settings for StaticDisplayLevels and MaximumDynamicDisplayLevels from 2, 0 to 1 and 1 respectively. This will change the vertical navigation to display dynamic flyouts.
15. From the File menu, click Save to save your changes.
16. From the Folder list pane, right-click your master page file to display the shortcut menu.
17. Click Check In and, when prompted for the type of version, select Publish a major version and click OK.

18. Click Yes when prompted to view or modify the approval status.
19. From the pending documents in the master page gallery, select your new master page and, from the drop-down menu, click Approve/Reject.
20. From the Approval Status section, select the Approved: this item will become visible to others option and click OK.
21. Navigate back to the Site Settings page for the root site.
22. From the Look and Feel section click the master page link.
23. Set the site master page to your new master page, for example mybrand.master, and check the box to reset all subsites to inherit this site master page setting.
24. Set the system master page to your new master page, for example mybrand.master, and check the box to reset all subsites to inherit this system master page setting.
25. Click OK.
26. Navigate back to the home page of the root site. You will notice that the Quick Launch menu has changed to display flyouts, and the logo is now independent of the site logo setting in the browser user interface.
27. Navigate to the Document Center site and you should see that the site has not picked up the color scheme of the root site, but it does contain the site logo and the Quick Launch customizations since these are coming from the master page. The reason it is not picking up the site theme is because a theme is site-specific. Also, because we applied the theme to the site before creating our new master page, the master page does not have the <meta> tag of the theme. If we were to reapply the theme, this would insert the tag and all subsites would inherit the same theme. But remember, this causes problems and conflicts when styles collide and is not really how themes were designed, so it is better to use the alternate CSS URL setting.
28. Navigate to the Site Settings page at the root of your site and click the Site Themes link.
29. Select Default Theme and click Apply.
30. Navigate back to the home page of the root site. You will notice that the look and feel is back to the default Microsoft look and feel, but the master page is still in effect showing the logo and the Quick Launch customizations. Next we'll create our custom style sheet to match the Obsidian theme.

31. From the file system, navigate to the `\Program Files\Common\Microsoft Shared\Web server extensions\12\TEMPLATE\THEMES` directory.
32. Copy and paste the Obsidian theme, and rename the new folder to MyStyles.
33. Create an new empty .css file in the MyStyles folder named MyStyles.css, and copy and paste the contents from the both theme.css and mossextensions.css into the new style sheet. If you wish, you can customize the style sheet, changing colors, and so forth.
34. Copy and paste the MyStyles folder into the _layouts directory making your files accessible via a URL.
35. Navigate back to the Site Settings page for the root site.
36. From the Look and Feel section click the master page link.
37. At the bottom of the page, set the Alternate CSS URL to map to your new style sheet (/_layouts/MyStyles/MyStyles.css).
38. Check the Reset all subsites to inherit this alternate CSS URL box to push the style sheet to all the subsites.
39. Click OK.
40. Navigate back to the home page of the root site. You will notice that the look and feel is back to the look and feel defined in your new style sheet.
41. Navigate to the Document Center and notice that it picks up the look and feel as defined by the new style sheet.
42. From the vertical menu showing the Document Center site hierarchy, click Documents.
43. From the Documents menu, click Upload. Notice that the Upload.aspx picks up the styles defined by MyStyles.css, but the pesky logo still reflects the Microsoft logo. Remember the Upload.aspx references application.master, and not your master page. Since we are only changing the logo, we'll take the easy route and simply overwrite the titlegraphic.gif with our own image. However, if you wanted to make additional changes to the _layouts branding, you would go the route of making a copy of the _layouts directory, customizing the application.master, and setting your Web application to reference it instead.
44. From the file system, create a new folder to store your custom layouts folder. For example C:\MyCust\.

45. From the file system, navigate to the `\Program Files\Common\Microsoft Shared\Web server extensions\12\TEMPLATE\IMAGES` directory.
46. Rename the image file titlegraphic.gif to titlegraphic.bck.
47. Copy and paste your logo file into the IMAGES directory and rename the file to titlegraphic.gif.
48. Using the browser user interface, navigate back to the Upload page. Notice that even though the page references application.master, it now displays your custom logo.

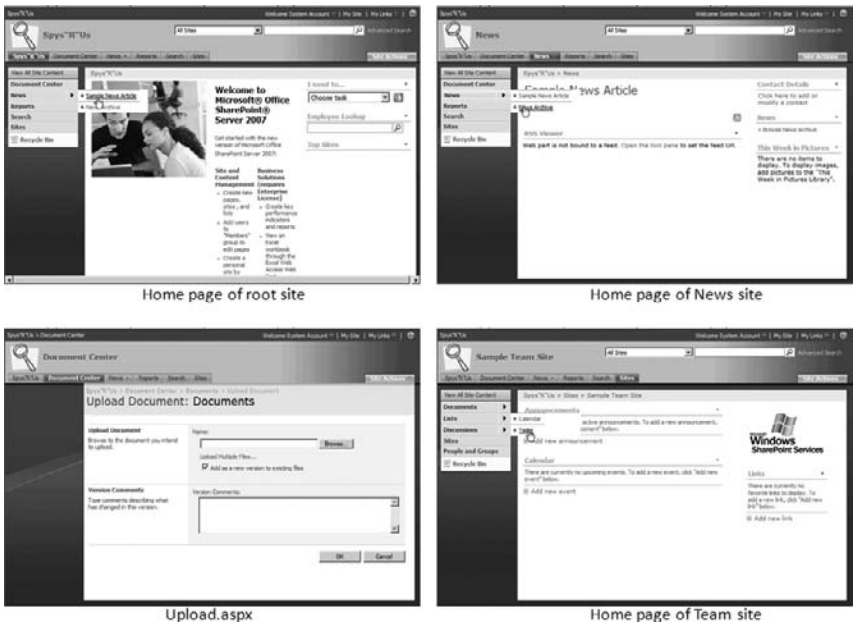


Figure 14.37 *Final Result Branding applied throughout Site Collection*

Throughout this chapter we have discussed the basic concepts and components involved in branding a SharePoint site. Our branding exercise showed how to create our own custom master page and style sheet and apply it to all the sites in our site collection. However, we still have the issue that newly created sites will not pick up our new branding. In the next two chapters we will examine some options to ensure that new sites adopt the appropriate look and feel.

This page intentionally left blank

15

Site Definitions and Templates

SharePoint provides a templating architecture that allows you to create a definition of a site and to use this definition as the template from which to create new sites. The site definition defines the components that make up a site, such as the navigational structure, the welcome page, Web Parts that appear on provisioned pages, and the document libraries and lists that are activated upon site creation. This templating architecture, first introduced by SharePoint Team Services and carried forward to SharePoint Portal Server 2003, still plays an important role in SharePoint Server 2007. The structure and content of site definitions have been considerably optimized in SharePoint Server 2007, making them much easier to work with. This optimization is due, in large part, to the introduction of Features, which have replaced the more functional aspects, such as defining the structure of document libraries and lists. We'll be covering Features in more detail in Chapter 16, Features. The important point here is that site definitions *still* exist and *still* play an important role with respect to customization; therefore, if you want to customize your deployment of SharePoint Server 2007 you need to have an understanding of the underlying templating architecture. In this chapter, we examine the templating aspects of SharePoint, such as site definitions and site templates, and show you how to create your own custom site definitions and templates. At the end of the chapter, we revisit the topic of branding and walk you through another branding exercise that leverages site definitions.

15.1 Defining Site Definitions, Site Templates, and Custom Templates

The terminology that Microsoft has used for their templating architecture remains a source of confusion for many people. Perusing the various Microsoft help files and MSDN and TechNet articles, you will find many inconsistencies, especially regarding site definitions and site templates. The two terminologies seem to be interchangeable in some texts and aspects of the product, highlight-

ing the fact that even the product team finds the whole topic confusing. Let's take a brief look at some examples of these inconsistencies.

The WSS 3.0 SDK describes a site definition using the following text:

"*A site definition provides the base blueprint for a site, what lists it features, the default navigation structure, etc. A site definition involves a set of XML, ASPX, and other files on the front-end Web server.*"

The same page in the SDK then goes on to describe a site template using the following text:

"*A site template is a package that contains a set of differences and changes from a base site definition. This package is stored as an .stp file, which is a CAB file that contains XML files that define the differences. The .stp file can be downloaded and uploaded to site collections by users with appropriate rights, meaning that .stp files have some measure of portability.*"

So a quick translation of the above is that a site definition stores the server-based files used to instantiate a site. Once a site is instantiated from a site definition, you can customize that site further by adding Web Parts, lists, and so on, and then, using the browser interface, you can save your changes to a site template file which you can use to instantiate additional sites.

So what's wrong with these definitions? They seem perfectly fine on the surface—it's only when you actually start to work with the product that things start to get confusing. The biggest offender is the fact that the folder in which *all* site definitions reside is named SiteTemplates but it doesn't contain a single stp file! Also, when you start working with Features, you'll find that you can associate a Feature with a specific configuration defined within a site definition, yet Microsoft refer to this association as the Feature/Site Template Association and the actual element name used in the XML file is `FeatureSiteTemplateAssociation`. Another example from the SDK is the description of the `SiteProvision` method. One of the parameters of this method is `Name="SiteTemplate"`, and the description provided in the SDK for this parameter is as follows:

"*Specifies the site definition to use to instantiate the site. If not specified, by default Windows SharePoint Services creates the site by using the stan-*

dard Team Site definition. The following table shows the values for the default site definitions included in an installation of Windows SharePoint Services…."

Notice that you don't see the words *site template* used anywhere in the description for the `SiteTemplate` parameter, yet *site definition* is prevalent.

As you can see the terminology and the usage in supporting documentation is inconsistent and can prove frustrating at times. In an effort to remove some of the ambiguity surrounding these topics, the following section provides a brief explanation of our understanding of the terminologies and how they are connected.

15.1.1 Site Definitions

As previously mentioned, a site definition is simply a set of files that define the composition of a site in SharePoint from which other sites are created. The site definition files for SharePoint 2007 reside in the `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\SiteTemplates` directory, with each subfolder in the SiteTemplates directory representing a single site definition. A site definition describes the components, such as the navigational structure, provisioned pages, and activated Features, that comprise a site. The site definition files and folders reside on each front-end Web server in your SharePoint deployment; any additional custom site definitions must be installed on all front-end Web servers before they can be used. SharePoint Portal Server 2003 site definitions were language-specific, so you had to have a complete set of site definitions for each language represented in your SharePoint deployment; this was true even if the site definitions remained the same for each language. This burden has been removed from SharePoint Server 2007 and site definitions are no longer language-specific. This is immediately noticeable when examining the path to the SiteTemplates directory, as the path no longer contains the locale identifier.

Table 15.1 lists the site definitions created by default during the installation of SharePoint Server 2007. The site definitions tagged as upgraded are 2007 versions of the definitions used in the previous version of SharePoint, most of which are installed to aid the upgrade process from a custom SharePoint Portal Server 2003 site definition to a new SharePoint Server 2007 site definition. SharePoint Portal Server 2003 site definition folders required a prefix of SPS; this limitation no longer applies to SharePoint Server 2007 site definitions.

Table 15.1 *SharePoint Server 2007 Site Definitions*

Site Definition	Description
GLOBAL	Special type of site definition located in `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\GLOBAL` directory containing the common elements for all provisioned sites in SharePoint, such as the master page gallery and base types.
BDR	Site definition used for creating Document Center sites to centrally manage documents within your enterprise.
BLANKINTERNETCONTAINER	Special type of site definition known as a portal site definition or a Virtual Site definition. Site definition contains a manifest file that defines the structure of a site deriving from multiple site definitions. The xml file that contains the portal site definition is located in the TEMPLATE\XML\InternetBlank.xml. Defines a site with a starter site hierarchy for an Internet-facing site or a large intranet portal. Includes a home page, a sample press releases subsite, a Search Center, and a login page.
BLANKINTERNET	Site definition used to create a site for publishing Web pages, with workflow functionality enabled. The associated Site Template registration for this site definition is hidden and therefore does not appear as a selection to the user when creating sites.
Blog	Base Windows SharePoint Services blog site definition, used when creating Blog sites.
CENTRALADMIN	Site definition used for central administration site. The associated site template registration for this site definition is hidden and therefore does not appear as a selection to the user when creating sites.
MPS	Base Windows SharePoint Services Meeting Workspace site definition. Contains multiple configurations including: Basic Meeting Workspace, Blank Meeting Workspace, Decision Meeting Workspace, Social Meeting Workspace, Multipage Meeting Workspace.
offile	Site definition used to create sites designed for records management.
OSRV	Site definition used for the shared services administration site. The associated site template registration for this site definition is hidden and therefore does not appear as a selection to the user when creating sites.
PROFILES	Site definition used when creating a profile site.
PUBLISHING	Site definition used when creating a blank publishing site. The matching site template registration is named “CMSPUBLISHING” in the WebTemp.xml file.

Table 15.1 *SharePoint Server 2007 Site Definitions (continued)*

Site Definition	Description
SPS	UPGRADED: Site definition used when creating the top-level home site for the collaboration portal site definition. The associated site template registration for this site definition is hidden and therefore does not appear as a selection to the user when creating sites.
SPSCOMMU	UPGRADED: SharePoint Portal Server 2003 Community Area site definition. The associated site template registration for this site definition is hidden and therefore does not appear as a selection to the user when creating sites.
SPSMSITE	Site definition used for delivering personalized views, data, and navigation into a My Site.
SPSMSITEHOST	Site definition used for the site used to host personal sites (My Sites) and the public People Profile page. This site should only be provisioned once per shared service provider. If a My Site Host site is not created within the SharePoint farm then public profile pages will not be available to users.
SPSNEWS	UPGRADED: SharePoint Portal Server 2003 site definition for News Area. The associated Site Template registration for this site definition is hidden and therefore does not appear as a selection to the user when creating sites.
SPSNHOME	Site definition used for creating New sites for publishing new articles and links to news articles.
SPSPERS	Site definition used for creating a personal space for a SharePoint user. The associated Site Template registration for this site definition is hidden and therefore does not appear as a selection to the user when creating sites.
SPSREPORTCENTER	Site definition for creating a site for creating, managing, and presenting Web pages that display dashboards and key performance indicators to communicate business intelligence information.
SPSSITES	Site definition for creating a site that lists and categorizes sites within your deployment.
SPSTOC	UPGRADED: SharePoint Portal Server 2003 site definition for Contents Area. The associated site template registration for this site definition is hidden and therefore does not appear as a selection to the user when creating sites.

Table 15.1 *SharePoint Server 2007 Site Definitions (continued)*

Site Definition	Description
SPSTOPIC	UPGRADED: SharePoint Portal Server 2003 site definition for Topics Area. The associated Site Template registration for this site definition is hidden and therefore does not appear as a selection to the user when creating sites.
SRCHCEN	Site definition used to create a Search Center site with tabs. Two default tabs are created: one for general searches and one for information about people.
SRCHCENTERLITE	Site definition used to create a site containing pages to display search results and advanced searches.
Sts	Base Windows SharePoint Services Team Site definition. Consists of multiple configurations including: Team Site, Blank Site, and Document Workspace.
WebManifest /SPSPORTAL	Special type of site definition known as a portal site definition or a virtual site definition. Site definition contains a manifest file that defines the structure of a site deriving from multiple site definitions. Defines a site with a starter site hierarchy for an intranet divisional portal. Includes a home page, a News site, a Site Directory, a Document Center, and a Search Center with tabs.
Wiki	Base Windows SharePoint Services Wiki site definition used when creating Wiki sites.

The first site definition listed in Table 15.1 is the GLOBAL site definition, considered as the parent site definition for all sites and contain the definitions of elements common to all sites. The GLOBAL site definition resides in the `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\GLOBAL` directory, as opposed to the SiteTemplates directory, in which all other site definitions reside. If you examine the content of the GLOBAL site definition folder, you will see that it contains some master pages and several schema files. The schema files defined in the GLOBAL site definition include ONET.XML, STDVIEW.XML, and VWSTYLES.XML. STDVIEW.XML defines the base view from which new views are created, and VWSTYLES.XML defines the base styles for viewing list data available on the Create View page for a list. Modifying the contents of STDVIEW.XML is not recommended, and doing so may potentially cause your sites to malfunction. All site definitions contain an ONET.XML schema file. The ONET.XML schema file that resides in the GLOBAL site definition is referred to as the global ONET.XML file, and it defines list templates for hidden lists, list base types, configurations, and modules that apply globally to a SharePoint deployment. The ONET.XML file that appears in

each site definition goes on to define the navigational structure, document templates, configurations, modules, and various components specific to a particular type of site. A site definition's ONET.XML can describe multiple configurations of the elements defined. For example, the STS site definition, in conjunction with the GLOBAL site definition, defines all the lists, views, document templates, Web Parts, navigational elements, and Features required to create a team site, a blank site, and a document workspace site. Each site type has a matching `Configuration` element specified in the ONET.XML file that tells SharePoint which lists and features that specific type of site needs upon site creation. We delve into more detail on the inner workings of ONET.XML later in the chapter, but the `Configuration` element is important to highlight at this stage, as it is essential to the understanding of site templates. The `Configuration` element has an `ID` attribute that is used as a reference for other XML files, such as those used when defining features and the WEBTEMP*.XML files used to register site definitions as a site template.

15.1.2 Site Templates

A site template references a specific configuration within a given site definition. For example, the team site and document workspace site templates both inherit from the STS site definition, but they each instantiate different lists and Features upon site creation. Site templates are defined, or registered, in a WEBTEMP*.XML file and, once registered, they appear in the list of available site templates in the browser interface from which the user can choose when creating a site collection or site.

Each front-end Web server in a SharePoint Server 2007 deployment has several WEBTEMP*.XML files containing the template registrations. WEBTEMP.XML is the main WEBTEMP*.XML file installed with Windows SharePoint Services, registering templates such as the team site and the meeting workspace templates. Subsequent template registrations installed by SharePoint Server 2007 are represented with a suffix. For example, the file that contains the majority of the SharePoint Server 2007–specific template registrations is named WEBTEMPSPS.XML. Figure 15.1 shows the relationship between the STS site definition and the registration of three site templates associated with that site definition. Figure 15.2 shows a snapshot of content from the WEBTEMP.XML file highlighting the registration of the STS site definition configurations. Notice that each configuration registration is listed as an available site template that appears in the browser user interface. So a site template is simply the registration of a specific configuration within a given site definition.

Table 15.2 lists all the site templates registered by SharePoint Server 2007. The table shows the site definition and its configuration registrations in the associated WEBTEMP*.XML file. Note that the site definitions are

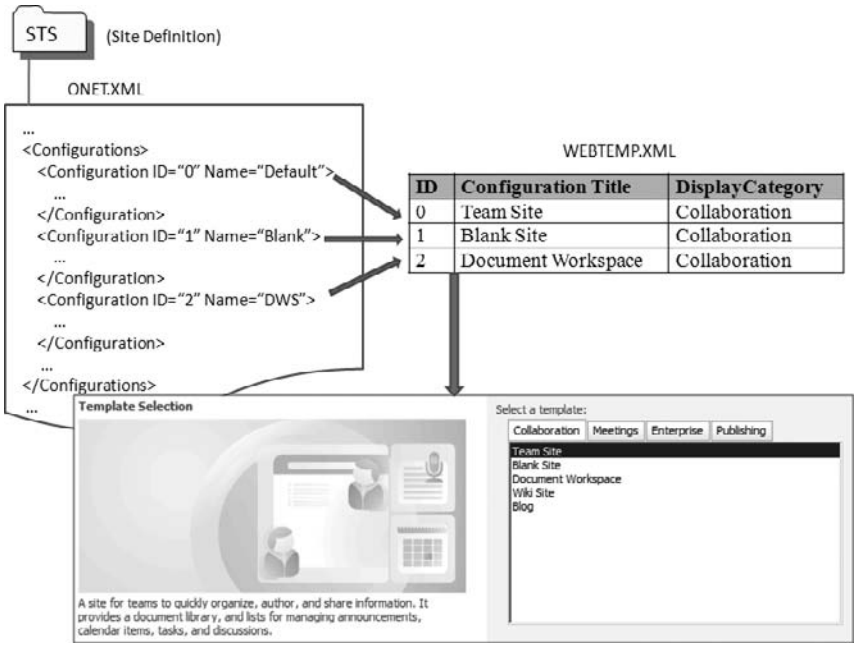


Figure 15.1 *STS site definition and registered configurations/site templates.*

```
- <Template Name="STS" ID="1">
    <Configuration ID="0" Title="Team Site" Hidden="FALSE" ImageUrl="/_layouts/images/stsprev.png" Description="A site
      for teams to quickly organize, author, and share information. It provides a document library, and lists for
      managing announcements, calendar items, tasks, and discussions." DisplayCategory="Collaboration" />
    <Configuration ID="1" Title="Blank Site" Hidden="FALSE" ImageUrl="/_layouts/images/blankprev.png" Description="A
      blank site for you to customize based on your requirements." DisplayCategory="Collaboration"
      AllowGlobalFeatureAssociations="False" />
    <Configuration ID="2" Title="Document Workspace" Hidden="FALSE" ImageUrl="/_layouts/images/dwsprev.png"
      Description="A site for colleagues to work together on a document. It provides a document library for storing the
      primary document and supporting files, a tasks list for assigning to-do items, and a links list for resources related
      to the document." DisplayCategory="Collaboration" />
  </Template>
```

Figure 15.2 *STS template registration in WEBTEMP.XML.*

assigned a Template ID in WEBTEMP*.XML, which uniquely identifies the registration within the SharePoint deployment.

Our definition of a site template differs somewhat from the WSS 3.0 SDK description mentioned earlier in the chapter, and although we have described site templates, we have not yet mentioned .stp files. The reason for this is that .stp files more appropriately fall into the category of *custom* site templates.

Table 15.2 *Site Template Registrations*

Site Definition/Folder	ID	Registered Site Templates/Configurations		
STS	1	WEBTEMP.XML		
		ID	**Title**	**Hidden**
		0	Team Site	FALSE
		1	Blank Site	FALSE
		2	Document Workspace	FALSE
MPS	2	WEBTEMP.XML		
		ID	**Title**	**Hidden**
		0	Basic Meeting Workspace	FALSE
		1	Blank Meeting Workspace	FALSE
		2	Decision Meeting Workspace	FALSE
		3	Social Meeting Workspace	FALSE
		4	Multipage Meeting Workspace	FALSE
CENTRALADMIN	3	WEBTEMP.XML		
		ID	**Title**	**Hidden**
		0	Central Admin Site	TRUE
Wiki	4	WEBTEMP.XML		
		ID	**Title**	**Hidden**
		0	Wiki Site	FALSE
BDR	7	Webtempbdr.en-US.xml		
		ID	**Title**	**Hidden**
		0	Document Center	FALSE
Blog	9	WEBTEMP.XML		
		ID	**Title**	**Hidden**
		0	Blog	FALSE

Table 15.2 *Site Template Registrations (continued)*

Site Definition/Folder	ID	Registered Site Templates/Configurations		
SPS	20	Webtempsps.xml Also referenced in the portal site definition for the collaboration portal site template.		
		ID	**Title**	**Hidden**
		0	SharePoint Portal Server Site	TRUE
SPSPERS	21	OBSOLETE Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	SharePoint Portal Server Personal Space	TRUE
SPSMSITE	22	OBSOLETE Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Personalization Site	TRUE
SPSTOC	30	OBSOLETE Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Contents Area Template	TRUE
SPSTOPIC	31	OBSOLETE Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Topic Area Template	TRUE
SPSNEWS	32	OBSOLETE Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	News Site	TRUE
SPSNHOME	33	Webtempsps.xml		

Table 15.2 *Site Template Registrations (continued)*

Site Definition/Folder	ID	Registered Site Templates/Configurations		
		ID	**Title**	**Hidden**
		0	News Site	FALSE
SPSSITES	34	Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Site Directory	FALSE
SPSCOMMU	36	OBSOLETE Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Community area Template	FALSE
SPSREPORT-CENTER	38	Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Report Center	FALSE
PUBLISHING/CMS-PUBLISHING	39	Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Report Center	FALSE
		SetupPath="SiteTemplates\PUBLISHING"		
OSRV	40	Webtemposrv.xml		
		ID	**Title**	**Hidden**
		0	Shared Services Adminis-tration	TRUE
WebManifest/ SPSPORTAL	47	Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Collaboration Portal	FALSE
		ProvisionData="SiteTemplates\\WebManifest\\ PortalWebManifest.xml"		
SRCHCEN	50	Webtempsps.xml		

Table 15.2 *Site Template Registrations (continued)*

Site Definition/Folder	ID	Registered Site Templates/Configurations		
		ID	**Title**	**Hidden**
		0	Search Center with Tabs	FALSE
PROFILES	51	Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Profiles	TRUE
BLANKINTERNET-CONTAINER	52	Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Publishing Portal	FALSE
		ProvisionData=”xml\\InternetBlank.xml”		
BLANKINTERNET	53	Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	Publishing Site	TRUE
SPSMSITEHOST	54	Webtempsps.xml		
		ID	**Title**	**Hidden**
		0	My Site Host	FALSE
SRCHCENTERLITE	90	Webtempsrch.xml		
		ID	**Title**	**Hidden**
		0	Search Center	FALSE
		1	Search Center	TRUE
offile	14483	Webtempoffile.xml		
		ID	**Title**	**Hidden**
		0	Records Center	TRUE
		1	Records Center	FALSE

15.1.3 Custom Site Definitions and Custom Site Templates

Out of the box, SharePoint Server 2007 provides a host of templates from which to choose when creating your sites; however, there may be occasions when these templates do not meet your needs. SharePoint provides a number of supported mechanisms for creating your own custom site definitions and site templates. Before we proceed, it is important to stress that you should *not* directly modify any of the out-of-the-box site definition and WEBTEMP*.XML files, doing so parks you firmly in unsupported territory. There is one exception for customizing a WEBTEMP*.XML file, and that is to simply hide one of the default templates from appearing in the list of available templates in the browser user interface. In this instance, it is safe to edit the WEBTEMP*.XML files to set the Hidden attribute on the relevant configuration to TRUE.

To create a custom site definition, you simply make a copy of an existing site definition that you would like to customize, give the site definition folder a unique name, and then customize the site definition files and feature associations as necessary. For example you can change the navigational structure, or identify a specific set of Features to be activated upon site creation. Once you have created your custom site definition, you need to register your custom templates. To register your custom site definition and its associated configurations as custom site templates, copy the WEBTEMP.XML file, give the new file a unique name by appending a string to the name of the original file, (e.g., WEBTEMPSPYSRUS.XML), and customize the contents of the file to reflect your own registrations. Figure 15.3 shows the registration for a custom site definition (spysrus) and its associated configuration. The `Name` attribute of the `Template` element should match the folder name of your new site definition, and, to avoid conflict with Microsoft's templates, the `ID` attribute should use unique values greater than 10,000. However, there are a few Microsoft template registrations that fall into this range, such as the offile template. The `Configuration` element uses the `ID` attribute to map to a specific configuration in the site definition. The `Title` attribute defines the name of the site template as it should appear in the list of available site templates in the browser user interface, and the `DisplayCategory` attribute determines which tab the site template appears under in the template selection section of the browser user interface. You can add your own custom categories and tabs by simply assigning your custom name to the `DisplayCategory` attribute.

15.1.4 Feature Stapling

It is important to note that not all the functionality provided by a site definition is defined in the actual definition files. For example, the files that appear

Figure 15.3
Custom site definition and template registration.

in the master page gallery and the Web Part gallery for site collections are actually provisioned using features. These Features are not activated directly in the site definition files; instead, they are associated with the site definitions using a technique known as stapling. Chapter 16, Features covers Feature stapling in more detail, but for now all you need to know is that there are several files in the Features directory on the front-end Web server that contain the association information. To leverage the same functionality in your custom site definition, you need to either add an association to the existing staple files or, preferably, create your own feature staple in which to store all your associations. Figure 15.4 shows the folder structure of two top-level site collections edited using SharePoint Designer 2007. Site Collection A was created using the out-of-the-box Team Site site template that uses STS as its base site definition, and Site Collection B was created from a custom site template that uses a custom version of the STS site definition as its base site definition. Notice that Site Collection B is missing lots of content in both the master page gallery and the Web Part gallery, which means that we are missing some feature associations with the custom site definition.

So how do we identify which Features we need to staple to our custom site definition? You could navigate to the Features directory, located at `\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATES\FEATURE` and search for files that contain the string "stapl" in their name and contain the site definition configuration reference (e.g. STS#0), which returns all the feature stapling files associations that ship with SharePoint Server 2007 (Figure 15.5).

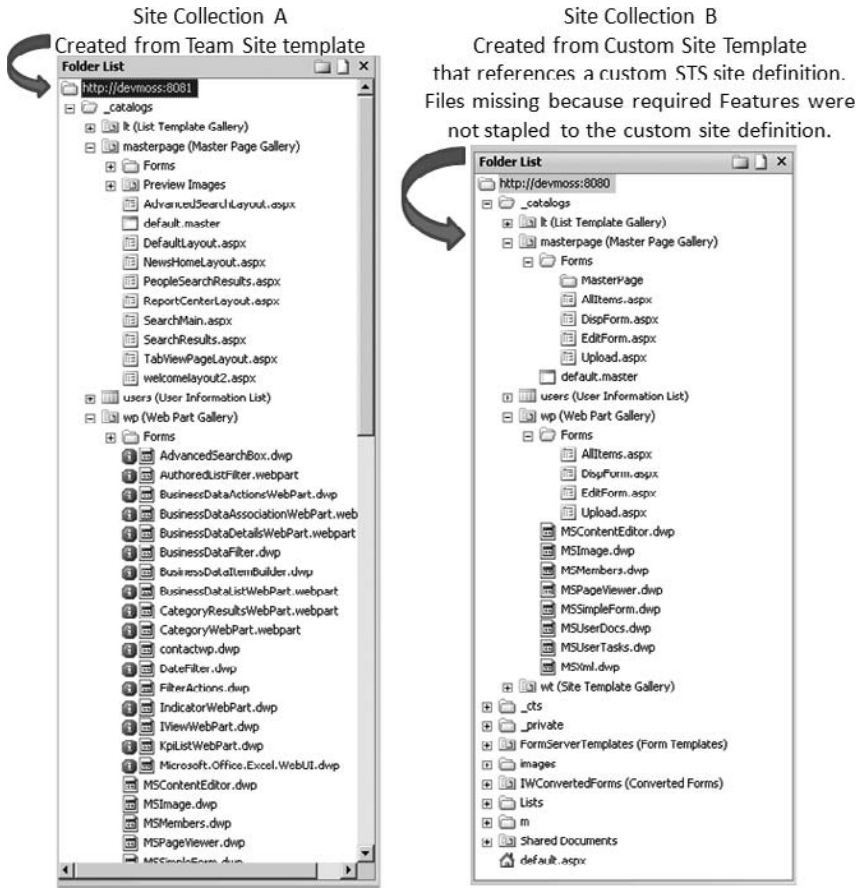


Figure 15.4 *Custom Site Definition without required Feature Associations.*

With each staple file returned, either add the references to the configurations in your own custom site definition or copy the references to a new XML file and replace the site definition configuration references with your own site definition configuration references. Figure 15.6 shows an example of a new staple Feature folder that contains two XML files. The feature.xml file defines the Feature and provides a pointer to the element manifest file spysrussitstapling.xml that identifies the Features to staple to specific site definition configurations. The `Id` attribute in the `Feature` element of the feature.XML file must be a unique GUID to identify the Feature. You can create a unique identifier by using the GUID generator tool (guidgen.exe) provided with Visual Studio. The `Id` attribute in the `FeatureSiteTemplateAssociation` element of the element manifest file must map to an the ID of an existing Feature, and the `TemplateName` attribute is used to specify the site definition configuration (site template) to which the Feature will be stapled. If you create your own Feature staple, you

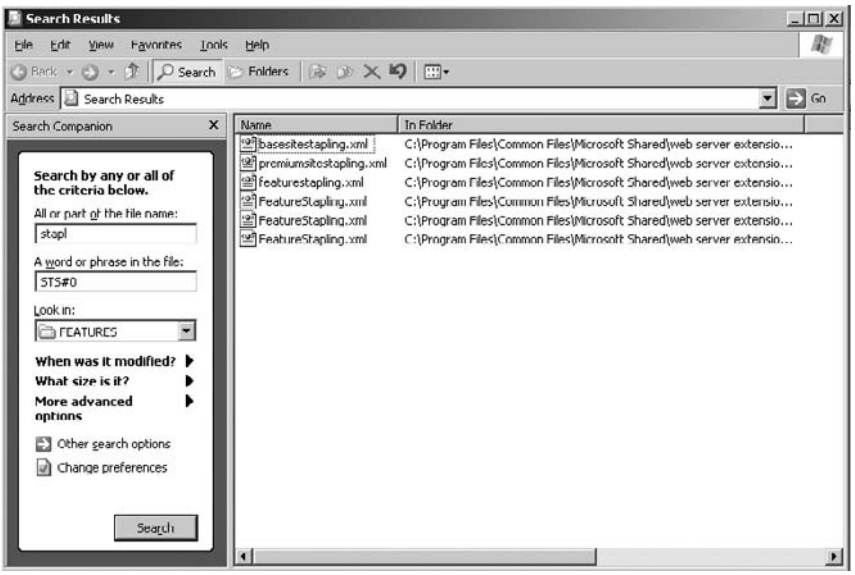


Figure 15.5
Searching for STS#0 feature associations.

Figure 15.6
SpysRUs Feature Staple

need to install the Feature for it to take effect. Figure 15.7 shows the installation command necessary to both install and activate this Feature. The Feature is automatically activated with the `installfeature` command, since its scope is set to the farm level.

Figure 15.7
Command line to install a Feature

```
stsadm.exe -o installfeature -name featurefoldername
```

At this point, we have discussed how to create a custom site definition, register the configurations as site templates in WEBTEMP*.XML, and staple the required Features. A quick reset of IIS and you should be able to create sites and site collections from your newly registered custom site template.

15.1.5 .STP Files

If you want to make some adjustments to the template of a site, but you don't want to create a complete custom site definition, then you have the option to create a site using your preferred template; make some changes to the site through the browser user interface, such as adding Web Parts, lists, and document libraries; and then save these changes to a custom site template file (.stp). The custom site template file (.stp) is a compressed cabinet (CAB) file containing a set of files that describe the differences between the site and its base site definition; it has a file size limit of 10MB. This file is then saved to the site template gallery, from which you can edit the template properties. Once an .stp file appears in the site template gallery, it is listed as an available template in the Template Selection section when creating a new site or site collection. The site template gallery is site-specific, so the .stp file must be uploaded to the gallery of the site from which you want to create additional sites. To save a site as a site template, navigate to the Site Settings page and from the Look and Feel menu click on the Save site as template link.

Note that if you have the publishing feature activated on a site, you cannot use the Save site as template functionality—in fact, the link is no longer accessible through the browser user interface. You can try and bypass this limitation by navigating directly to the Save site as template URL; however, the fact that the option disappears when the publishing feature is activated indicates that problems may arise when saving templates in this manner. If you choose to proceed down this route, you may find yourself in unsupported territory. SharePoint Designer does *not* tailor access to its menu option to save a site as a template based on whether the publishing feature is enabled, effectively allowing you to bypass this restriction. To save a site as a site template using SharePoint Designer, from the File menu click Export

and then click SharePoint Site Template. This will bring you to the Save site as template page in the browser user interface, allowing you to save the site to an .stp file that is then published to the site template gallery.

15.1.6 Portal Site Definitions (Also Known as Virtual Site Definitions)

Office SharePoint Server 2007 brings a new type of site definition to the table, one that provides a great deal more flexibility with respect to creating custom a Web site. With SharePoint Server 2007, a site definition can inherit one or more site definitions. So what does this gain us? While Windows SharePoint Services allows you to create a site definition that maps to a single SharePoint site, SharePoint Server 2007 extends this functionality by allowing you to create a portal site definition file that comprises the hierarchy of site definitions to be used upon site creation. The Collaboration Portal site template is based on a portal site definition (Figure 15.8). If you examine the contents of WEBTEMPSPS.XML, you will see that the Collaboration Portal site template contains a reference to the manifest file that contains the portal site definition used to provision the Web site. The `ProvisionData` attribute defines the location of the provision date, which in this case happens to be the portal site definition file PortalWebManifest.xml. PortalWebManifest.xml defines the hierarchy, which in this case defines a site based on the SPS site definition at the top of the hierarchy, followed by four subsites based on the SPSNHOME, SPSSITES, SRCHCEN, and BDR site definitions. If you open a site collection created using the Collaboration Portal site template, you will notice that the structure is reflected in the collaboration portal.

15.1.7 Application Templates

Application templates refer to a set of files that comprise custom site definitions and custom site templates in addition to special functionality, such as a specific workflow, content types, and Features. At the time of writing, Microsoft provides 40 Windows SharePoint Services 3.0 application templates, referred to as The Fabulous 40 (see the Microsoft Web site for download details). These templates contain a number of SharePoint solution files (.wsp files) and a number of site templates. The solution files package the site definitions and custom functionality into a single file, making it easy to deploy to your SharePoint farm. Also available as a downloadable package are four master pages that complement the application templates. Figure 15.9 shows the knowledge base template in action with the block red master page applied. The great thing about the application templates is that you can use them as the building blocks for creating your own custom solutions.

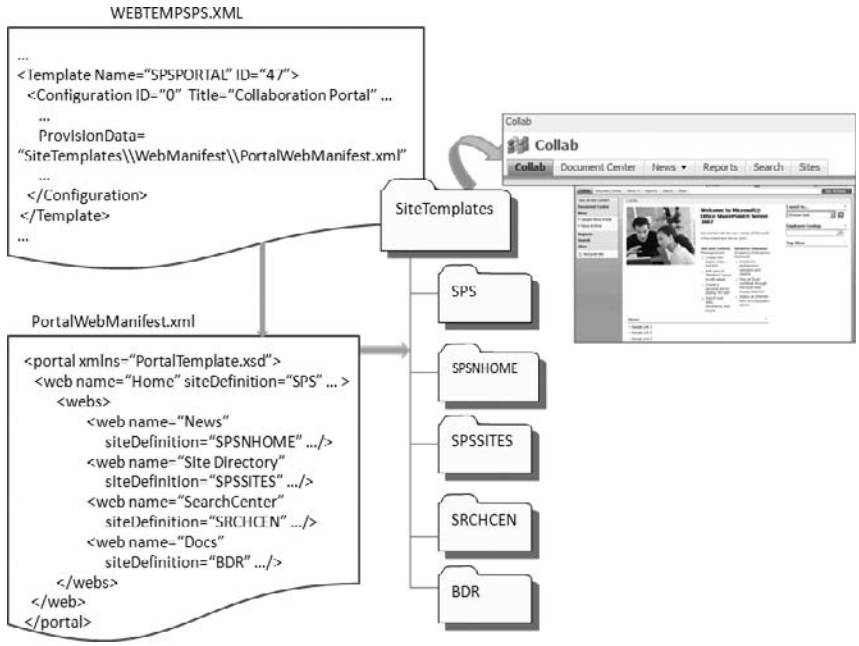


Figure 15.8 *Collaboration portal site definition.*

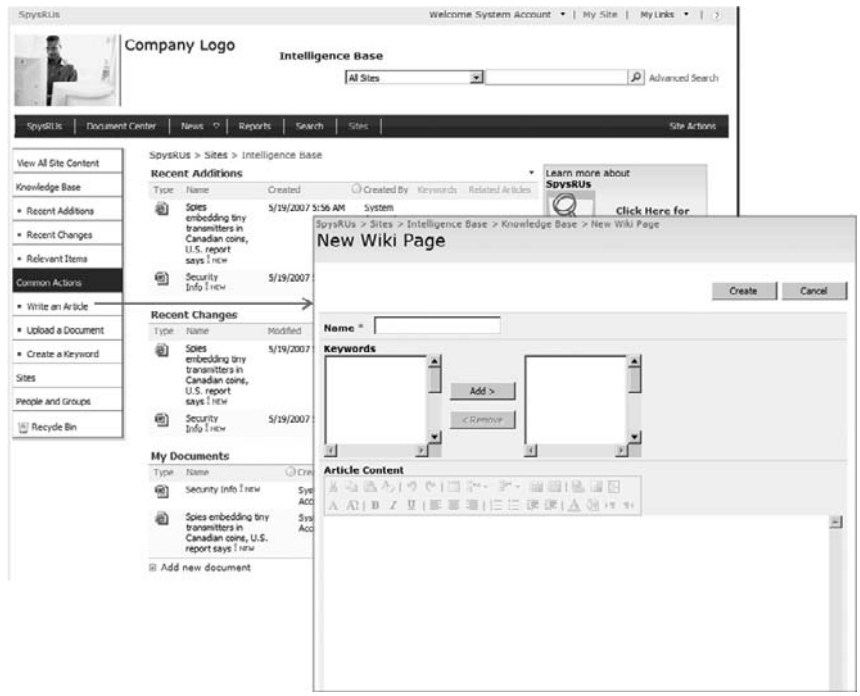


Figure 15.9 *Knowledge Base Application template in action.*

15.2 Dissecting ONET.XML

As previously mentioned, SharePoint extracts the site structure and layout information for a site from a set of XML files contained within the site definition folders, the most prominent of which is ONET.XML. If you examine the content of the ONET.XML file in each of the default site definitions, you will notice that it is considerably smaller than its SharePoint Portal Server 2003 version. This is due to the fact that the definition settings that were common across all site definitions have been rolled up into the GLOBAL site definition, and the common document libraries and lists, which were once defined in each site definition, are now implemented through the use of Features. That aside, ONET.XML is the most important file when dealing with site definitions, and it comprises many useful elements. Figure 15.10 shows the top-level elements in the XML file. Let's take a look at these elements and find out a bit more on their actual purpose with respect to site generation.

Figure 15.10 *ONET.XML structure.*

15.2.1 The Project Element

The Project element is the top-level element of the ONET.XML file and has a number of associated attributes that can be set to control various aspects of the site. For example, you can set the `DisableWebDesignFeatures` attribute to prevent certain Features' being editable from SharePoint Designer 2007. The Windows SharePoint Services SDK provides a description of the supported attributes and their usage. Of particular interest to the branding is the `AlternateCSS` attribute. The `AlternateCSS` attribute basically allows you to define an alternate cascading style sheet to override the default Microsoft

style sheet. Any sites created from a site definition with this attribute populated will automatically use the style sheet specified. Store the custom style sheet in the `\\Microsoft Shared\Web Server Extensions\12\TEMPLATE\LAYOUTS\LCID\STYLES` directory, and you can reference the path in the ONET.XML using `/_layouts/[%=System.Threading.Thread.CurrentThread.CurrentUICulture.LCID%]/styles/`, which dynamically resolves the locale used by the system. In Chapter 14, Branding your SharePoint 2007 Sites, we mentioned that the alternate CSS URL setting can be controlled by setting the `AlternateCSS` attribute in a site definition, by populating the AlternateCSSUrl property of the SPWeb object through code, or by populating the alternate CSS URL setting through the browser user interface. When a site is created from a site definition that has the `AlternateCSS` attribute set to a custom style sheet, its alternate CSS URL setting in the browser user interface will automatically map to this new style sheet.

The `AlternateHeader` attribute allows you to replace the top horizontal banner area with your own custom header. This attribute existed in SharePoint Portal Server 2003 and offered a glimmer of hope to Web designers in their crusade to achieve a consistent look and feel across all Web pages within their portal implementation. Unfortunately, it didn't really live up to its potential, because not all pages in SharePoint actually referenced the `AlternateHeader` attribute. While the `AlternateHeader` attribute is still available with populating site definitions, it has really taken a back seat to master pages with respect to providing a consistent branding experience. If you want to implement a consistent look and feel throughout your Web sites, we recommend that you go the master page route. See Chapter 14, Branding your SharePoint 2007 Sites, for in-depth information on master pages.

15.2.2 NavBars

The `NavBar` element, as one would expect, controls the navigation bars that appear on a SharePoint home page and comprises one or more `NavBar` elements. The navigation bar that appears at the top of the page is defined first by the `NavBar` element, which is named the SharePoint Top Navbar. This navigation bar may contain `NavBarLink` elements that define hyperlinks to add to the top navigation area. All subsequent NavBar elements define the quick launch area of the page—in other words, the vertical navigation area that appears on the left of a Web page. With SharePoint Portal Server 2003, the horizontal and vertical navigational structures that appeared on the portal area pages were controlled by specific Web controls, and, as such, customizations to the `NavBar` elements in ONET.XML did not affect portal area pages. This has been corrected in SharePoint Server 2007, and now the `NavBar` elements affect pages in both a SharePoint Server 2007 site and a Windows SharePoint Services 3.0 site.

Adding an additional menu item to the navigation bar of a *future* SharePoint site created using a custom site definition is straightforward; simply add a new `NavBarLink` element to the `NavBar` element of choice in the site definition's ONET.XML file, or create your own `NavBar` element. Figure 15.11 shows sample code that adds a set of custom application links to a new `NavBar` element on the quick launch area of a SharePoint team site. The figure also shows a new site that was created using this custom site definition. This method of adding custom navigation to a SharePoint site is quite rudimentary and inflexible with respect to extensibility. Any additional customizations applied to the site definition do not affect existing sites. If you want to create custom navigation that is extensible and allows for the generation of dynamic links, a better method would be to create a custom site map provider and implement it through a combination of Features and master pages. We examine this scenario in more detail in Chapter 16, Features.

Figure 15.11 *Creating new Navigation Links in the Quick Launch area.*

15.2.3 Configurations Element

We have mentioned the `Configurations` element a number of times already throughout the chapter when describing the relationship between site definitions and site templates, but we haven't gone into too much detail as to what the `Configurations` element contains. To recap, the `Configurations` element in ONET.XML is comprised of one or more `Configuration` elements that determine the content and functionality to be generated when a site is instantiated from a site definition. A `Configuration` element has a number of associated attributes that describe the configuration, a complete list of which is available in the WSS 3.0 SDK. The only required attribute is the `ID` attribute, which is integer based and uniquely identifies the configuration. Two attributes that are once again relevant to the brand-

ing scenario we have been following are `CustomMasterUrl` and `MasterUrl`, which define the master pages that sites created from the site definition will use for their publishing pages and system pages respectively. Figure 15.12 shows the attributes in action. In this particular example, both attributes are set to the same master page: corporate.master. Corporate.master is a master page file at the root of the site definition folder which is then provisioned using the `Modules` element defined later in the ONET.XML file. Figure 15.13 shows the master page gallery of a site that was created using the custom site definition. Notice that the page picks up the branding defined by the corporate.master master page, and also notice that the gallery contains the provisioned corporate.master file.

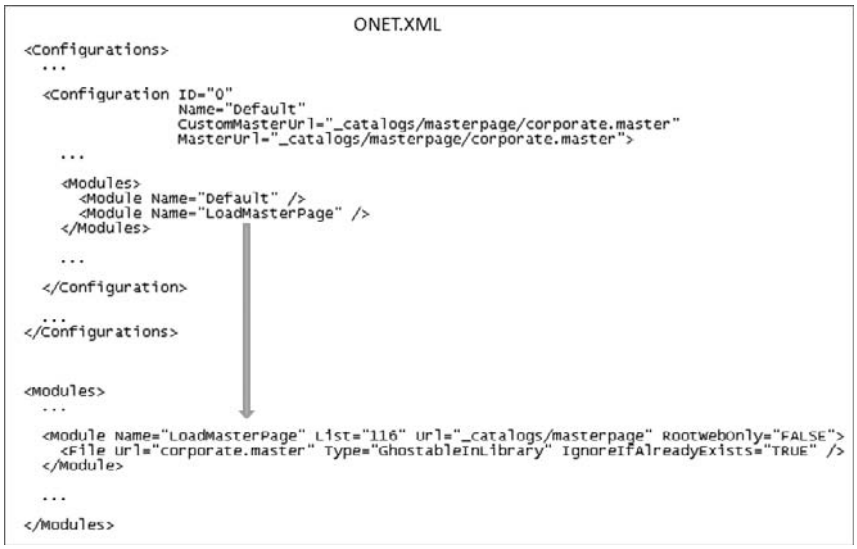


Figure 15.12
Setting the master pages during site creation.

The `Configuration` element contains a number of child elements, including: `ExecuteUrl`, `Lists`, `Modules`, `SiteFeatures`, and `WebFeatures`. The `ExecuteUrl` element allows you to specify the URL of a page that you would like to appear or to be executed immediately after a site has been created. This is a very useful mechanism if you want specific code to run after site creation. The `Lists` element identifies the collection of lists to be included in the site. If you have defined a List template directly in the ONET.XML file, you can use the `List` element in a specific configuration to tell SharePoint to instantiate that list type. However, a more common practice in SharePoint Server 2007 is to use Features to define list templates and then use the `FeatureID` attribute of the List element to provision a specific list type for a specific configuration. This is an especially useful method if you have a number of custom list types that are used in a number of custom

Figure 15.13 *Provisioned master page.*

site definitions; you no longer have to define the same list templates over and over again in each site definition.

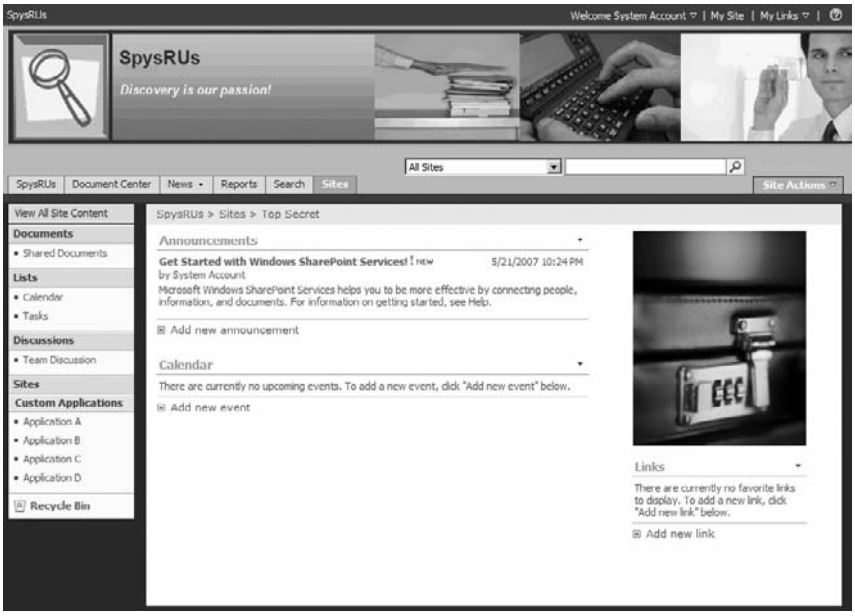
ONET.XML contains two `Modules` elements; the first is a child element of the `Configuration` element, and the second is a child element of the `Project` element. The individual modules that are specified in the `Configuration` section map to individual `Module` elements defined in the second `Modules` element, which is found toward the bottom of the ONET.XML file. In other words, the list of modules that appears in the `Configuration` element references the actual module definition defined in the second `Modules` element. Figure 15.12 shows an example of the ONET.XML containing both occurrences of the `Modules` element. If you examine the code, you will see that the first occurrence basically references two modules `Default` and `LoadMasterPage`. The figure then goes on to show the full definition for the `LoadMasterPage` module. In this instance, the `LoadMasterPage` module provisions a file into the master page gallery of the instantiated site using the `File` element. The `File` element also has a number of child elements worth special mention, namely the `AllUsersWebPart`, `NavBarPage`, and `View` elements. The `AllUsersWebPart` element allows you to specify the Web Parts that should appear on a specific aspx page and where they should appear. This is a particularly useful element, as it provides a mechanism for you to activate your own custom Web Parts upon site creation. Implementing a Web Part using the `AllUsersWebPart` element is fairly straightforward, regardless of whether you are implementing a custom Web Part or a default SharePoint Web Part.

Simply include the Web Part definition in a `CDATA` section of the `AllUsersWebPart` element (Figure 15.14). In this example, we simply replaced the regular Windows SharePoint Services image (homepage.gif), which appears with a default team site, with our own custom image file, brief.gif. Notice that the Web Part is defined to appear as the first Web Part in the right zone. Figure 15.15 shows the resulting Web site, which displays a picture of a briefcase instead of the typical Windows SharePoint Services logo shown in Figure 15.11.

Figure 15.14 *AllUsersWebPart element.*

```
<Modules>
  <Module Name="Default" Url="" Path="">
    <File Url="default.aspx" NavBarHome="True">
      ...
      <AllUsersWebPart WebPartZoneID="Right" WebPartOrder="1"><![CDATA[
                <WebPart xmlns="http://schemas.microsoft.com/WebPart/v2"
                         xmlns:iwp="http://schemas.microsoft.com/WebPart/v2/Image">
                    <Assembly>Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
                              PublicKeyToken=71e9bce111e9429c</Assembly>
                    <TypeName>Microsoft.SharePoint.WebPartPages.ImageWebPart</TypeName>
                    <FrameType>None</FrameType>
                    <Title>$Resources:wp_SiteImage;</Title>
                    <iwp:ImageLink>/_layouts/images/brief.gif</iwp:ImageLink>
                    <iwp:AlternativeText>$Resources:core,sitelogo_wss;</iwp:AlternativeText>
                </WebPart>
                ]]>
      </AllUsersWebPart>
      ...
    </File>
  </Module>
  ...
</Modules>
```

Figure 15.15 *Custom image added to default Web Part.*



The `NavBarPage` element allows you to declare a provisioned page as the site home page, and the `View` element allows you to specify the list views for a given page within a module. The `View` element also provides you with an

easy method to add list view Web Parts to a page; in fact, this is exactly how the announcements and calendar list view Web Parts are instantiated in Figure 15.15. Figure 15.16 shows the corresponding `View` element definitions. The `View` element has a large amount of associated attributes, all of which are well documented in the WSS 3.0 SDK. These attributes allow you to specify such things as how many row items to return in the View, and whether to display only the files of a specific folder.

Figure 15.16 *View element definition.*

```
<Modules>
  <Module Name="Default" Url="" Path="">
    <File Url="default.aspx" NavBarHome="True">
      <View List="$Resources:core,lists_Folder;/$Resources:core,announce_Folder;"
                BaseViewID="0" WebPartZoneID="Left" />
      <View List="$Resources:core,lists_Folder;/$Resources:core,calendar_Folder;"
                BaseViewID="0" RecurrenceRowset="TRUE"
                WebPartZoneID="Left" WebPartOrder="2" />
      ...
  </Module>
  ...
</Modules>
```

The `SiteFeatures` element identifies the site collection Features to activate in site collections created using the associated configuration. The `WebFeatures` element identifies the Web Features to activate in sites created using the associated configuration. Both elements contain a Feature child element that is used to individually specify each Feature that needs activation. All that is needed to activate a given Feature is the Feature ID. It is important to note that SharePoint activates Features listed in ONET.XML in the order in which they appear; if you have any Features that are depended upon by other Features then these need to be listed first.

15.3 Site Definition Solution Files

In Chapter 10, Planning the SharePoint Deployment we introduced the new solution deployment concept introduced with SharePoint Server 2007 and Windows SharePoint Services 3.0, which basically allows you to package all the components that comprise your specific customization into a single solution file. The solution file is a CAB file with a WSP extension, and it provides a convenient way to deploy site definitions across an entire SharePoint farm. This means that you no longer need visit each Web front-end server to copy your site definition files, and if you need to remove the site definitions from the farm, you can easily retract the definitions using the central administration browser interface. There are many files associated with a given site definition, especially when you start tracing the features stapled to the site, and we have seen that manually creating a site definition leaves a lot of room for errors. The good news is that Microsoft has provided a number of Visual Studio tools to automate this process as much as possible. The tools are collectively known as the Visual Studio 2005 extensions for Windows SharePoint

Services 3.0 and include project templates that provide the basis for a blank site definition and a site definition based on the Team Site configuration from the STS site definition. The great thing about using the Visual Studio tools and templates is that you can take advantage of the built-in IntelliSense support for CAML, making it much easier for you to avoid introducing errors when building your custom site definition. Once you have created your custom site definition, all you need to do is build the solution as you would any Visual Studio project and then select the menu option to deploy the solution. SharePoint then deploys your site definition to all the Web front end servers in the farm and generates the necessary WEBTEMP*.XML entries and Feature files. The site definition project templates include a SharePointProvision.cs file that allows you to insert custom code that you would like to be activated automatically upon site creation, and since it's in a Visual Studio project, you can take advantage of all the Visual Studio debug options for debugging your site definition. Figure 15.17 shows a solution file created using the Visual Studio Team Site site definition project template and its solution deployment information in the central administration browser user interface. Something to keep in mind is that the default setting for deploying SharePoint solutions from within Visual Studio is to deploy immediately. During the deployment process, an IISREST will be performed, which means SharePoint will be temporarily unavailable to your users; therefore, it is recommended to deploy solutions inside your regularly scheduled maintenance window. It is also important to note that the template ID assigned to your custom site definition configuration in the WEBTEMP*.XML files defaults to 1001 regardless of whether you have used this ID before. The ID is configurable by modifying the solution manifest.

In addition, the toolset provides a Solution Generator, which is basically a standalone program that you can use to generate a custom site definition from *any* instantiated site. This means that you can create a team site from the default Microsoft site template, apply all your customizations, and then use the Solution Generator to create a custom site definition from your customized site. Similar to the Visual Studio Team Site site definition project template, the Solution Generator creates all the necessary files required when deploying a solution in SharePoint, such as the solution manifest files and the site provisioning code. Figure 15.18 shows a list of the tasks performed by the Solution Generator when creating a custom site definition.

15.4 Site Definition Branding Exercise

In the previous chapter, our branding exercise showed how to create our own custom master page and style sheet and apply it to all the sites in our site collection. However, we still had the issue that newly created sites did not pick up our new branding. As we have seen throughout this chapter, site definitions provide a mechanism for provisioning the files and determining the

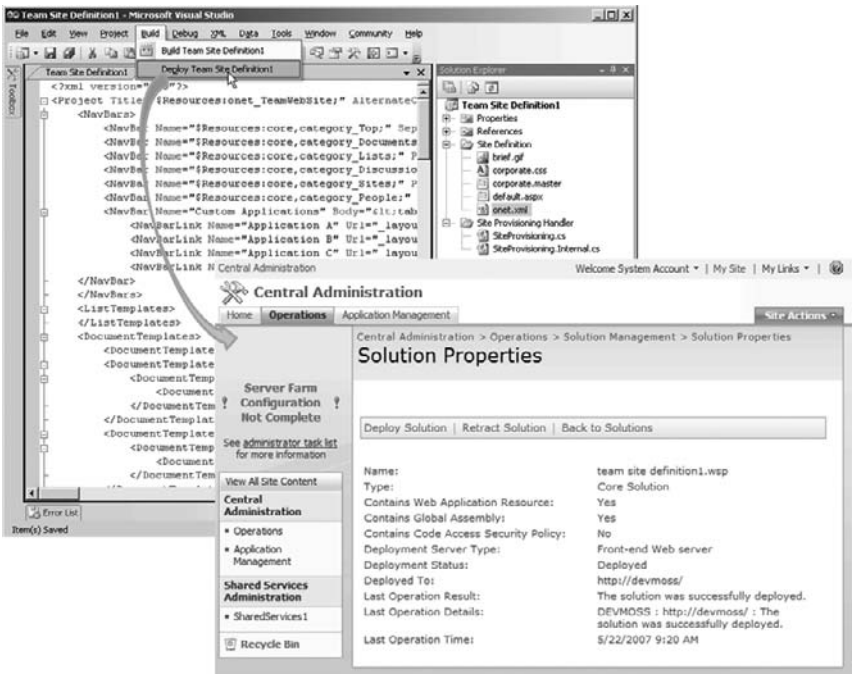


Figure 15.17
Custom site definition solution.

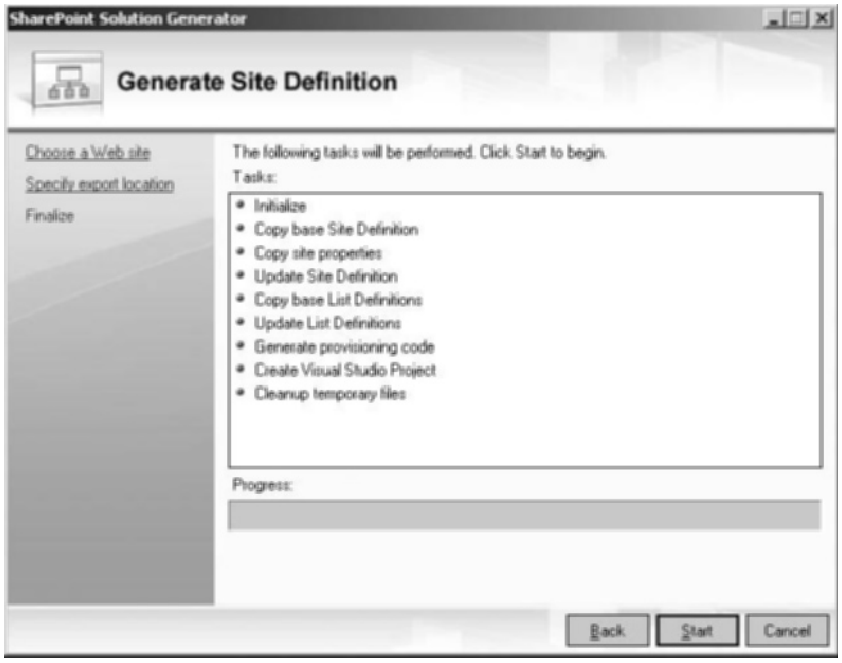


Figure 15.18
Solution Generator task list.

branding that a site adopts during site creation. The following exercise shows how to create a custom site definition using the Visual Studio site definition template. The custom site definition will set the alternate css URL, the master pages, and the home page image for the site.

To follow along with this exercise, you will need the following resources: a custom masterpage with which to brand your site, a cascading style sheet containing the styles you wish to implement, and an image file to replace the Windows SharePoint Services image that appears on a Team Site home page. Copy the .css file to your _layouts/LCID/styles directory. You will also need to download and install the Visual Studio 2005 extensions for Windows SharePoint Services 3.0; this package is accessible from the Microsoft Web site.

1. Launch Visual Studio 2005 and choose to create new project.
2. From the New Project dialog, select the SharePoint project type which is listed in the C# project types.
3. Select the Team Site Definition project template and give both your project and your solution a name—for example, CustomTeamSite (Figure 15.19). Visual Studio creates all the necessary files that comprise your site definition. In the underlying project folder in Windows Explorer, you will notice that a solution.xml file exists. This is basically the solution manifest file, which you are also free to customize to meet your own needs. The manifest settings are accessible through the Project Properties dialog.
4. Click OK.
5. From the Project menu, click CustomTeamSite Properties. The project properties page appears.
6. From the left menu, click SharePoint Solution. You may have to wait a few minutes while Visual Studio loads all of the appropriate settings from the solution manifest file (Figure 15.20).
7. Expand the Solution and SiteDefinition nodes, and select the Manifest to display the properties and their values.
8. Set the `TemplateID` to a unique ID within your SharePoint farm. If you have not already used the `TemplateID` of 1001, you can leave the default value.
9. Change the `Title` to reflect the title that you want to appear in the browser user interface as a template selection when creating a site.
10. Change the `DisplayCategory` to reflect the tab under which you want the site template to appear when selecting a template in the

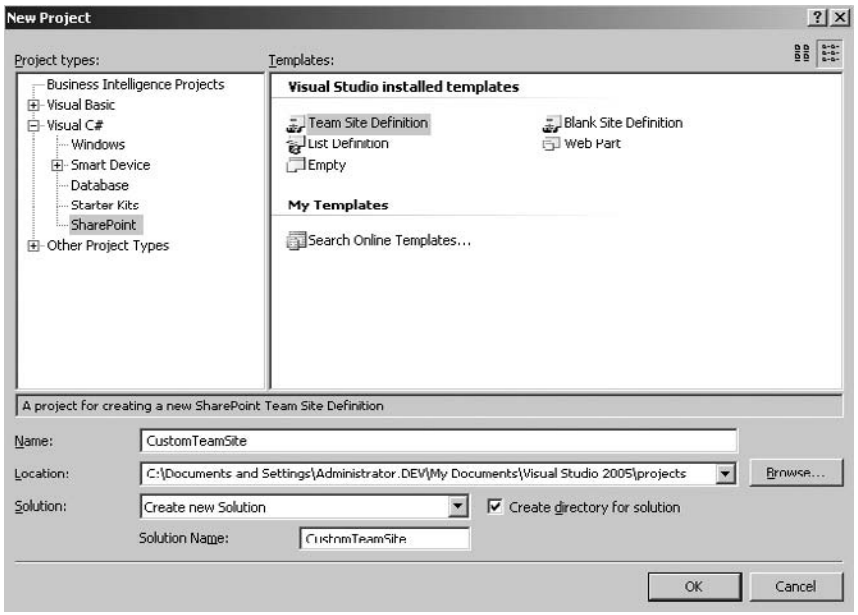


Figure 15.19
Creating a custom team site definition.

browser user interface. This completes the solution properties setting task.

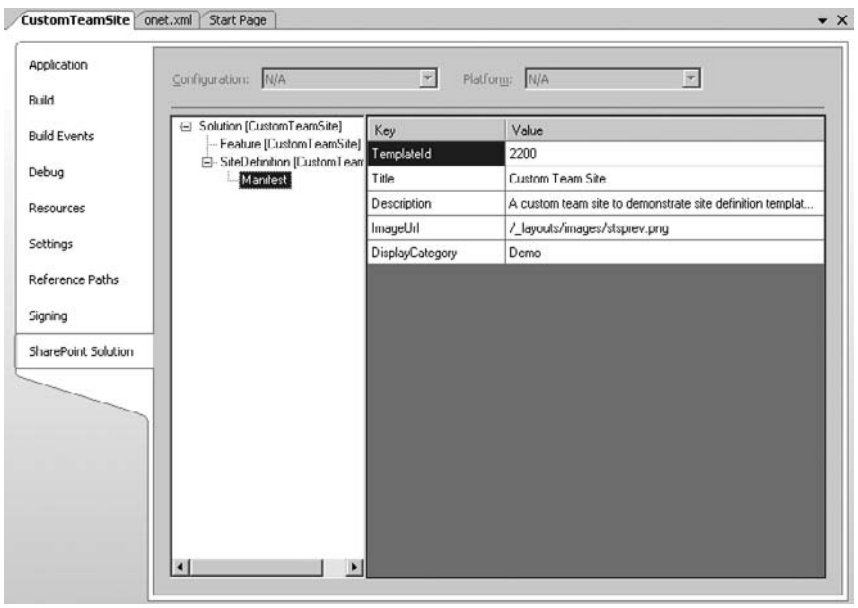


Figure 15.20
Properties loaded from the solution manifest.

11. From the Solution Explorer window, open the ONET.XML file for editing.
12. Add the XML listed in Figure 15.21 to the `Project` element.
13. Navigate to the `Configuration` element named `Default` with an `ID` of 0.

```
AlternateCSS="/_layouts/[%=System.Threading.Thread.CurrentThread.CurrentUICulture
.LCID%]/styles/corporate.css"
```

Figure 15.21 *AlternateCSS attribute of Project element.*

14. Set the `CustomMasterUrl` and the `MasterUrl` attributes to your master page file (Figure 15.22). Notice that we are referencing master pages that we expect to appear in the master page gallery for the site. This requires us to create a provisioning module that will instruct SharePoint to upload our master page during site creation.

```
    <Configuration ID="0" Name="Default"
CustomMasterUrl="_catalogs/masterpage/corporate.master"
MasterUrl="_catalogs/masterpage/corporate.master">
```

Figure 15.22 *CustomMasterUrl and MasterUrl attributes of Configuration element.*

15. Staying within the same `Configuration` element, navigate to the `Modules` child element.
16. Add two Module references, one named `LoadMasterPage` and the other named `LoadImage`. The code should look similar to that shown in Figure 15.23. Note that at this point, we haven't actually defined the modules; we have just added a reference to them. The modules are defined later in the ONET.XML in the `Modules` element that is a child of the `Project` element.

```
<Modules>
  <Module Name="Default" />
  <Module Name="LoadMasterPage" />
  <Module Name="LoadImage" />
</Modules>
```

Figure 15.23 *Modules child element.*

17. Navigate to the Default module contained in the `Modules` element that is a child of the `Project` element. This will appear immediately after the closing `Configuration` element tag.

18. Replace the image path for the file specified in the `AllUsersWebPart` element with `/Images/brief.gif` (Figure 15.24; replace brief.gif with your image file name).

Figure 15.24 *Updating the Web Part image file setting.*

19. Navigate to the closing tag for the `Default` module and add two additional `Module` elements, as shown in Figure 15.25. These modules basically tell SharePoint to place the master page and the image files in the master page gallery and images folder of the provisioned site respectively.

```
<Module Name="LoadMasterPage" List="116"
        Url="_catalogs/masterpage" RootWebOnly="False">
  <File Url="corporate.master" Type="GhostableInLibrary"
        IgnoreIfAlreadyExists="TRUE" />
</Module>
<Module Name="LoadImage"
        Url="Images" RootWebOnly="False">
  <File Url="brief.gif" Type="Ghostable"
        IgnoreIfAlreadyExists="TRUE" />
</Module>
```

Figure 15.25 *Modules required to provision files.*

20. To provision the files successfully, we need to load the files into the site definition folder. Simply drag your files from your desktop, or wherever you have them stored, and drop them directly into the Site Definition folder in the Visual Studio Solution Explorer that contains the ONET.XML file. All that remains is for us to build and deploy the solution.

21. From the Build menu, click Build CustomTeamSite.

22. From the Build menu, click Deploy CustomTeamSite. Notice the Visual Studio status bar displays the progress of the solution deployment process, going through stages such as Deploy Started…Creating Solution…Deploying Solution...Deploy Succeeded. This process may take some time. Once the solution is deployed successful, you can create sites based on the newly created site definition.
23. Navigate to the site directory of your site collection and click the Create Site link.
24. Populate all the necessary fields, such as the name of the site and the URL.
25. From the Template Selection section, click the tab that you chose as the display category for your custom site template, and select your custom site template.
26. Click Create. Your new SharePoint site should be provisioned with your master page, style sheet, and custom image applied (Figure 15.26).

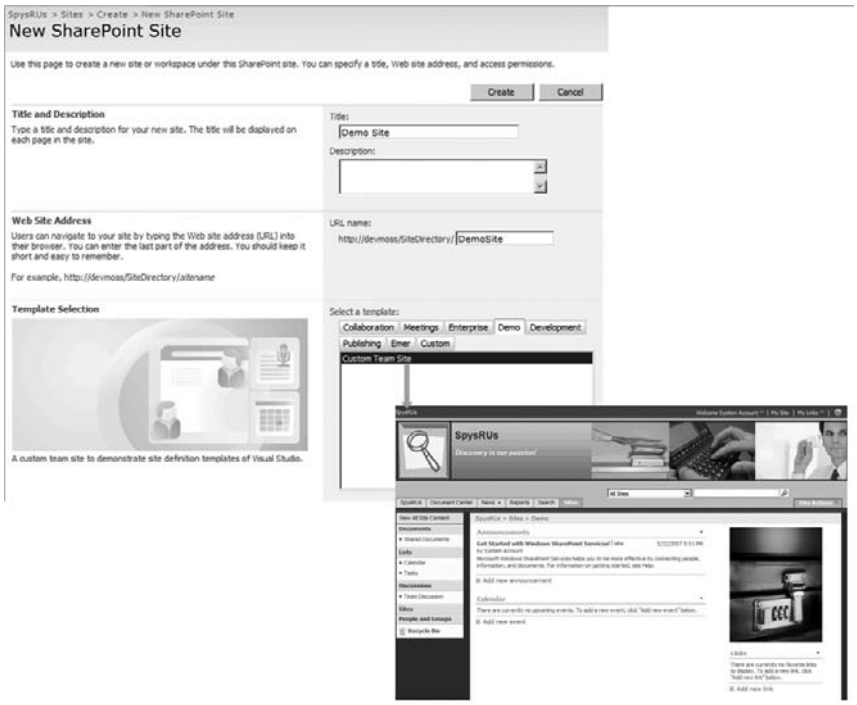


Figure 15.26 *Newly provisioned site.*

27. If you edit the newly provisioned site in SharePoint Designer 2007 and examine the _catalogs/masterpage folder and the images folder, you will see they contain the provisioned files from your site definition.

Throughout this chapter we have discussed the basic concepts and components involved in building your own custom site definitions and site templates. Our branding exercise showed how to provision own custom master page and images and apply them to newly created sites. Site definitions provide great opportunities for customization; however, they don't offer much in the way of flexibility when it comes to customizing sites after creation. The next chapter discusses the much more versatile topic of Features.

16

Features

In the previous chapter we discussed how to customize SharePoint using custom site definitions. Customizing SharePoint in this way is not new; SharePoint Portal Server 2003 was built on the site definition and site template infrastructure and included common XML files such as ONET.XML and WebTemp.XML. However, comparing the content from the ONET.XML of a SharePoint Portal Server 2003 site definition with the ONET.XML of a SharePoint Server 2007 site definition shows a significant reduction in the complexity and size of the ONET.XML. For example, one such difference is the lack of content in the `ListTemplates` element which, for SharePoint 2003, is responsible for defining the lists available to a site provisioned from the site definition. The types of lists defined include an announcements list, a document library, and a links list. Lists are still prevalent in SharePoint Server 2007 sites, so how do they get defined? The answer comes in the form of Features. A Feature is a new concept introduced with the SharePoint Server 2007 platform which allows you to easily add and remove functionality and customizations to and from your SharePoint sites. This chapter describes the Feature architecture and steps you through the Feature lifecycle.

16.1 What is a Feature?

When it comes to customizing and extending the capabilities of a SharePoint Portal Server 2003 deployment there is little in the way of efficiency or flexibility. Outside of the Web Part world, adding new functionality, such as a new type of list or navigation bar, to a SharePoint Portal Server 2003 deployment often results in creating a new custom site definition or modifying an out-of-the-box site definition, thereby placing your deployment in unsupported territory. It is cumbersome and inefficient to have to create an entire custom site definition to implement additional functionality. Thankfully, SharePoint Server 2007 introduces the concept of Features. A Feature is a package of components that implements a specific piece of functionality to your SharePoint deployment, such as a custom list, a set of Web Parts or a

complex workflow. When a Feature is installed, an entry is added to the configuration database of the SharePoint farm. The scoping of a Feature determines where in the farm it can then subsequently be activated; a Feature can be scoped to the farm, the Web application, the site collection or the Web site level. For example, if the Feature is scoped at the Web site level, the Feature is available for activation from the Site Settings page of each individual site within a site collection; whereas scoping at the site collection level means that the Feature is available for activation from the Site Collection Settings page at the root of a site collection.

With Features, you no longer need to create or modify a custom site definition to effect a simple change in your SharePoint environment. Instead, you can package all the various components needed to provide specific functionality and deploy the package as a Feature. For example, staying with the branding theme from the previous two chapters, let's suppose you want to change the look and feel of all your SharePoint site collections to match the season. Although this is quite common for a Web site, in SharePoint Portal Server 2003 this is difficult to achieve. However, in SharePoint 2007 the flexibility of master pages and Features makes this particular scenario quite straightforward. We will examine the inner workings of the scenario as we progress through this chapter, but, at a high level, it involves creating a set of Features containing all the master pages, themes, style sheets, and custom code and applying the appropriate brand at the appropriate time.

16.2 SharePoint 2007 Features

SharePoint 2007 and WSS 3.0 make extensive use of Features to provide capabilities ranging from simple file provisioning to fully functional workflow components. A quick glance at the Feature pages in a typical site will give you some idea of the components that have been implemented through this new capability. For example, if you navigate to the Site Collection Features page of a typical collaboration portal installation, you will see a host of Feature offerings, some active, some inactive (Figure 16.1).

Similarly, if you view the Farm Features, Web Application Features and Site Features pages, you will see additional components available to activate and deactivate at will. However, where Features are concerned, what you see is not actually what you get. Features are *much* more pervasive in the SharePoint platform than is visible through the Web interface. For example, the list of master pages and their associated style sheets and images are provisioned to a site collection through a hidden Feature named PublishingLayouts. Features physically reside in the `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\FeatureS` directory. If you examine this folder on a server running SharePoint Server 2007, you will see a folder named PublishingLayouts (Figure 16.2). Inside

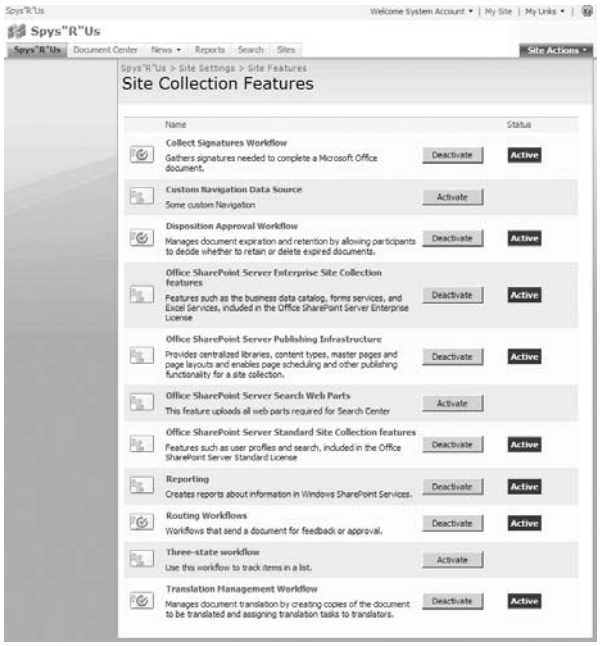


Figure 16.1
Default Site Collection Features.

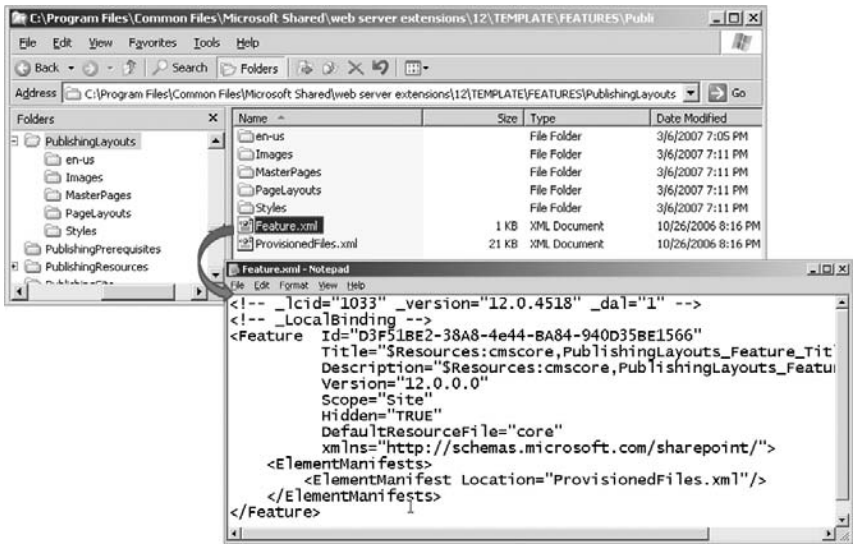


Figure 16.2
PublishingLayouts Feature

the PublishingLayouts folder are a number of files and folders containing all the master pages and their associated files. The PublishingLayouts Feature is a hidden Feature scoped to the site collection level. If you refer back to the Site Collection Features page for a Collaboration Portal site (Figure 16.1) you will see that the PublishingLayouts Feature is not listed because it is a

hidden Feature. The PublishingLayouts Feature is activated upon site collection creation and provisions all the files and images listed in the Feature definition into the appropriate document library, such as placing master pages in the master page gallery. This is just one example of how the implementation of Features provides core functionality to SharePoint. Examining each of the default Features in the `FeatureS` directory can give you a good understanding of the capabilities that Features bring to the table.

16.3 Feature Architecture

We have stated that a Feature is a package of components that implements a specific piece of functionality to your SharePoint deployment. But what does this actually mean, and what does a Feature physically look like? At the very basic level, a Feature is a folder containing one or more files that define and deliver the functionality. As previously mentioned, all Feature folders reside in the `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\FeatureS` directory. If you examine each of the default Microsoft Feature folders, you will see they each contain a Feature definition file (Feature.xml). The Feature definition file contains a `Feature` element that defines the necessary attributes, properties, activation dependencies, paths to element definition files, and references to receiver assemblies. The structure of the `Feature` element is shown in Figure 16.3.

Figure 16.3 *Feature.XML structure.*

```
<Feature...>
  <ActivationDependencies>
      <ActivationDependency>
            ...
      </ActivationDependency>
  </ActivationDependencies>
  <ElementManifests>
      <ElementManifest>
            ...
      </ElementManifest>
  </ElementManifests>
  <Properties>
      <Property>
            ...
      </Property>
  </Property>
</Feature>
```

The child elements, `ActivationDependencies`, `ElementManifests`, and `Properties`, are all optional elements each further defining the Feature. We will revisit each of these child elements in a moment, but for now let's

Figure 16.4
Feature element attributes.

```
<Feature
    Id = "GUID in format:11111111-1111-1111-1111-11111111111 "
    Title = "Text"
    Description = "Text"
    Creator = "Text"
    ImageUrl = "Text"
    ImageUrlAltText = "Text"
    Version = "Text"
    Scope = "Farm|WebApplication|Site|Web"
    Hidden = "TRUE" | "FALSE"
    ReceiverAssembly = "Text"
    ReceiverClass = "Text"
    AlwaysForceInstall = "TRUE" | "FALSE"
    ActivateOnDefault = "TRUE" | "FALSE"
    AutoActivateInCentralAdmin = "TRUE" | "FALSE"
    DefaultResourceFile = "Text"
    RequireResources = "TRUE" | "FALSE"
    SolutionId = "Text" >
</Feature>
```

focus on the parent element. Attributes of the parent `Feature` element specify characteristics such as the title of the Feature as it would appear on the Feature settings page in the browser user interface. Figure 16.4 and Table 16.1 describe the attributes that are associated with the `Feature` element. Note that only the Feature `Id` and the `Scope` attributes are required for a Feature to successfully install. A Feature containing these two attributes alone would not be very useful, but it would install nonetheless, and appear in the Feature settings page of the specified scope.

The `Scope` attribute allows you to define where the Feature will be activated and on which Feature settings page it will appear. This setting is restricted to one of the following choices: `Farm`, `WebApplication`, `Site`, and `Web`. If the `Scope` attribute is set to `Farm` then the Feature is installed at the farm level and, if not hidden, appears on the Manage Farm Features page accessible from the Operations page in the central administration user interface. Features scoped at the farm level are activated automatically by default; however, this behavior can be controlled by setting the `ActivateOnDefault` Feature attribute. If the `Scope` attribute is set to `WebApplication`, the Feature is installed at the Web application level and, if not hidden, appears on the Manage Web Application Feature page accessible from the Application Management page in the central administration user interface. Features scoped at the Web application level are *not* automatically activated by default. Again, this behavior can be controlled by setting the `ActivateOnDefault` Feature attribute. However, this only affects newly created Web applications, existing Web applications still need to be activated. If the `Scope` attribute is set to `Site`, the Feature is installed at the site collection level and, if not hidden, appears on the Site Collection Features page accessible from the Site Settings

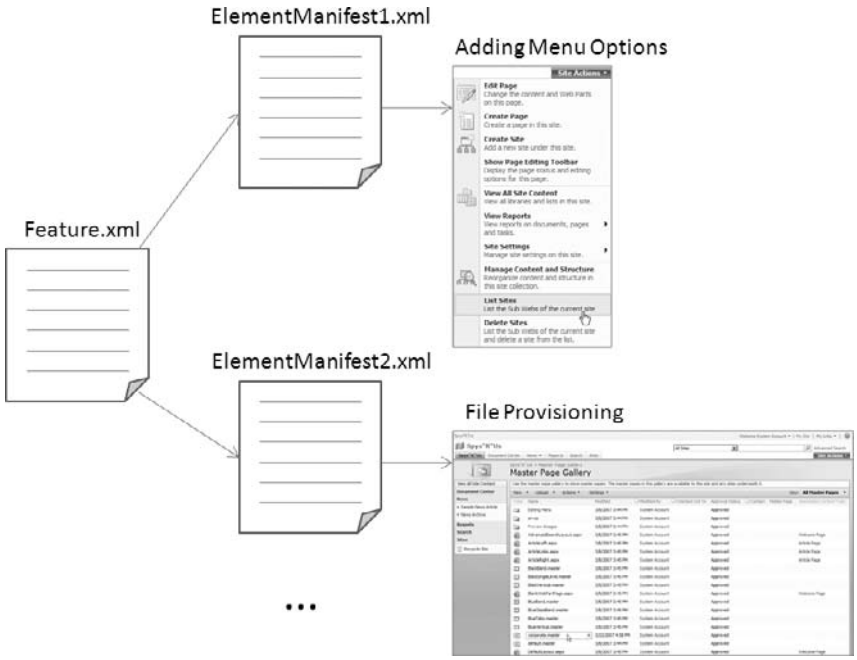


Figure 16.5
Element manifests.

page at the root of the site collection. If the `Scope` attribute is set to `Web`, the Feature is installed at the individual Web site level and, if not hidden, appears on the Site Features page accessible from the Web site's Site Settings page. Features scoped at the site collection level and the Web site level are *not* activated by default and, unlike the `Farm` and `WebApplication` scoping, this behavior cannot be controlled by the `ActivateOnDefault` attribute. It should be noted, however, that Features scoped at the site collection and Web site levels are automatically activated upon site creation if they are associated, or stapled, to a site definition. We'll discuss the detail of Feature activation and Feature stapling later in the chapter.

Another important factor that influences the scope of your Feature is the element type that you are implementing. For example, if you are using Features to create a content type, then your Feature must be scoped to the site collection level, as content type Features are not supported at the Web site, Web application, and Farm levels. The Elements By Scope section of the WSS 3.0 SDK provides a complete listing of the supported element types for each scope.

Table 16.1 *Feature Element Attributes*

Attribute	Description	Required
Id	Attribute that defines the Globally Unique Identifier (GUID) for the Feature.	Yes
Title	Attribute that defines the title of the Feature as you would like it to appear in the Feature Settings page for the specified scope. For example, if the Feature scope is set to Site Collection, the title will appear for the Feature in the Site Collection Features page under the Site Collection administration. Note: if the `Hidden` attribute is set to `False` the Feature title will not be listed on the Features Setting page. Best practice is to include an appropriate title that would let the administrator know what the Feature will actually do upon activation. If you leave this attribute blank, the Feature name on the Features pages is replaced with the text: "Untitled Feature."	No
Description	Text attribute that defines the description of the Feature as you would like it to appear in the Feature Settings page for the specified scope. For example, if the Feature scope is set to Site Collection, the description will appear for the Feature in the Site Collection Features page under the Site Collection administration. Note: if the `Hidden` attribute is set to `False`, the Feature description will not be listed on the Features Setting page. The attribute works hand in hand with the Feature title, and best practice is to include an appropriate description that would let the administrator know what the Feature will actually do upon activation.	No
Creator	Attribute that defines the author, or creator of the Feature.	No
ImageUrl	Attribute that defines a site-relative URL to the image file that will be used to represent the Feature on the Feature Settings page.	No
ImageUrlAltText	Attribute defining the alternate text for the image file defined in the `ImageUrl` attribute.	No
Version	Attribute defining the version of the Feature in System.Version compliant format. For example, 1.0.0.0.	No

Table 16.1 *Feature Element Attributes (continued)*

Attribute	Description	Required
Scope	Attribute defining the scope of the Feature, which ultimately determines where the Feature will be activated and on which Feature Settings page it will appear. This setting is restricted to one of the following choices: `Farm`, `WebApplication`, `Site`, and `Web`.	Yes
Hidden	Attribute that defines whether the Feature will appear on the Feature Settings pages in the browser user interface. Default setting: `False`	No
ReceiverAssembly	Attribute that defines the Feature Receiver assembly file, located in the global assembly cache and used to handle specific Feature events, such as Feature activation. This attribute is used in conjunction with the `ReceiverClass` attribute. Reference the section on Feature Receivers for more information.	No
ReceiverClass	Attribute that defines the class that implements the Feature event handler. This attribute is used in conjunction with the `ReceiverAssembly` attribute. Reference the section Feature Receivers for more information.	No
AlwaysForceInstall	Attribute that affects all scopes and defines whether the Feature forces an install even if the Feature is currently installed. Default setting: `False`	No
ActivateOnDefault	Attribute that determines if a Feature should be activated automatically upon installation, or when a Web application is instantiated. Unfortunately, this attribute is only applicable to Features scoped at the `Farm` level and the `WebApplication` level. Only newly created Web applications are affected by this setting. This attribute cannot be used for Features scoped at the `Site` and `Web` levels. Default setting:`False`	No

Table 16.1 *Feature Element Attributes (continued)*

Attribute	Description	Required
AutoActivateInCentralAdmin	Attribute that determines whether a Feature should be installed and activated *only* in the central administration area. For example, if you have a Feature scoped to site, and the `AutoActivateInCentralAdmin` attribute set to `True`, then the Feature will be installed and activated in the Site Collection of the central administration Web site only. It will not be installed at any other site collection. Default setting: `False`	No
DefaultResourceFile	Attribute used to define a *shared* file to be used for all your Feature resources. The default directory that Windows SharePoint Services checks for the existence of the file referenced in this attribute is the `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\Resources` folder. If you would like to reference a resource file but do not want to use a shared resource file, use the `RequireResources` attribute instead.	No
RequireResources	Attribute that allows you to determine whether a Feature is visible in the browser user interface, depending on whether resources exist for a particular language for the Feature. If the RequireResources attribute is set to `True`, then Windows SharePoint Services checks the `\Resources` folder of the Feature for resource files with the name format `Resources.xx-xx.resx`, where `xx-xx` is the culture. If the file for a particular language is not present, then the Feature will be hidden from the user interface. Default setting: `False`	No
SolutionId	Specifies the GUID of the solution to which the Feature belongs.	No

16.3.1 Element Manifests

A Feature can optionally contain references to one or more element manifests. An element manifest is a separate file containing definitions for specific element types, such as a content type, a field definition, or a menu action definition (Figure 16.5). Table 16.2 shows a list of the various element types that you can use Features to define. As you can see, Features can be used to provide many forms of additional functionality to your SharePoint deploy-

Table 16.2 *Element Type*

Element Type	XML Element	Description
List Template Files	ListTemplate	Defines a list template to display on the Create page when creating content in SharePoint.
List Instances	ListInstance	Defines the default data to provision in a specific list type upon creation.
Modules	Module	Defines the files to provision in a SharePoint library. For example, the PublishingLayouts Feature uses the Module element to provision the default master pages to the master page gallery.
Field Definition Schema	Field	Defines the properties of a site column. The site column can then be assigned to lists across many SharePoint sites.
Event Registrations	Receivers	Defines an event receiver for handling list item events. See Chapter 17 for more information on event registrations.
Custom Action Definitions	CustomAction & CustomActionGroup	Defines a custom action for a link or toolbar in the SharePoint browser interface. Use CustomActionGroup to group a set of custom actions together.
Content Type Definitions	ContentType	Defines a content type, including any event handlers associated with the content type.
Delegate Controls	Control	Identify specific custom controls to represent SharePoint functionality at run time. For example, you can leverage this definition to replace the Search box on all the SharePoint pages with your own custom Search box.
Feature Site Template Associations	FeatureSiteTemplateAssociation	Allows you to associate a Feature with a specific site template, so that when a site is created using that specific site template, it inherits the associated Features. This concept is known as Feature stapling and is discussed later in the chapter.
Workflow Definitions	Workflow	Defines a workflow, including identifying the forms and metadata used in the workflow process. For example, the ReviewWorkflows Feature folder contains definitions for the SharePoint Server 2007 review workflow template, such as the Approval and Feedback workflows.
Content Type Binding Schema	ContentTypeBinding	Binds a content type to a specific list by specifying the content type GUID and the URL of the list.

Table 16.2 *Element Type (continued)*

Element Type	XML Element	Description
Document Converter	DocumentConverter	Defines a custom document converter that converts files of one format to files of a different format.

ment. To further illustrate the purpose of element manifest files, we'll examine two element types in more detail: modules and custom actions.

16.3.2 Defining Modules

Module elements are used extensively throughout SharePoint. In Chapter 15, we showed how the `Module` element in ONET.XML can be used to provision files during the instantiation of a site. Similarly, when defining Features, the `Module` element can be used to provision files to specific locations in SharePoint. We have already discussed how the PublishingLayouts hidden Feature is responsible for provisioning the default master pages, such as BlueBand.master, that populate the SharePoint Server 2007 master page gallery. The ProvisionedFiles.xml element manifest of the PublishingLayouts Feature contains module definitions that provision all the necessary files and images. Figure 16.6 shows an example of an element manifest that provisions files to the master page gallery, the style library, and the site collection images library. The contents of the Feature folder are displayed in the upper left corner of the image. The Feature comprises three subfolders containing the files to be provisioned, a Feature.xml file defining the Feature, and an element manifest file named ProvisionedFiles.xml defining the file provisioning `Module` elements. There are three module elements defined: CustomBrandMasterPages, CustomBrandStyleSheets, and CustomBrandImages. The CustomBrandMasterPages module provisions three master page files into the master page gallery. The `List` attribute of the `Module` element specifies the type of list in which the files will be provisioned. In this instance, the list type is equal to 116, which translates to a master page gallery. As a side note, you can find the list template definition for the master page gallery in the ONET.XML of the GLOBAL site definition. The `Url` attribute of the `Module` element goes on to define the URL of the destination folder in which the files will be provisioned. In this instance, the `Url` is a site-relative URL that points to the master page gallery of the current site. The `Path` attribute specifies the source folder in which the files to be provisioned reside, which in this case equals the MasterPages folder within our Feature folder. Finally, the `RootWebOnly` attribute defines whether the files are provisioned *only* in the root Web of the site collection. In our example, we only want the files to be populated in the master page gallery at the root of the site collection, so we set this attribute to `True`. The `File` element is a child element of the `Module` element and is used to

identify the files to be provisioned. In our example, the `File` element has two attributes: the `Url` attribute and the `Type` attribute. The `Url` attribute specifies the location of the file within the folder defined by the `Path` attribute of the `Module` element. The `Type` attribute specifies that the file should be cached in memory on the front-end Web server and determines whether the file will be provisioned in a document library (`GhostableInLibrary`) or outside a document library (`Ghostable`); as we are provisioning our files in the master page gallery, and the master page gallery is effectively a document library, we set this attribute to `GhostableInLibrary`. The CustomBrandStyleSheets and CustomBrandImages modules perform the same type of provisioning function as the CustomBrandMasterPages module. For completeness, the corresponding Feature.xml file is shown in Figure 16.7. This file references the ProvisionedFiles.xml element manifest. All that remains is for the Feature to be installed and activated. Upon activation, the files are copied to the appropriate location. Notice that this time the `Hidden` attribute of the `Feature` element is set to `True`, which means that the Feature will not appear in the browser user interface for activation, so we will need to activate the Feature by some other mechanism, such as through the command line. We discuss the installation and activation process later in this chapter.

Figure 16.6
File provisioning element manifest .

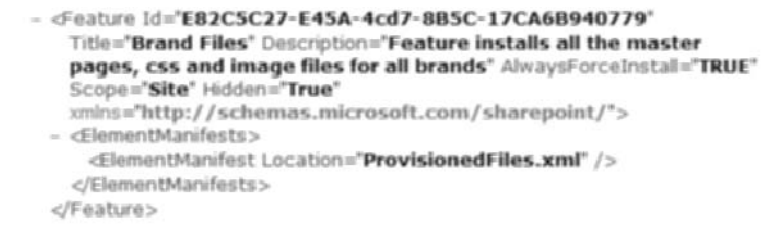


Figure 16.7
File provisioning Feature.xml.

Customized and uncustomized pages

When provisioning files through the use of Features, it is important to understand what happens when files that you have provisioned are subsequently *customized* using the Web content management functionality in SharePoint. In our example, we used Features to provision some master page files to the master page gallery. Any updates that we want to make to those master pages can be rolled out by simply overwriting the existing files in the Feature folder. Any sites currently using the master page will automatically reflect the new updates upon a page refresh. This is great and shows how the convenience of Features makes the rollout of subsequent updates very easy; however, the concept only applies to site collections that have not customized the provisioned master pages through the Web content management Features of SharePoint. If you use the Web content management Features to publish your changes to the master page gallery, then the master pages that you check in and check out are marked as customized (unghosted), and SharePoint will subsequently reference the database for the file instead of the Feature folder. If you update the original master in the file system, the changes will *not* take effect in the site collections that contain the customized master. Fortunately, you can easily uncustomize (reghost) the master page to its original by navigating to `/_layouts/reghost.aspx` and entering the URL to your master page (e.g., `_catalogs/masterpage/emer2.master`). This is especially useful because at the time of writing you cannot delete a master page from the master page gallery. Please read Microsoft KB 926812 for more information.

16.3.3 Defining Custom Actions

The `CustomActions` element allows you to add additional menu options to various locations throughout the SharePoint user interface. Our next example shows how, using the `CustomActions` element, you can easily add additional menu options, or actions as they are referred to in SharePoint, to the Site Actions drop-down menu. Figure 16.8 shows the Feature.xml file, the element manifest file, and the resulting output of a Feature that adds two additional menu options, List Sites and Delete Sites, to the Site Actions menu. The `CustomAction` element has a number of attributes that help

define the custom action. The `Id` attribute is used to uniquely identify the custom action. This attribute may be a simple string such as CustListSites or a GUID, the important factor being that the identifier is unique within the realm of custom actions. The `GroupId` attribute determines in which menu grouping the custom action will appear. Examples of possible values include `SiteActions` and `NewMenu`. The `SiteActions` setting tells SharePoint to add the custom actions to the Site Actions menu, as shown in our example. The `NewMenu` setting tells SharePoint to add the custom actions to the New menu option that appears in lists and document libraries. When defining a custom action, you specify the menu grouping along with the namespace associated with that grouping. The associated namespace is defined in the `Location` attribute of the `CustomAction` element. The `SiteActions` grouping is located in the `Microsoft.SharePoint.StandardMenu` namespace. Table 16.3 lists the menu group identifiers and their corresponding location settings. You can use the `RegistrationId` and `RegistrationType` attributes of the `CustomAction` element to further define where and when a specific custom action appears. For example, you can specify that your custom action should only appear in a specific list or content type.

The `Title` attribute of the `CustomAction` element represents the text shown in the user interface for the action, and the `Description` attribute represents the text description that appears below the action. You can associate an image with your action using the `ImageUrl` attribute; our example does not leverage this attribute. The `Sequence` attribute determines the position of the action within the grouping. In our example, we have set the `Sequence` attribute for our two custom actions to `1001` and `1002`, ensuring that they appear at the bottom of the grouping. Had we not specified a sequence number, the actions would still have appeared at the bottom of the Site Actions menu; however, they would appear in alphabetical order, making the Delete Sites action appear before the List Sites action.

The `Rights` attribute is quite powerful, allowing you to easily tailor the accessibility of a menu option to a user's permissions. In our example, we have set the `Rights` attribute to `ManageSubWebs`, which means that if a user has been granted the `ManageSubWebs` permission, they will see the Delete Sites menu option; otherwise, the menu option will not be displayed. The possible settings for the `Rights` attribute can be found by examining the `SPBasePermissions` enumeration of the `Microsoft.SharePoint` namespace. The `SPBasePermissions` enumeration shows all the built-in permissions available in SharePoint. If you want to check for multiple permissions, separate each permission value with a comma—for example, `Rights="AddListItems, DeleteListItems"`.

The only task remaining is to link the menu option to a specific action. In our example, we connect the custom actions to two aspx pages stored in the _layouts directory: ListSites3.aspx and DeleteSite2.aspx. For details on

Figure 16.8 *Adding new menu actions.*

the code behind these aspx pages, please refer to Chapter 12, SharePoint Programmability. The `UrlAction` element is a child element of the `CustomAction` element, which contains a `Url` attribute that specifies the site-relative URL of the page to be displayed.

In addition to adding custom actions to the SharePoint menus, you are able to hide the default actions installed by SharePoint. This capability is provided through the use of the `HideCustomAction` element, and the trick to hiding an existing action is in knowing its identifier. If you scour the `FeatureS` folder you will find many Features that define custom actions in SharePoint, giving you access to the identifier for the custom action. At the time of writing, Microsoft has not released a documented list of identifiers for all the

Table 16.3 *GroupIDs and Locations for SharePoint Menus*

GroupId	Location	Description
SiteCollectionAdmin	Microsoft.SharePoint.SiteSettings	The links that appear on the Site Settings page under the Site Collection Administration column
SiteAdministration	Microsoft.SharePoint.SiteSettings	The links that appear on the Site Settings page under the Site Administration column
Galleries	Microsoft.SharePoint.SiteSettings	The links that appear on the Site Settings page under the Galleries column
UsersAndPermissions	Microsoft.SharePoint.SiteSettings	The links that appear on the Site Settings page under the Users and Permissions column
Customization	Microsoft.SharePoint.SiteSettings	The links that appear on the Site Settings page under the Look and Feel column
SiteActions	Microsoft.SharePoint.StandardMenu	The links that appear on the Site Actions Menu
NewMenu	Microsoft.SharePoint.StandardMenu	The links that appear on the New menu of a document library or list
ActionsMenu	Microsoft.SharePoint.StandardMenu	The links that appear on the Actions menu of a document library or list
SettingsMenu	Microsoft.SharePoint.StandardMenu	The links that appear on the Settings menu of a document library or list
UploadMenu	Microsoft.SharePoint.StandardMenu	The links that appear on the Upload menu of a document library or list
ActionsMenuForSurvey	Microsoft.SharePoint.StandardMenu	The links that appear on the Actions menu of a survey list
SettingsMenuForSurvey	Microsoft.SharePoint. StandardMenu (note: there is an error in the SDK regarding the location of this GroupId)	The links that appear on the Settings menu of a document library or list

actions in the default SharePoint menus. Do not waste time trying to search for the identifier of a specific option on one of the default SharePoint toolbars. Using the `HideCustomActions` element, you can override the menu options that have been implemented as custom actions in SharePoint; however, you *cannot* override the out-of-the-box toolbar buttons using `HideCustomAction`, because the toolbar buttons are actually implemented using control templates.

Consequently, to override the default SharePoint toolbar buttons, such as those found on a document library form, you must override the associated templates. The ASP.NET control templates are defined in the `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\CONTROLTEMPLATES\` directory. Directly editing the default .ascx files that ship with SharePoint is not supported; therefore, to override a default template, you must create a custom .ascx file that has the same control template ID of the default template and drop it into the `\CONTROLTEMPLATES` directory. More information on overriding the default custom control templates can be found in the WSS 3.0 SDK.

16.3.4 Feature Properties

The `Properties` element allows you to define a set of properties, in the format of key/value pairs, to be used by your Feature. This is particularly useful when you have event receivers or Feature receiver assemblies associated with your Feature, because your code then has instant access to the Feature property bag. The `Properties` property of the `SPFeature` class or `Microsoft.SharePoint.Administration.SPFeatureDefinition` class gives you access to the collection of properties defined in a Feature. We will discuss Feature receivers and the Feature object model later in the chapter. Figure 16.9 shows the Feature.XML for the ReviewWorkflows Feature that ships with SharePoint Server 2007. Notice that there are two properties defined, consisting of key/value pairs: `RegisterForms` and `GloballyAvailable`. Also notice that a Feature receiver is registered for this Feature, which means that the `RegisterForms` and `GloballyAvailable` properties and their corresponding values are accessible to the Feature receiver.

Figure 16.9 *ReviewWorkflows Feature illustrating use of properties.*

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Copyright (c) Microsoft Corporation. All rights reserved. -->
- <Feature Id="02464C6A-9D07-4F30-BA04-E9035CF54392"
  Title="$Resources:Feature_Title;" Description="$Resources:Feature_Description;"
  ImageUrl="WorkflowSolutionFeature.gif" Version="12.0.0.0" Scope="Site"
  AlwaysForceInstall="TRUE" 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/" DefaultResourceFile="_Res">
  - <ElementManifests>
    <ElementManifest Location="ReviewFeedback.xml" />
    <ElementManifest Location="ReviewApproval.xml" />
  </ElementManifests>
  - <ActivationDependencies>
    <ActivationDependency FeatureId="C9C9515D-E4E2-4001-9050-
      74F980F93160" />
  </ActivationDependencies>
  - <Properties>
    <Property Key="RegisterForms" Value="Forms\*.xsn" />
    <Property Key="GloballyAvailable" Value="true" />
  </Properties>
</Feature>
```

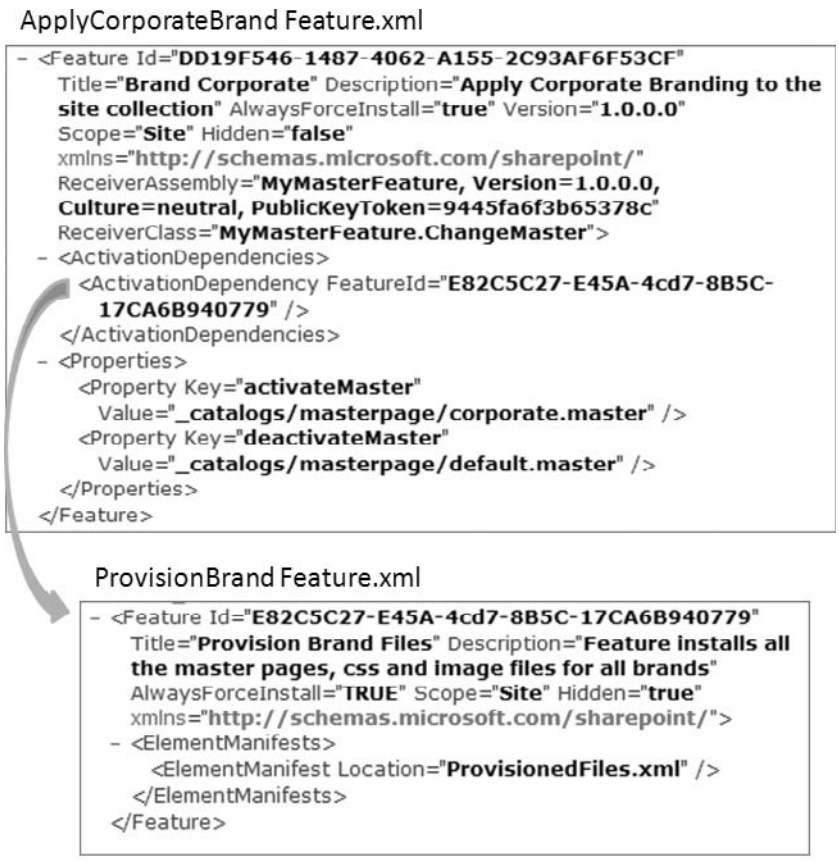
16.3.5 Activation Dependencies

The `ActivationDependencies` element allows you to identify any additional Features that your Feature relies on before it can be activated. For example, let's suppose you have a Feature (ApplyCorporateBrand) that implements code that, upon activation, sets the system master and the site master settings of a site to a specific master page. Another Feature (ProvisionBrand) provisions master pages to the master page gallery, including the master page referenced by the ApplyCorporateBrand Feature. Consequently, the ApplyCorporateBrand Feature *depends* upon the ProvisionBrand Feature being activated; therefore, the Feature.xml for the ApplyCorporateBrand Feature should include a reference to the ProvisionBrand Feature in a separate `ActivationDependency` child element of the `ActivationDependencies` element. The `ActivationDependency` element has an associated `FeatureId` attribute that is set to the GUID of the Feature upon which the current Feature depends. Figure 16.10 shows the Feature.xml of the ApplyCorporateBrand and the ProvisionBrand Features. Notice that the GUID listed in the activation dependency for the ApplyCorporateBrand file matches the `Id` attribute of the ProvisionBrand Feature.

The process of identifying activation dependencies is quite straightforward; however, there is a certain complexity with respect to the rules to which activation dependencies need to adhere. For example, the `Scope` and `Hidden` attributes affect the activation dependency options for a given Feature. The rules are as follows:

- If FeatureB depends upon FeatureA of the same scope, and FeatureA is not activated when FeatureB is activated, SharePoint automatically activates FeatureA.
- If FeatureB is deactivated while FeatureA is still activated and FeatureB depends upon FeatureA, SharePoint automatically deactivates FeatureA only if there are no additional Features dependent upon FeatureA.
- A Feature cannot depend on another Feature of a more restrictive scope. For example, if FeatureA is scoped to the site collection level, and FeatureB is scoped at the Web site level, then FeatureA cannot depend upon FeatureB; however, FeatureB can depend upon FeatureA.
- You cannot activate a Feature if the dependent Feature is of a different scope and is inactive. If FeatureA is scoped to the site collection level, and FeatureB is scoped to the Web site level, and FeatureB depends upon FeatureA, then if FeatureA is inactive, any attempt to activate FeatureB results in an error.
- A Feature cannot depend on another Feature of a different scope if the Feature that is depended upon is hidden. For example, if Fea-

Figure 16.10
ActivationDependencies Example



tureA is scoped to the site collection level, and FeatureB is scoped to the Web site level, then FeatureB cannot depend upon FeatureA if FeatureA is hidden.

- Hidden Features cannot have activation dependencies. For example, if there are two Features, FeatureA and FeatureB, and the `Hidden` attribute of FeatureB is set to `TRUE`, then FeatureB cannot depend upon FeatureA; but FeatureA can depend upon FeatureB.
- Multilevel dependencies are not supported for Features that have the `Hidden` attribute set to `FALSE`; only single-level dependencies are supported in this instance. For example, if you have three Features, FeatureA, FeatureB, and FeatureC, each with their `Hidden` attribute set to `FALSE`, then you cannot have scenario in which FeatureC depends upon FeatureB which in turn depends upon FeatureA. However, Fea-

tureC can depend upon FeatureA, and FeatureB can depend on FeatureA.

- Multilevel dependencies are supported for Features that have the `Hidden` attribute set to `TRUE`. For example, suppose you have three Features, FeatureA, FeatureB, and FeatureC; FeatureB and FeatureC have their `Hidden` attribute set to `FALSE`, and FeatureA has its `Hidden` attribute set to `TRUE`. In this scenario, you can have FeatureC depend upon FeatureB, which in turn depends upon FeatureA.

When designing your Feature architecture, especially where cross-scope activation dependencies are concerned, it is important to ensure that your Feature is protected from being left in an undesirable state. For example, if you have a collection of same-scoped Feature activation dependencies and overlapping cross-scoped Feature activation dependencies, then you need to make sure that in no circumstance is a Feature left activated when the Feature upon which it depends has been deactivated. If this scenario is unavoidable, then at the very least, you should ensure that your Feature can handle the missing Feature gracefully.

16.4 Feature Receivers

A Feature receiver is a special type of event receiver assembly that allows you to run custom code when specific Feature events occur, such as when a Feature is installed, activated, deactivated, or uninstalled. Figure 16.10 shows the Feature.xml for the ApplyCorporateBrand Feature. In this particular example, we use a Feature receiver to help push down a specific master page to all sites within the site collection in which the Feature receiver is activated. This functionality mimics the behavior of the "Reset all subsites to inherit this site/system master page" setting on the Site Master Page Settings aspx page. When the Feature is deactivated, we then reverse the process and set the master page to the default Microsoft master page. The `Feature` element has two attributes specific to Feature receivers: `ReceiverAssembly` and `ReceiverClass`. The `ReceiverAssembly` attribute is set to the strong name of the assembly that you would like to handle the events triggered by the Feature; the assembly DLL must be installed in the global assembly cache. The `ReceiverClass` attribute is used in conjunction with the `ReceiverAssembly` attribute and specifies the class within the receiver assembly that contains the event handler code. In our example, the `ReceiverAssembly` attribute is set to the MyMasterFeature receiver assembly and the `ReceiverClass` attribute is set to the MyMasterFeature.ChangeMaster receiver class. In addition to the receiver settings, the ApplyCorporateBrand Feature definition file contains two `Property` definitions, `activateMaster` and `deactivateMaster`; each property value is set to the URL of a specific master page in the default master page gal-

lery. As previously mentioned, the properties defined in the Feature.xml file are accessible to Feature receivers. Figure 16.11 shows some very basic code for the ChangeMaster class in the MyMasterFeature receiver assembly. Once again, it is important to note that this code is for demonstration purposes only, as the code does not contain the appropriate exception handling required for production. The ChangeMaster class inherits from the `SPFeatureReceiver` class, thus making it a Feature receiver class (line 9). Once you have set up the Feature receiver inheritance, you are free to override the Feature receiver methods such as `FeatureInstalling`, `FeatureActivated`, `FeatureDeactivating`, and `FeatureUninstalling`. In this example, we override the `FeatureActivated` method (lines 11 thru 24) and the `FeatureDeactivating` method (lines 26 thru 39). The `FeatureActivated` method retrieves the value associated with the `activateMaster` property in the Feature definition, traverses through each site in the site collection, and sets the site's `MasterUrl` and `CustomMasterUrl` properties equal to the value retrieved from the activateMaster property. The `FeatureDeactivating` method retrieves the value associated with the `deactivateMaster` property in the Feature definition, traverses through each site in the site collection, and sets the site's `MasterUrl` and `CustomMasterUrl` properties equal to the value retrieved from the `deactivateMaster` property. The properties provided by the `SPFeatureProperties` object include a pointer to the parent of the Feature, so you always have access to the Web site, site collection, Web application, or farm in which the Feature is being activated (line 16 and line 31).

16.5 Feature Lifecycle

So far we have discussed the various files, attributes, and elements that comprise the Feature architecture, but we have not really discussed how to actually create, install and activate a Feature. The basic lifecycle of a Feature involves creating the Feature, deploying the Feature, installing the Feature, activating the Feature, deactivating the Feature, uninstalling the Feature and then ultimately removing the Feature files from all the front-end Web servers (Figure 16.12). The adding and removing of Feature files can be performed manually on each front-end Web server; however the best practice recommendation is to use the Solution Deployment Framework to deploy and manage the Feature. Using the Solution Deployment Framework offers the advantage of being able to easily retract the solution from all Web servers within the farm. Figure 16.12 doesn't take into account the removal of the files provisioned by the Feature itself. As a best practice, your Feature should clean up its footprint during the uninstall process; in the event that the files cannot be removed, such as is the case for master pages, the uninstall process should ensure that the files are hidden or, at the very least, that all settings revert to their default or prior settings. In the remainder of this section, we'll assume that the deployment and removal of the files, including provisioned

Figure 16.11 *MyMasterFeature. ChangeMaster Feature receiver code (C#).*

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

namespace MyMasterFeature
{
    class ChangeMaster:SPFeatureReceiver
    {
        public override void FeatureActivated(SPFeatureReceiverProperties properties)
        {
            SPFeaturePropertyCollection propColl = properties.Feature.Properties;
            SPFeatureProperty propA = propColl["activateMaster"];
            string activateMaster = propA.Value;
            SPSite siteColl = (SPSite)properties.Feature.Parent;
            SPWebCollection webColl = siteColl.AllWebs;
            foreach (SPWeb spWeb in webColl)
            {
                spWeb.MasterUrl = siteColl.ServerRelativeUrl + activateMaster;
                spWeb.CustomMasterUrl = siteColl.ServerRelativeUrl + activateMaster;
                spWeb.Update();
            }
         }

        public override void FeatureDeactivating(SPFeatureReceiverProperties properties)
        {
            SPFeaturePropertyCollection propColl = properties.Feature.Properties;
            SPFeatureProperty propB = propColl["deactivateMaster"];
            string deactivateMaster = propB.Value;
            SPSite siteColl = (SPSite)properties.Feature.Parent;
            SPWebCollection webColl = siteColl.AllWebs;
            foreach (SPWeb spWeb in webColl)
            {
                spWeb.MasterUrl = siteColl.ServerRelativeUrl + deactivateMaster;
                spWeb.CustomMasterUrl = siteColl.ServerRelativeUrl + deactivateMaster;
                spWeb.Update();
            }
      }

        public override void FeatureInstalled(SPFeatureReceiverProperties properties)
        {
        }

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

files, are handled, and we will focus on the creation, installation, activation, deactivation, and uninstall phases of the life cycle.

16.5.1 Creating a Feature

As previously mentioned, a Feature is a folder containing one or more files that define and deliver the functionality. There is no real rocket science behind the creation of Features; the process involves creating a folder in the \Feature directory, creating a Feature definition file, and creating the additional files required by the Feature, such as an element manifest file. Obviously, if your Feature registers an event receiver or implements a Feature assembly, then the process is more complex and involves the development effort of creating the custom

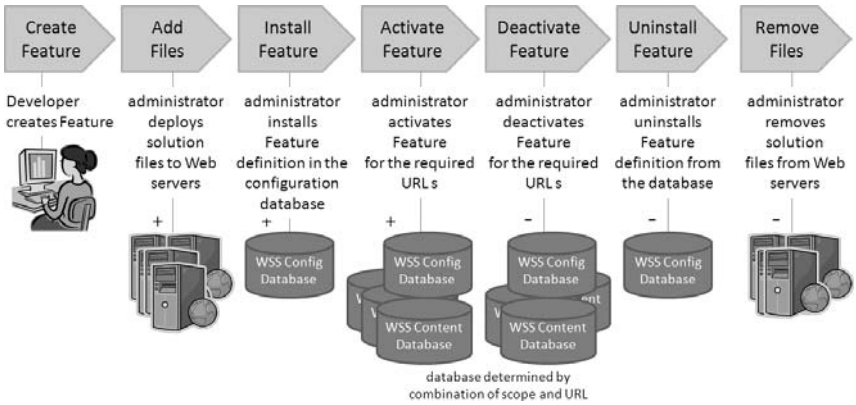


Figure 16.12
Feature lifecycle.

assembly. Let's step through the basic process of creating a Feature that adds a Hello World menu option to the Site Action menu.

1. Create a simple .aspx with the text "Hello World," save the file as HelloWorld.aspx, and copy it to a custom subfolder in the _layouts directory (e.g., _layouts/CustomApps/HellowWorld.aspx).
2. Navigate to the `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\FeatureS` directory and create a subfolder named HelloWorld.
3. Open the HelloWorld folder and create two text files named Feature.xml and AddMenu.xml
4. Open the newly created Feature.xml file and add the XML code listed in Figure 16.13. Replace the Feature ID with your own unique GUID. You can use Visual Studio's GUID Generator tool to create a new GUID. Once you have added the XML code, save the file.
5. Open the newly created AddMenu.xml file and add the XML code listed in Figure 16.14. Save the file.

That's all there is to it. The Feature is now ready to be installed and activated. The above steps described are the manual steps involved with creating a Feature; however, it should be noted that you are not restricted to develop Features using Notepad.exe. In fact, Visual Studio 2005 is the ideal environment for developing Features and custom application pages. When using Visual Studio, create a Feature folder structure in your project (\Feature\HelloWorld\), and either use a post build event to copy the files to the \Feature directory or use the Solution Deployment Framework. Using a post-build event to copy files is fine in a development environment; however, when working in produc-

Figure 16.13 *HelloWorld Feature.xml.*

```
<Feature Id="EEBB4B87-5AC7-4440-9DD4-C81576F07F93"
    Title="Add Hello World Menu"
    Description="Installs the &quot;Hello World&quot;
Link in Site Actions menu."
    Version="1.0.0.0"
    Scope="Site"
    xmlns="http://schemas.microsoft.com/sharepoint/">
    <ElementManifests>
        <ElementManifest Location="AddMenu.xml" />
    </ElementManifests>
</Feature>
```

Figure 16.14 *HelloWorld AddMenu.xml.*

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
    <CustomAction
        Id="AddHelloWorldLink"
        Location="Microsoft.SharePoint.StandardMenu"
        GroupId="SiteActions"
        Title="Hello World"
        Description="Display Hello World Application Page">
        <UrlAction Url="/_layouts/CustomApps/HelloWorld.aspx"/>
     </CustomAction>
</Elements>
```

tion it is recommended that you package your Features into a solution file. Once the Feature has been created and successfully deployed, all that remains is to install the Feature.

16.5.2 Installing a Feature

Features are installed using the `installFeature` operation of the stsadm.exe command line utility. Figure 16.15 shows the command required to install our HelloWorld Feature; you can specify either the name of the Feature folder, using the –`name` switch, or the path to the Feature.xml, file using the –`filename` switch.

Figure 16.15 *installFeature command.*

```
stsadm.exe -o installfeature -name HelloWorld
```

Upon installation, the Feature definition is added to the `Objects` table within the SharePoint configuration database. If you view the contents of the `Objects` table, you will see an object with the `Name` field set to "FeatureDefinition/<FeatureGUID>", where <FeatureGUID> is replaced with the GUID of the installed Feature (Figure 16.16). Viewing the corresponding properties of the object, you will see information associated with the Feature, such as the GUID of the solution if a solution was used to deploy the Feature.

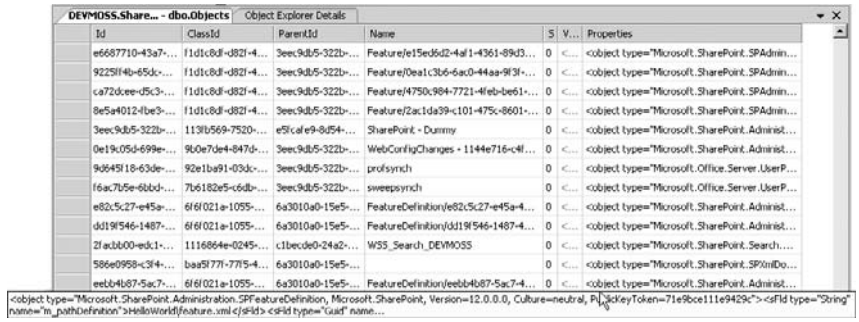


Figure 16.16 *Installing a Feature updates the configuration database.*

Once the Feature is installed, it is ready for activation. If you make updates to your Feature .XML files and need to reinstall the Feature, reset IIS and then reinstall your Feature with the `–force` switch at the end of the command line.

16.5.3 Activating a Feature

Features can be activated using either the stsadm.exe command line utility or through the SharePoint browser user interface. To activate and deactivate a Feature through the user interface, you must ensure that the `Hidden` Feature attribute in the Feature.xml file is set to `False`. The page from which a Feature is activated is dependent upon the scope of the Feature. Table 16.4 shows the location of the Feature management by scope. As mentioned earlier in the chapter, when you install a Feature scoped to the farm level, the Feature is activated automatically by default, unless the `ActivateOnDefault` Feature attribute is set to `False`.

Table 16.4 *Feature Management Pages*

Scope	Activation/Deactivation Page
Web	Site Features page, accessible from the Site Administration menu grouping on the Site Settings page for a Web site.
Site	Site Collection Features page, accessible from the Site Collection Administration menu grouping on the Site Settings page at the root of the site collection.
Web Application	Manage Web Application Features page, accessible from the Application Management page in the SharePoint Central Administration browser interface.
Farm	Manage Farm Features page, accessible from the Operations page in the SharePoint Central Administration browser interface.

The HelloWorld Feature that we are working with is scoped at the site collection level with the `Hidden` attribute set to `False`; therefore, we can activate this particular Feature from the Site Collection Features page (Figure 16.17). Alternatively we can use the `activateFeature` operation of stsadm.exe, as shown in Figure 16.18. The `activateFeature` operation accepts the name of the Feature or the path to the Feature.xml of the Feature in addition to the URL of the site in which you want the Feature activated. Given that our Feature is scoped at the site collection level, we will pass the activation command the URL to the root of the site collection.

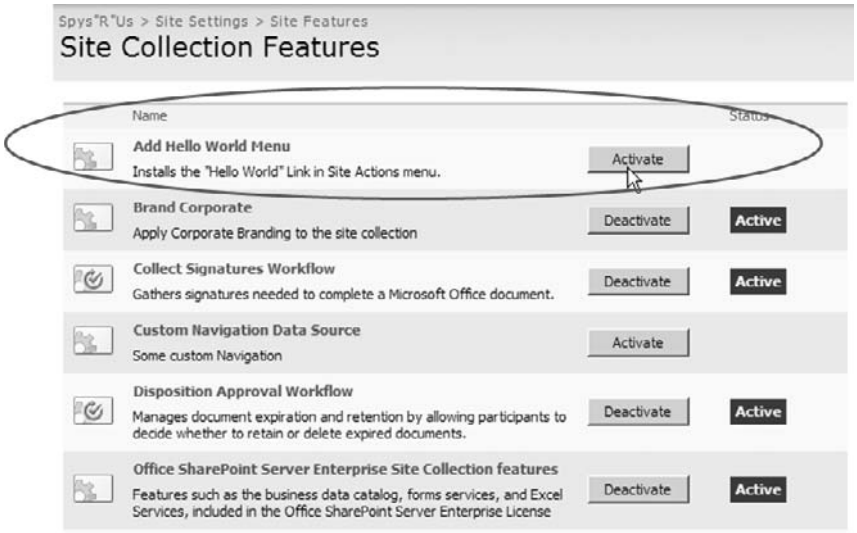


Figure 16.17 *Activate Feature using browser UI.*

```
stsadm.exe -o activatefeature -name HelloWorld -url http://devmoss/
```

Figure 16.18 *activateFeature command.*

It is important to remember that Features may be activated automatically, because of activation dependencies, or upon site creation if they are associated or stapled to a site definition.

When you activate a Feature, the database to which the Feature is added is determined by the scope and the destination URL of the activation command. For example, a Feature scoped to the site collection level is added to the `Features` table in the content database of the site collection in which it is activated. If, on the other hand, the Feature is scoped to the Web application level, then an entry is added to the `Objects` table in the farm's configuration database. An activated Feature scoped at the farm and Web application levels is represented in the configuration database with a Name field set to: `Fea-`

ture/<FeatureGUID>/<ParentID>, where the <FeatureGUID> is replaced with the GUID of the installed Feature. An activated Feature scoped at the site collection and Web site levels is represented in the content database. Table 16.5 shows the Feature database entries depending on scope, and Figure 16.19 shows the entry added to the content database for the activated HelloWorld Feature; the `SiteID` field is populated with the identifier of the site collection in which the Feature was activated. If the Feature was scoped to the Web site level, the `WebId` field would be populated with the identifier of the Web site for which the Feature was activated.

Table 16.5 *Database entries for Activated Features*

Scope	Location	Field Settings
Farm	Objects table within the Configuration database	`Name` field is set to: `Feature/<FeatureGUID>/00000000-0000-0000-0000-000000000000`
WebApplication	Objects table within the Configuration database	`Name` field is set to: `Feature/<FeatureGUID>/<WebAppGUID>`
Site	Features table within the content database in which the specified site collection resides	`FeatureId` field is set to the Feature GUID
Web	Features table within the content database in which the specified Web site resides	`FeatureId` field is set to the Feature GUID

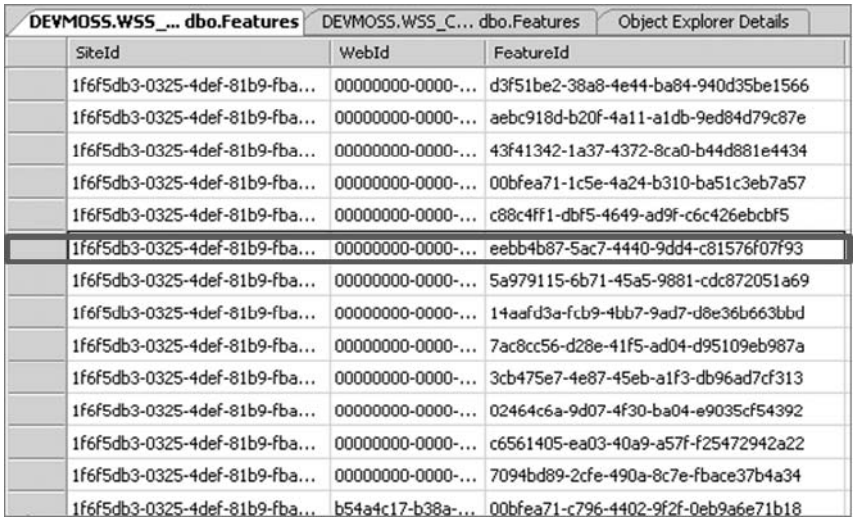


Figure 16.19 *Activating a Feature adds an entry to the content database.*

Once the Feature is activated, the functionality provided by the Feature is available. In our HelloWorld example, this means that the Site Actions menu should have an additional action at the bottom of the menu (Figure 16.20).

Figure 16.20 *Hello World Action on Site Actions Menu.*

16.5.4 Deactivating a Feature

Features can be deactivated using either the stsadm.exe command line utility or through the SharePoint browser user interface. Once again, if you want to be able to deactivate a Feature through the user interface, you must ensure that the `Hidden` Feature attribute in the Feature.xml file is set to `False`. The page from which a Feature is deactivated is dependent upon the scope of the Feature (Table 16.4). Figure 16.21 shows the Deactivate option for our HelloWorld Feature. Alternatively, we can use the `deactivateFeature` operation of stsadm.exe, as shown in Figure 16.22. The `deactivateFeature` operation accepts the name of the Feature or the path to the Feature.xml of the Feature in addition to the URL of the site in which you want the Feature deactivated. We pass the deactivation command the URL that we used to activate the Feature in the root of the site collection.

When you deactivate a Feature, the database entry which was added during activation is removed. For example, if we deactivated the HelloWorld Feature, the database row highlighted in Figure 16.19 would be removed from the content database. If the Feature you are deactivating depends upon

Figure 16.21 *Deactivate Feature using browser UI.*

```
stsadm.exe -o deactivatefeature -name HelloWorld -url http://devmoss/
```

Figure 16.22 *deactivateFeature command.*

another Feature, and no additional Features depend upon that Feature, then both Features are automatically deactivated.

Additionally, when you deactivate a Feature, the functionality that the Feature provided is removed from the area in which the Feature was deactivated. For example, when we deactivate our HelloWorld Feature, the Hello World action disappears from the Site Actions menu.

16.5.5 Uninstalling a Feature

Features are uninstalled using the `uninstallFeature` operation of the stsadm.exe command line utility. Figure 16.23 shows the command required to uninstall our HelloWorld Feature; you can specify either the name of the Feature folder, using the `–name` switch, or the path to the Feature.xml file, using the `–filename` switch.

```
stsadm.exe -o uninstallfeature -name HelloWorld
```

Figure 16.23 *uninstallFeature command.*

When a Feature is uninstalled, the Feature definition is removed from the `Objects` table within the SharePoint configuration database.

16.6 Feature Stapling

Feature stapling allows you to associate, or staple, Features to an existing site definition or custom site definition. For example, if you had a custom Feature that set the master page, the cascading style sheet, and the theme to custom settings, and you wanted these settings to apply when a site was created using a specific site definition, then you would use a Feature staple to staple your custom Feature to that site definition. A Feature staple is itself a

Feature; it comprises a Feature folder, a Feature definition file, and an element manifest file. The element manifest file uses the `FeatureSiteTemplateAssociation` element to associate the Feature with a specific site definition. The `FeatureSiteTemplateAssociation` element has two attributes: `Id` and `TemplateName`. The `Id` attribute must map to the ID of an existing Feature, and the `TemplateName` attribute is used to specify the site definition configuration (site template) to which the Feature will be stapled. Figure 16.24 shows an example of a Feature staple that staples a branding Feature to the site SpysRUs site template. If you want to make your Feature available to all newly created sites, regardless of the site definition from which they were created, you can staple your Feature to the GLOBAL site definition. Figure 16.25 shows an updated version of the element manifest used in our previous example; the `TemplateName` has now been replaced with the GLOBAL setting. Note that you do not need to specify a configuration number when stapling to the GLOBAL site definition.

Figure 16.24
Feature stapling.

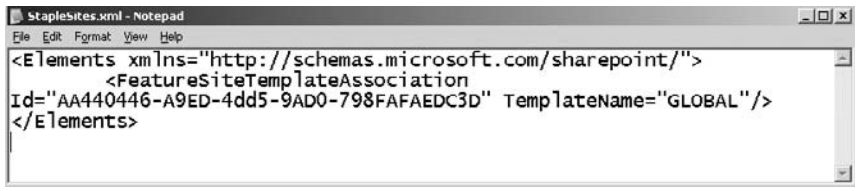


Figure 16.25
StapleSites.xml

As you can see, Feature stapling allows you to staple a Feature to existing site definitions, even the typically untouchable Microsoft-generated site definitions. This ability provides an extremely easy way of adding additional functionality and customizations without having to reproduce an entire site definition. Feature stapling provides easier manageability and maintainabil-

ity and increases the likelihood of surviving future upgrades and service pack releases.

16.7 Deploying Features using the Solution Framework

The WSS 3.0 solution framework provides a method for collating the various components that comprise a solution and packaging them together into a solution file. The solution file is basically a cabinet-based format, with a .wsp extension instead of a .cab extension. Once the solution file is created, the solution file can then be used to deploy the contained functionality to a SharePoint farm. Chapter 11 introduced the concept of the solution framework and described several techniques that you can use to create a .wsp file. If your solution file requires internal directories, you must use the `MakeCAB.exe` technique, because Microsoft Visual Studio .NET does not support creating cabinet files with internal directories. When working with Features and custom applications in Visual Studio, you create a folder hierarchy that mimics the `TEMPLATE\FeatureS` and `TEMPLATE\LAYOUTS` folder hierarchy, which means that you will be working with internal directories; therefore, the `MakeCAB.exe` technique is the method you should use to create your solution files. Building a solution file using the `MakeCAB.exe` involves first creating a Feature project, then adding the solution manifest file to the project, copying the files to the SharePoint directories, creating a .ddf file, and then running MakeCAB.exe to create the solution file (.wsp). The solution file can then be deployed to the SharePoint farm using stsadm.exe. Let's walk through each of these steps using our Hello World example:

1. Using Visual Studio 2005 create a new empty project.
2. Create two folders at the root of the project, named TEMPLATE and WSP.
3. Add two subfolders to the TEMPLATE folder, named FeatureS and LAYOUTS.
4. Open the FeatureS folder and add another subfolder named HelloWorld
5. Open the HelloWorld folder and add two XML files named Feature.xml and AddMenu.xml
6. Add the XML listed in Figure 16.13 to the Feature.xml file, replacing the GUID with your own unique identifier. You can use the Visual Studio Create GUID tool for this.
7. Add the XML listed in Figure 16.14 to the AddMenu.xml file.

8. Open the LAYOUTS folder and add another subfolder named CustomApps.
9. Open the CustomApps folder and create a new Windows Form (.aspx page) with some simple Hello World text. Save the file as HelloWorld.aspx.
10. Navigate to the root of the solution and create a new XML file named manifest.xml.
11. Add the XML listed in Figure 16.26 to the manifest.xml, replacing the `SolutionId` GUID with your own unique identifier. Once again, you can use the Visual Studio Create GUID tool for this.

Figure 16.26
Manifest.xml.

12. At the root of the solution, add a new text file and rename the file HelloWorld.ddf.
13. Add the text listed in Figure 16.27 to the HelloWorld.ddf file.

Figure 16.27
HelloWorld.ddf.

The files used in the Feature should be referenced by stating both the source path and the destination path to the files.

14. At this point, the content of your Visual Studio project should be similar to that shown in Figure 16.28.

Figure 16.28 *Visual Studio Project Layout*

15. Navigate to the root of the solution and create an install.bat file.
16. Add the code listed in Figure 16.29 to the install.bat file.

Figure 16.29 *Install.bat*

17. From the Project menu, click <project> Properties.
18. Navigate to the Build Events tab and enter the commands shown in Figure 16.30 in the post-build event command line.
19. Save and build the project.
20. Exit Visual Studio.

The install.bat file copies all the Feature and custom application files to their appropriate directory in SharePoint and creates the solution file using

Figure 16.30 *Post-build event commands*

`MakeCAB.exe`. Once that task is complete, the install.bat file adds the solution file to the solution store, using the `addsolution` operation of `stsadm`. The solution has not yet been deployed. You can deploy the file either by using `stsadm deploysolution` operation or through the central administration user interface; in fact, we could have added the `deploysolution` command to the install.bat file. Deploying a solution that contains a Feature ensures that all the Feature files are deployed on the appropriate servers within the server farm and also adds the appropriate entries to the configuration database by performing the same function as running the `installFeature stsadm` command. Once a Feature is deployed, it is available for activation at the defined scope. Retracting a solution that implements a Feature has the same effect as running the `uninstallFeature stsadm` command; the Feature definition entry is removed from the configuration database.

16.8 Branding Using Features Exercise

In this chapter, we have explained the inner workings of Features and walked you through the entire Feature lifecycle. Now let's take a look at how Features can help us in the world of branding. Chapter 14, Branding your SharePoint 2007 Sites explained how pages are rendered in WSS 3.0 and SharePoint Server 2007 and described how master pages, page layouts, themes, and style sheets can all be leveraged when branding your SharePoint sites. The branding mechanism used in the examples in Chapter 14 leverages the content management capabilities of SharePoint Server 2007, allowing us to change the brand of our site by publishing newer versions of a master page and pushing these changes down to all sites in the site collection through the master page settings page in the browser (Figure 16.31). This version control capability in SharePoint Server 2007 is great for deploying branding changes to a single site collection and especially great for development environments when you are creating your master pages. However, if you have multiple site collections in your deployment, it becomes a very time-consuming process to roll out branding in this manner as you would have to upload and publish the master page to each site collection individually. Additionally, it is difficult to ensure that new sites generated using the default Microsoft site templates will automatically adopt the new look and feel. Chapter 15, Site Definitions and Templates showed us

how to associate a custom master page with a new SharePoint site that was provisioned using a custom site template. We can of course apply custom master pages to the default site templates in the same manner; however, we end up modifying the default Microsoft files, which consequently places us in unsupported territory and puts our customizations at risk of being overwritten with future updates and service pack releases. So what do we do? How *can* we deploy a brand to our entire SharePoint deployment in a manner that offers flexibility and efficiency when applying updates? Let's examine how Features can help us overcome this particular challenge.

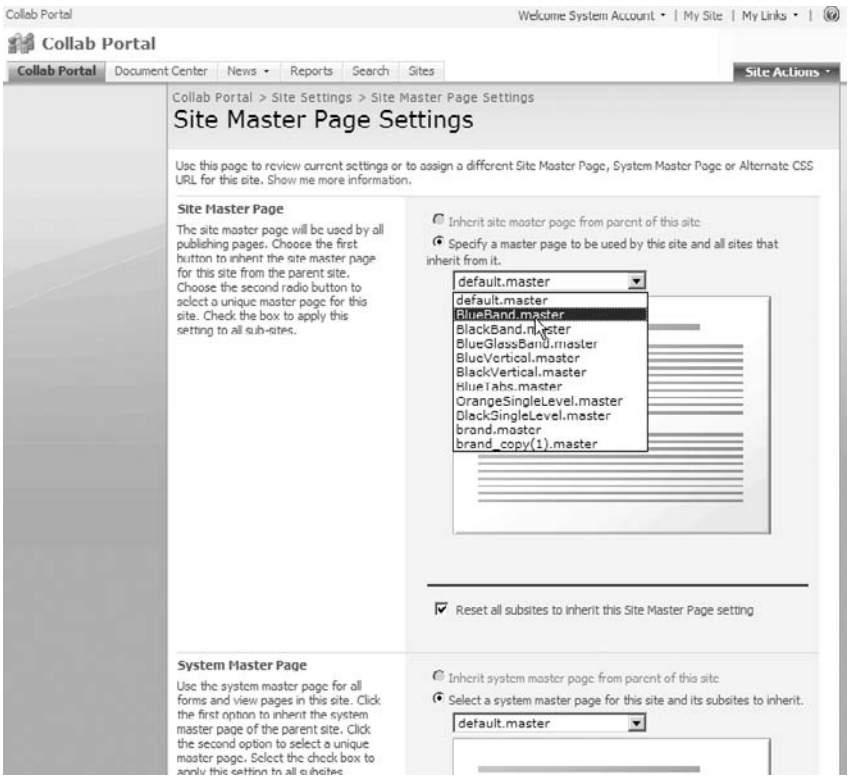


Figure 16.31
Master page setting using browser user interface.

16.8.1 Branding Scenario

The fictitious company SpysRUs would like a method of being able to update its portal branding, depending on the season. For example, if it is Christmas, they would like the look and feel of their sites to reflect a Christmas theme, if it is St. Patrick's Day, they would like the look and feel to reflect an Irish theme, and if it's just a regular day they would like the look and feel to reflect the standard corporate SpysRUs brand. They would like this capability to be available in all existing and future sites, and they want

the ability to use the default Microsoft site templates upon site creation. We need to implement an *easy* way to turn on and turn off specific brands and have the branding apply to all existing and future sites.

To implement the base components of our branding scenario, we need to create four Features scoped at the site collection level: Brand, Brand1Corp, Brand2Irish and Brand3Xmas. The Brand Feature provisions all the files needed for branding the portal, in much the same way that the default PublishingLayouts Feature provisions the out-of-the-box Microsoft branding files. The remaining Features are responsible for applying the brand specific to that Feature. For example, upon activation, the Brand2Irish Feature changes the branding to reflect a St. Patrick's Day look and feel. Given that the Brand1Corp, Brand2Irish, and Brand3Xmas Features all depend on the files provisioned by the Brand Feature being in place, the ActivationDependency setting for each of these Features is set to Brand. Figure 16.32 shows the Feature folders for our branding scenario. In addition to the four base branding Features, we have two additional Features: BrandInherit and BrandStaple. The BrandInherit Feature is scoped to the Web site level, and upon activation, it forces the current Web site to inherit its branding settings from its parent. This Feature is hidden, and therefore it cannot be activated from the browser user interface. By default, when a site is created in SharePoint, it does *not* inherit its default branding settings, such as the theme, the master page settings, and the site logo setting from the parent site. Therefore, the purpose of the BrandInherit Feature is to allow all newly created sites to inherit branding from the settings of the parent site, regardless of the site template used for creation. For example, if our Brand3Xmas Feature is activated in a site collection, and a new site is created using the default Microsoft Team Site site template, then upon activation of the BrandInherit Feature, the new site inherits the settings from the parent site. We use a final Feature, named BrandStaple, to staple the BrandInherit Feature to the GLOBAL site definition, thereby automatically activating the BrandInherit Feature upon site creation for *any* newly created site.

The Brand Feature is comprised of a number of subfolders that contain the branding files to be provisioned to document libraries within SharePoint. Figure 16.33 shows the contents of the Brand Feature, Figure 16.34 shows the Feature definition file, Feature.xml, and Figure 16.35 shows the contents of the element manifest, ProvisionFiles.xml. The Feature provisions the contents of the Images folder to the site collection images document library, the contents of the StyleSheets folder to the style library, and the contents of the MasterPages folder to the master page gallery. The Feature is a hidden Feature; therefore, it must be activated using the activateFeature stsadm command.

The Brand1Corp Feature is comprised of a single Feature definition file, Feature.xml (Figure 16.36). The definition file contains a reference to a Feature receiver that is used to handle Feature events, such as the activation of

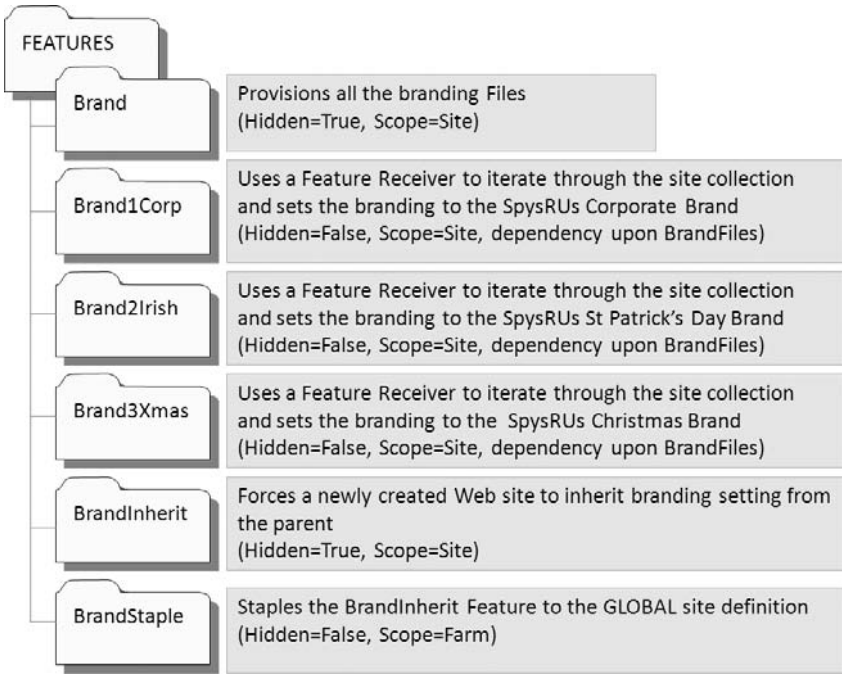


Figure 16.32
Branding Feature folders.

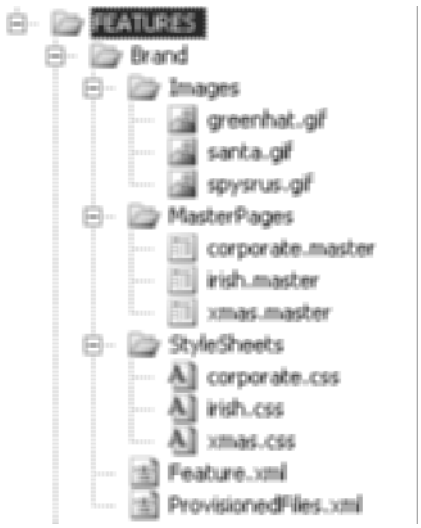


Figure 16.33
Brand Feature files and folders.

the Feature. The file also details the activation dependency that the Feature has upon the Brand Feature.

Figure 16.34
Brand Feature.xml.

```
<!-- _lcid="1033" _version="12.0.4518" _dal="1" -->
<!-- _LocalBinding -->
<Feature  Id="E82C5C27-E45A-4cd7-8B5C-17CA6B940779"
          Title="Provision Brand Files"
          Description="Feature installs all the master pages, css and image files for all brands"
          AlwaysForceInstall="TRUE"
          Scope="Site"
          Hidden="true"
          xmlns="http://schemas.microsoft.com/sharepoint/" >
    <ElementManifests>
        <ElementManifest Location="ProvisionedFiles.xml"/>
    </ElementManifests>
</Feature>
```

Figure 16.35
Brand ProvisionFiles.xml.

```
<!-- _lcid="1033" _version="12.0.4518" _dal="1" -->
<!-- _LocalBinding -->
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
    <Module Name="EmerBrandMasterPages" List="116" Url="_catalogs/masterpage" Path="MasterPages" RootWebOnly="TRUE">
        <File Url="corporate.master" Type="GhostableInLibrary" />
        <File Url="irish.master" Type="GhostableInLibrary" />
        <File Url="xmas.master" Type="GhostableInLibrary" />
    </Module>
    <Module Name="EmerBrandStyleSheets" Url="Style Library" Path="StyleSheets" RootWebOnly="TRUE">
        <File Url="corporate.css" Type="GhostableInLibrary" />
        <File Url="irish.css" Type="GhostableInLibrary" />
        <File Url="xmas.css" Type="GhostableInLibrary" />
    </Module>
    <Module Name="EmerBrandImages" Url="SiteCollectionImages" Path="Images" RootWebOnly="TRUE">
        <File Url="spysrus.gif" Type="GhostableInLibrary" />
        <File Url="greenhat.gif" Type="GhostableInLibrary" />
        <File Url="santa.gif" Type="GhostableInLibrary" />
    </Module>
</Elements>
```

Brand2Irish and Brand3Xmas are also comprised of a single Feature definition file that leverages the same Feature receiver and has the same activation dependency as the Brand1Corp Feature; however, their property values differ, causing the Feature receiver to set different branding for each Feature. This is a great example of the power and versatility of Features. The Feature.xml files for Brand2Irish and Brand3Xmas are shown in Figure 16.37 and Figure 16.38, respectively. Figure 16.39 shows the code for the ChangeBranding class of the Feature receiver. The ChangeBranding class retrieves the branding properties defined in the Feature definition file, iterates the properties through all the sites in the site collection, and sets the master page,

Figure 16.36
Brand1Corp Feature.xml.

```
<Feature  Id="6E79812F-7530-4b22-93B6-ADD717624F3A"
          Title="Brand Corporate"
          Description="Apply Corporate Branding to the site collection"
          AlwaysForceInstall="true"
      Version="1.0.0.0"
          Scope="Site"
          Hidden="false"
          xmlns="http://schemas.microsoft.com/sharepoint/"
      ReceiverAssembly="BrandingReceiver, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=46b7247bceb41436"
      ReceiverClass="BrandingReceiver.ChangeBranding" >
    <ActivationDependencies>
      <ActivationDependency FeatureId="1AEA5653-7628-4d73-B9D0-6C7C9A564F00" />
    </ActivationDependencies>
    <Properties>
        <Property Key="activateMaster" Value="_catalogs/masterpage/corporate.master"/>
        <Property Key="activateTheme" Value="Obsidian"/>
        <Property Key="activateCSS" Value="StyleLibrary/corporate.css"/>
        <Property Key="activateLogo" Value="SiteCollectionImages/spysrus.gif"/>
</Properties>
</Feature>
```

the theme, the cascading style sheet, and the site logo to the corresponding property value.

Figure 16.37
Brand2Irish Feature.xml.

```
<Feature  Id="5D911E1E-1B53-4bba-9BF2-B53C93AB55BB"
          Title="Brand St Patricks Day"
          Description="Apply St Patricks Day Branding to the site collection"
          AlwaysForceInstall="true"
      Version="1.0.0.0"
          Scope="Site"
          Hidden="false"
          xmlns="http://schemas.microsoft.com/sharepoint/"
      ReceiverAssembly="BrandingReceiver, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=46b7247bceb41436"
      ReceiverClass="BrandingReceiver.ChangeBranding" >
    <ActivationDependencies>
      <ActivationDependency FeatureId="1AEA5653-7628-4d73-B9D0-6C7C9A564F00" />
    </ActivationDependencies>
    <Properties>
        <Property Key="activateMaster" Value="_catalogs/masterpage/irish.master"/>
        <Property Key="activateTheme" Value="Verdant"/>
        <Property Key="activateCSS" Value="StyleLibrary/irish.css"/>
        <Property Key="activateLogo" Value="SiteCollectionImages/greenhat.gif"/>
    </Properties>
</Feature>
```

Figure 16.38
Brand3Xmas Feature.xml.

```
<Feature  Id="8B5F6021-F7F5-4ef2-93A9-8FAA2DB73C39"
          Title="Brand Christmas"
          Description="Apply Christmas Branding to the site collection"
          AlwaysForceInstall="true"
      Version="1.0.0.0"
          Scope="Site"
          Hidden="false"
          xmlns="http://schemas.microsoft.com/sharepoint/"
      ReceiverAssembly="BrandingReceiver, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=46b7247bceb41436"
      ReceiverClass="BrandingReceiver.ChangeBranding" >
    <ActivationDependencies>
      <ActivationDependency FeatureId="1AEA5653-7628-4d73-B9D0-6C7C9A564F00" />
    </ActivationDependencies>
    <Properties>
        <Property Key="activateMaster" Value="_catalogs/masterpage/xmas.master"/>
        <Property Key="activateTheme" Value="Cardinal"/>
        <Property Key="activateCSS" Value="StyleLibrary/xmas.css"/>
        <Property Key="activateLogo" Value="SiteCollectionImages/santa.gif"/>
    </Properties>
</Feature>
```

Figure 16.39
ChangeBranding class code (C#).

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

namespace BrandingReceiver
{
    class ChangeBranding:SPFeatureReceiver
    {
        public override void FeatureActivated(SPFeatureReceiverProperties properties)
        {
            SPFeaturePropertyCollection propColl = properties.Feature.Properties;
            SPFeatureProperty propA = propColl["activateMaster"];
            SPFeatureProperty propB = propColl["activateTheme"];
            SPFeatureProperty propC = propColl["activateCSS"];
            SPFeatureProperty propD = propColl["activateLogo"];
            string activateMaster = propA.Value;
            string activateTheme = propB.Value;
            string activateCSS = propC.Value;
            string activateLogo = propD.Value;
            SPSite siteColl = (SPSite)properties.Feature.Parent;
            SPWebCollection webColl = siteColl.AllWebs;
            foreach (SPWeb spWeb in webColl)
            {
                spWeb.MasterUrl = siteColl.ServerRelativeUrl + activateMaster;
                spWeb.CustomMasterUrl = siteColl.ServerRelativeUrl + activateMaster;
                spWeb.ApplyTheme(activateTheme);
                spWeb.AlternateCssUrl = siteColl.ServerRelativeUrl + activateCSS;
                spWeb.SiteLogoUrl = siteColl.ServerRelativeUrl + activateLogo;
                spWeb.Update();
            }
        }
        public override void FeatureDeactivating(SPFeatureReceiverProperties properties)
        {
        }
        public override void  FeatureInstalled(SPFeatureReceiverProperties properties)
        {
        }
        public override void FeatureUninstalling(SPFeatureReceiverProperties properties)
        {
        }
    }
}
```

When the Brand1Corp is activated, it automatically activates the Brand Feature, if not already activated, because it is dependent upon it and scoped at the same level. The Brand1Corp Feature then applies the corporate branding to all the sites in the site collection, as shown in Figure 16.40.

When the Brand2Irish is activated, it automatically activates the Brand Feature, if not already activated, because it too is dependent upon it and scoped at the same level. The Brand2Irish Feature then applies the St. Patrick's Day branding to all the sites in the site collection, as shown in Figure 16.41.

When Brand3Xmas is activated, it automatically activates the Brand Feature, if not already activated, because it too is dependent upon it and scoped at the same level. The Brand3Xmas Feature then applies the Christmas Day branding to all the sites in the site collection, as shown in Figure 16.42. You can customize the branding of all the Features by manipulating the values of the Feature property bag.

Figure 16.40 *Corporate branding Feature activated.*

Figure 16.41 *St Patrick's Day Branding Feature Activated*

Figure 16.42 *Christmas Branding Feature Activated*

If we were to install the Feature set at this stage new sites would not pick up the branding that was applied at the parent. You could deactivate and activate one of the branding Features previously mentioned, which would cause the branding to be pushed down to all sites, thereby handling any new sites that had been created since the last time the Feature was activated; however, this is not a suitable workaround, as you would need to be able to determine the instant a site was created and deactivate/activate every time. A better method is to create another Feature scoped at the Web site level that, upon activation, runs code to tell the newly created site to inherit the branding from the parent. Figure 16.43 shows the Feature definition file for the BrandInherit Feature.

Figure 16.43 *BrandInherit Feature.xml*

```
<Feature  Id="465DAC2B-90DF-4efc-B11B-3EC898C1D64B"
          Title="Brand Inheritance"
          Description="Inherit Branding from parent web"
          AlwaysForceInstall="true"
      Version="1.0.0.0"
          Scope="Web"
          Hidden="true"
          xmlns="http://schemas.microsoft.com/sharepoint/"
      ReceiverAssembly="BrandingReceiver, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=46b7247bceb41436"
      ReceiverClass="BrandingReceiver.InheritSettings" >
</Feature>
```

Once again, this Feature contains a single Feature.xml file that references the same BrandingReceiver assembly referenced by Brand1Corp, Brand2Irish and Brand3Xmas; however, the notable difference is that BrandInherit is scoped at a different level, is set to, hidden *and* uses the InheritSettings class within the BrandingReceiver assembly, as opposed to the ChangeBranding class. Figure 16.44 shows the code used to implement the InheritSetting class. The code performs a similar function to the ChangeBranding code, except in this instance we retrieve the branding settings from the parent site instead of the Feature property bag, and we change the settings only for the current Web site.

We need a method of ensuring that the BrandInherit Feature is activated upon site instantiation, regardless of the site template used at the time of site creation. The BrandStaple Feature performs precisely this function by associating the BrandInherit Feature with the GLOBAL site definition. By using the GLOBAL site definition in our Feature stapling, we ensure that all existing and future site definitions will pick up the BrandInherit Feature. The BrandStaple Feature consists of a Feature definition file, Feature.xml, and an element manifest file, StapleSites.xml. Figure 16.45 shows the XML for the Feature definition, and Figure 16.46 shows the XML for the element manifest. If you examine the element manifest, you will notice that it contains a single FeatureSiteTemplateAssociation element whose Id attribute is set to the GUID of the BrandInherit Feature, and TemplateName attribute is set to

Figure 16.44
InheritSettings class code (C#).

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

namespace BrandingReceiver
{
    class InheritSettings : SPFeatureReceiver
    {
        public override void FeatureActivated(SPFeatureReceiverProperties properties)
        {
            SPWeb spWeb = (SPWeb)properties.Feature.Parent;
            spWeb.MasterUrl = spWeb.ParentWeb.MasterUrl;
            spWeb.CustomMasterUrl = spWeb.ParentWeb.CustomMasterUrl;
            spWeb.ApplyTheme(spWeb.ParentWeb.Theme);
            spWeb.AlternateCssUrl = spWeb.ParentWeb.AlternateCssUrl;
            spWeb.SiteLogoUrl = spWeb.ParentWeb.SiteLogoUrl;
            spWeb.Update();
        }

        public override void FeatureDeactivating(SPFeatureReceiverProperties properties)
        {
        }

        public override void FeatureInstalled(SPFeatureReceiverProperties properties)
        {
        }

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

```

GLOBAL, thus stapling the BrandInherit Feature to all site definitions. If you wanted to staple the Feature to a specific site definition, all that you would need to do is change the TemplateName to reference the site definition folder and specific configuration, for example STS#0. Note that when referencing the GLOBAL site definition you do not need to specify the configuration number. The BrandStaple Feature is scoped to the farm level, and therefore activates automatically by default. The BrandStaple Feature completes our branding Feature set required to ensure that a consistent brand is deployed throughout all the sites in a site collection. Ensuring consistency throughout the entire farm requires activating the Features at the appropriate levels throughout. Feature activation can be handled by a simple script.

In the next section, we will take you through the steps to set up our branding scenario.

16.8.2 Create the Feature Receiver

1. Using Visual Studio 2005, create a new C# class project named BrandingReceiver.
2. Add a reference to the Microsoft.SharePoint.dll.
3. Create two class files: ChangeBranding.cs and InheritSettings.cs.

Figure 16.45 *BrandStaple Feature.xml.*

```
<Feature  Id="BFFA1778-EC54-4aa9-96E4-A5C331C886B8"
          Title="Brand Stapling Feature"
          Description="Staples the Brand Inheritance Feature to all site
definitions "
      ActivateOnDefault="true"
          AlwaysForceInstall="true"
      Version="1.0.0.0"
          Scope="Farm"
          Hidden="false"
          xmlns="http://schemas.microsoft.com/sharepoint/">
    <ElementManifests>
       <ElementManifest Location="StapleSites.xml"/>
    </ElementManifests>
</Feature>
```

Figure 16.46 *BrandStaple StapleSites.xml.*

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
    <FeatureSiteTemplateAssociation Id="465DAC2B-90DF-4efc-B11B-3EC898C1D64B"
TemplateName="GLOBAL" />
</Elements>
```

4. Add the code listed in Figure 16.39 to ChangeBranding.cs.
5. Add the code listed in Figure 16.44 to InheritSettings.cs.
6. From the Projects menu, open the Project Properties dialog.
7. Click the Signing tab and sign the assembly, creating a new strong name key file.
8. Save and build the project.
9. Drag and drop the assembly DLL into the global assembly cache (WINDOWS\assembly); alternatively, use stsadm to install the assembly into the GAC.
10. Navigate to the newly installed assembly in the GAC and review the assembly properties. Take note of the assembly information, such as the version and the Public Key Token—this will be used when defining the receiver in the Feature set.
11. Close the project in Visual Studio.

Note that the Feature receiver should be deployed to all Web front-end servers; ideally, you would create a solution package for the Feature receiver and manage the deployment centrally.

16.8.3 Create the Branding Feature Set

Next, we'll create the branding Feature set, and we will leverage the solution framework to ease deployment of the Features. When following this exercise,

it is assumed that you have already created the appropriate style files, such as images, style sheets, and master pages.

1. Using Visual Studio 2005, create a new empty project.
2. Create two folders at the root of the project, named TEMPLATE and WSP.
3. Add one subfolder to the TEMPLATE folder named FeatureS.
4. Open the FeatureS folder and add six subfolders, named Brand, Brand1Corp, Brand2Irish, Brand3Xmas, BrandInherit, and BrandStaple.
5. Open the Brand folder and add three subfolders, named Images, MasterPages, and StyleSheets.
6. Open the Images folder and add three graphic files, named greenhat.gif, santa.gif, and spysrus.gif.
7. Open the MasterPages folder and add three master pages: corporate.master, irish.master, and xmas.master.
8. Open the StyleSheets folder and add three style sheets: corporate.css, irish.css and xmas.css.
9. Open the Brand folder and add two XML files named Feature.xml and ProvisionedFiles.xml.
10. Add the XML listed in Figure 16.34 to the Feature.xml file, replacing the GUID with your own unique identifier. You can use the Visual Studio Create GUID tool for this.
11. Add the XML listed in Figure 16.35 to the ProvisionedFiles.xml file.
12. Open the Brand1Corp folder and add an XML file named Feature.xml.
13. Add the XML listed in Figure 16.36 to the Feature.xml file, replacing the GUID with your own unique identifier.
14. Open the Brand2Irish folder and add an XML file named Feature.xml.
15. Add the XML listed in Figure 16.37 to the Feature.xml file, replacing the GUID with your own unique identifier.
16. Open the Brand3Xmas folder and add an XML file named Feature.xml.
17. Add the XML listed in Figure 16.38 to the Feature.xml file, replacing the GUID with your own unique identifier.

18. Open the BrandInherit folder and add an XML file named Feature.xml.
19. Add the XML listed in Figure 16.43 to the Feature.xml file, replacing the GUID with your own unique identifier.
20. Open the BrandStaple folder and add two XML files named Feature.xml and StapleSites.xml.
21. Add the XML listed in Figure 16.45 to the Feature.xml file, replacing the GUID with your own unique identifier.
22. Add the XML listed in Figure 16.46 to the StapleSites.xml file, replacing the GUID with your own unique identifier.
23. Navigate to the root of the solution and create a new XML file named manifest.xml.
24. Add the XML listed in Figure 16.47 to the manifest.xml, replacing the `SolutionId` GUID with your own unique identifier. Once again, you can use the Visual Studio Create GUID tool for this.

Figure 16.47 *Branding manifest.xml.*

```
<Solution SolutionId="A1ACE6F5-D16E-4150-9E39-519BF807839F"
          xmlns="http://schemas.microsoft.com/sharepoint/">
    <FeatureManifests>
        <FeatureManifest Location="Brand\Feature.xml"/>
        <FeatureManifest Location="Brand1Corp\Feature.xml"/>
        <FeatureManifest Location="Brand2Irish\Feature.xml"/>
        <FeatureManifest Location="Brand3Xmas\Feature.xml"/>
        <FeatureManifest Location="BrandInherit\Feature.xml"/>
        <FeatureManifest Location="BrandStaple\Feature.xml"/>
    </FeatureManifests>
</Solution>
```

25. At the root of the solution, add a new text file and rename the file Branding.ddf.
26. Add the text listed in Figure 16.48 to the Branding.ddf file. The files used in the Feature should be referenced by stating both the source path and the destination path to the files.
27. At this point, the content of your Visual Studio project should be similar to that shown in Figure 16.49.
28. Navigate to the root of the solution and create an install.bat file.
29. Add the code listed in Figure 16.50 to the install.bat file.
30. From the Project menu, click <project> Properties.

Figure 16.48
Branding.ddf

```
; *** Deploy Feature **
.OPTION EXPLICIT
.set CabinetNameTemplate=BrandingFeature.wsp
.set DiskDirectoryTemplate=CDROM
.Set CompressionType=MSZIP
.Set UniqueFiles="ON"
.Set Cabinet=on
.set DiskDirectory1=WSP
manifest.xml
TEMPLATE\FEATURES\Brand\Feature.xml Brand\Feature.xml
TEMPLATE\FEATURES\Brand\ProvisionedFiles.xml Brand\ProvisionedFiles.xml
TEMPLATE\FEATURES\Brand\Images\greenhat.gif Brand\Images\greenhat.gif
TEMPLATE\FEATURES\Brand\Images\santa.gif Brand\Images\santa.gif
TEMPLATE\FEATURES\Brand\Images\spysrus.gif Brand\Images\spysrus.gif
TEMPLATE\FEATURES\Brand\MasterPages\irish.master Brand\MasterPages\irish.
master
TEMPLATE\FEATURES\Brand\MasterPages\xmas.master Brand\MasterPages\xmas.master
TEMPLATE\FEATURES\Brand\MasterPages\corporate.master Brand\MasterPages\
corporate.master
TEMPLATE\FEATURES\Brand\StyleSheets\irish.css Brand\StyleSheets\irish.css
TEMPLATE\FEATURES\Brand\StyleSheets\xmas.css Brand\StyleSheets\xmas.css
TEMPLATE\FEATURES\Brand\StyleSheets\corporate.css Brand\StyleSheets\corporate.
css
TEMPLATE\FEATURES\Brand1Corp\Feature.xml Brand1Corp\Feature.xml
TEMPLATE\FEATURES\Brand2Irish\Feature.xml Brand2Irish\Feature.xml
TEMPLATE\FEATURES\Brand3Xmas\Feature.xml Brand3Xmas\Feature.xml
TEMPLATE\FEATURES\BrandInherit\Feature.xml BrandInherit\Feature.xml
TEMPLATE\FEATURES\BrandStaple\Feature.xml BrandStaple\Feature.xml
TEMPLATE\FEATURES\BrandStaple\StapleSites.xml BrandStaple\StapleSites.xml
;*** the end
```

31. Navigate to the Build Events tab and enter the commands shown in Figure 16.51 in the post-build event command line.
32. Save and build the project.
33. Exit Visual Studio.
34. All that remains is for you to activate your branding Features.
35. Navigate to the Site Collection Features page. You should see three branding Features in an inactive state (Figure 16.52).
36. Activate the Corporate Branding Feature.
37. Navigate through your site hierarchy and notice that the branding has been pushed to all sites and subsites throughout the collection.
38. Create a new site. Notice how the branding for the new site is inherited from the parent.
39. Navigate back to the Site Collection Features page.
40. Activate the Xmas Branding Feature and traverse the site collection to see the changes.
41. Create another new site and, once again, notice how the branding for the new site is inherited from the parent.

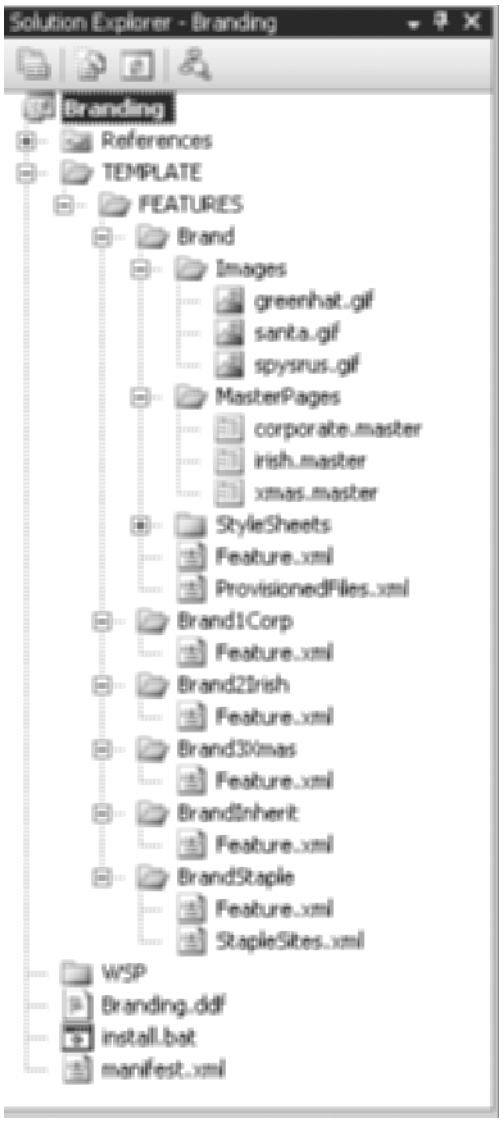


Figure 16.49 *Branding project layout.*

```
1  @SET TEMPLATEDIR="c:\program files\common files\microsoft
   shared\web server extensions\12\TEMPLATE"
2
3  xcopy /e /y TEMPLATE\* %TEMPLATEDIR%
4  makecab.exe /f Branding.ddf
5  stsadm.exe -o addsolution -filename WSP\BrandingFeature.wsp
```

Figure 16.50 *Branding Install.bat.*

42. Navigate back to the Site Collection Features page.

Figure 16.51
Post-build event commands.

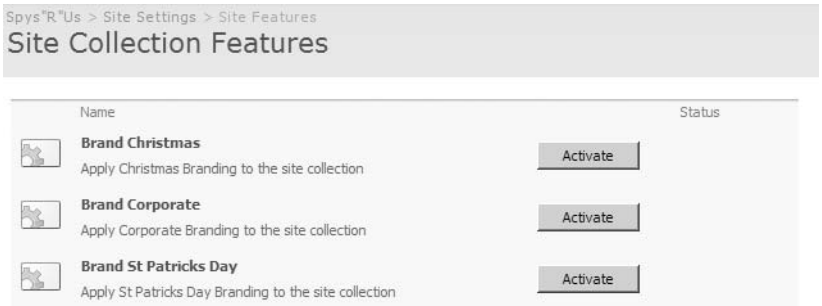


Figure 16.52
Branding Features.

43. Activate the St. Patrick's Day Branding Feature and traverse the site collection to see the changes.

44. Create another new site and, once again, notice how the branding for the new site is inherited from the parent.

Throughout this chapter, we have discussed the basic concepts and components involved when creating Features. Features provide an easy method for adding and removing functionality and customizations to and from your SharePoint sites. In Chapter 14, Branding your SharePoint 2007 Sites, our branding exercise showed us how to create our own custom master page and style sheet and apply it to all the sites in our site collection. However, we still had an issue, in that newly created sites did not pick up our new branding. In this chapter, we showed how Features, and in particular Feature stapling, can be used to overcome this limitation. The power and versatility of Features combined with the solution framework provides organizations with a rich development and administrative model for customizing and extending their SharePoint deployments.

This page intentionally left blank

17

Events

SharePoint Portal Server 2001 shipped out of the box with approval and routing functionality that could be extended further using the Web Storage System event sinks. The term "event sink" is used to describe the application code that is triggered when an event occurs. With the move to SharePoint Portal Server 2003, the underlying architecture and storage platform changed from the Web Storage System to Windows SharePoint Services 2.0 and Microsoft SQL Server. Surprisingly, out-of-the-box functionality, such as the approval and routing capabilities, disappeared; however, developers were still able to implement custom workflows by writing custom event handlers and binding them to a document library.

We have mentioned numerous times throughout this book that SharePoint Server 2007 and WSS 3.0 introduce an extensive set of new functionalities and customization capabilities to the SharePoint suite of products, and therefore it will come as no surprise for you to hear that these new capabilities extend to the event architecture. In this chapter, we take a quick look at the improvements made to the event architecture and show you how you can leverage events in your own applications.

17.1 What is an Event?

An event is a way to identify when a specific action occurs, such as when an item is added to a SharePoint list. An event handler is code that is used to handle the event once the event fires. For example, you can create an event handler that populates the properties of a document once it has been added to a document library. With WSS 2.0, we were fairly limited in what we could actually do with events; you were only able to bind asynchronous events to document libraries, and you were restricted to one event sink per document library. The fact that only asynchronous events were supported meant that you could not stop an action from happening; you could only be notified that it had happened. For example, there was no way to prevent an item that was being processed through a workflow from being deleted; the

only way to restore the item was to reinsert it into the library yourself. That said, you were passed the before and after states with your event handler, which allowed you to compare the state of a document in the document library before it was modified with the state after it was modified. Those of you familiar with the event architecture that came with SharePoint Portal Server 2001, courtesy of the Web Storage System, may have been frustrated by the fact that document library events were asynchronous only with SharePoint Portal Server 2003. Thankfully, Microsoft has redeemed themselves by giving the event architecture a considerable makeover in WSS 3.0.

The event architecture in WSS 3.0 provides many improvements over its predecessors, including support for synchronous events and asynchronous events, the ability to capture events from a variety of sources, and the ability to register multiple event handlers with a SharePoint container, such as a document library or list. Synchronous events refer to those events that fire *before* a specific action occurs, and asynchronous events refer to those events that fire *after* a specific action occurs. With synchronous events, the normal flow of the event is stopped until your event handler finishes processing its code; furthermore, you have the ability to cancel the event, resulting in no post events being executed. Synchronous events are also referred to as preprocessing events. Asynchronous events do not affect the normal flow of the event; the event continues processing while your code executes, and the page displays immediately. Asynchronous events are also referred to as postprocessing events. This is great news to developers, as now you are able to manipulate items before and after they have been added to a SharePoint container. For example, you can register an event handler with a specific document library that only allows members of a specific group the ability to create folders in that document library. The event handler would trap the synchronous event before the folder is created in the document library and, depending on the user's group membership, either create the folder or cancel the creation process. Assuming the folder was created, you could add additional postprocessing logic—for example, updating the audit log or sending an e-mail to notify the appropriate people that a new folder has been created. The scope of events has greatly improved in that we are no longer limited to a single document library; instead, we can trap events from document libraries, lists, Web sites, site collections, files, received e-mails, workflows, content types, or even individual fields in a list. For example, you can create an event handler to lock down a list so that end users cannot add additional columns, or you can create an event handler that detects when a user tries to delete a Web site. The logic in the event handler could then trigger a series of actions that include creating a backup of the site *before* actually deleting the site.

17.1.1 Event Receivers

WSS 3.0 offers a set of event receiver classes that are specifically designed to receive events triggered from a variety of sources. Each receiver class has a set of methods that you can override in your event handler. We'll walk you through the process of creating an event handler later in this chapter, but it basically entails creating a class that inherits an event receiver base class, such as `SPItemEventReceiver`, and overriding the available methods as needed. For example, the `SPItemEventReceiver` class includes an `ItemAdding` method, which is a synchronous event that occurs when a user adds an item to a SharePoint container object. And because the `ItemAdding` method represents a synchronous event, we can leverage this method in our event handler to perform a specific action *before* the item is added to the SharePoint container. Figure 17.1 shows the definition of the `SPItemEventReceiver` class, which includes a list of all the methods available to override. Notice that the class derives from the `SPEventReceiverBase` class.

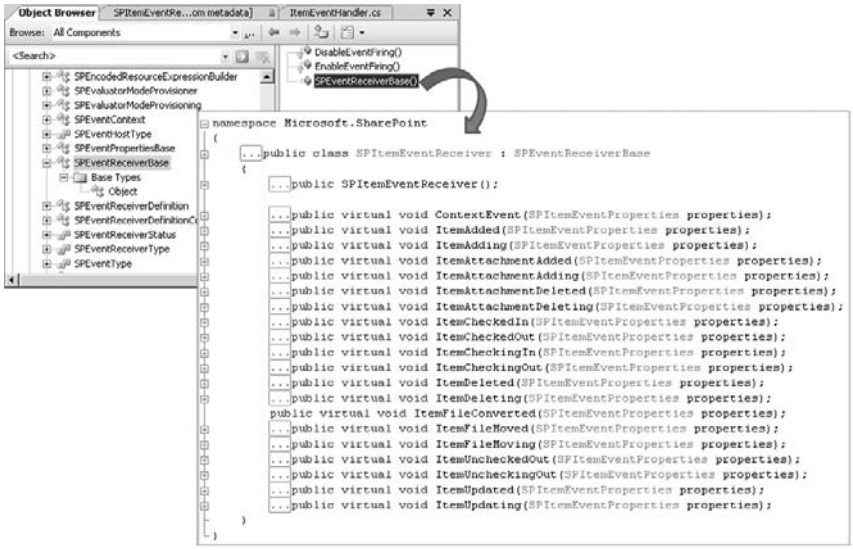


Figure 17.1 *SPItemEventReceiver class.*

WSS 3.0 provides three event receiver classes that derive from the `SPEventReceiverBase` class: `SPWebEventReceiver`, `SPItemEventReceiver`, and `SPListEventReceiver`. The base class provides two base methods, `DisableEventFiring` and `EnableEventFiring`, which can be leveraged in your event handler logic for any of the three aforementioned event receivers. When coding your event handlers, you may run into the situation in which you need to update the properties associated with the item that you are handling, but you do not want to cause an Updating and Updated event to fire. In this case, you would call the `DisableEventFiring` method prior to calling the

`Update` method for the item and then, after the update has been made, call the `EnableEventFiring` method to allow normal event handling to resume.

In addition to the methods offered by the base class, each class offers its own set of methods for handling asynchronous and synchronous events. As a general rule, methods ending in "ing" represent synchronous events (for example, `ItemAdding`), and methods ending in "ed" represent asynchronous events (for example, `ItemAdded`). Each method provides access to a list of properties associated with the event, such as the login details for the current user. The list of available properties varies depending on the event receiver class being leveraged. For example, the properties associated with the `SPItemEventReceiver` class include a `ListTitle` property, which returns the title of the list to which the event handler is currently registered; needless to say, the properties associated with the `SPWebEventReceiver` class do not include a `ListTitle` property. Let's examine each of these classes in more detail.

17.1.2 SPWebEventReceiver

The `SPWebEventReceiver` class provides a set of methods for handling events that occur to site collections and Web sites, such as deleting or moving a Web site. These methods are listed in Table 17.1, along with a brief description of each. Note that the `SPWebEventReceiver` class does not provide a method for

Table 17.1 *SPWebEventReceiver Methods*

Method	Description
SiteDeleting	Synchronous (preprocessing) event that is triggered when a site collection is *being* deleted.
SiteDeleted	Asynchronous (postprocessing) event that is triggered *after* a site collection has been deleted.
WebDeleting	Synchronous (preprocessing) event that is triggered when a Web site is *being* deleted.
WebDeleted	Asynchronous (postprocessing) event that is triggered *after* a Web site has been deleted.
WebMoving	Synchronous (preprocessing) event that is triggered when a Web site is *being* renamed or moved to a different parent object.
WebMoved	Asynchronous (postprocessing) event that is triggered *after* a Web site has been moved.

trapping the creation of Web sites. If you need this functionality you can leverage Features by creating a Feature receiver containing the custom code

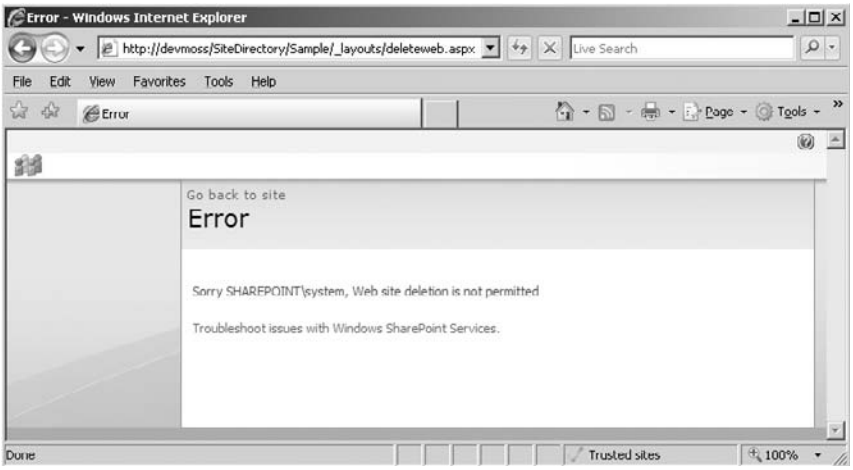
that you would like to run and then stapling this Feature to the site definition(s) associated with the new site. Chapter 16, Features, shows an example of how features can be used to apply branding to a site upon site creation.

Synchronous events provide the capability to cancel the event and display an error message to the user indicating why the event was canceled. Let's assume that we want to disable the deletion of Web sites in our implementation, regardless of the user's permission levels. We can achieve this functionality by creating an event handler that inherits the `SPWebEventReceiver` class and overrides the `WebDeleting` method to cancel the event. The code required is very simple and shown in Figure 17.2, and the resulting page that the user sees when they try to delete a Web site is shown in Figure 17.3.

Figure 17.2
Overriding the WebDeleting method (C#).

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

namespace MyEventHandler
{
    public class WebEventHandler : SPWebEventReceiver
    {
        public override void WebDeleting(SPWebEventProperties properties)
        {
            string curuser = properties.UserLoginName;
            properties.Cancel = true;
            string errmsg = "Sorry " + curuser + ", Web site deletion is not permitted";
            properties.ErrorMessage = errmsg;
        }
    }


}
```

Figure 17.3
Web site deletion error message.



Each method listed in Table 17.2 provides access to a set of Web event properties (`SPWebEventProperties`). This is illustrated by the `properties` parameter definition in line 11 of the code listing shown in Figure 17.2. The properties available in the `SPWebEventProperties` class are listed in Figure 17.2 and include many properties that you will find useful in your code. For example, the `UserLoginName` property provides access to the username information for the user that triggered the event, which can be quite useful for determining permissions or auditing events. In our code example, we use the `UserLoginName` property when creating the error message to display to the user. The error page produced shows that the user that triggered the event was the SharePoint System Account, which further illustrates that with a simple `properties.Cancel` operation we are able to prevent even the highly privileged System Account from being able to delete a Web site.

Table 17.2 *SPWebEventProperties*

Property	Description
EventType	Gets the type of event that occurred, such as `WebDeleting`.
FullUrl	Gets the absolute URL of the Web site in which the event was triggered.
NewServerRelativeUrl	When trapping the `WebMoved` event, this property gets the server relative URL of the new location to which the Web site has moved.
SiteId	Gets the GUID of the site collection in which the event was triggered.
UserDisplayName	Gets the display name of the user who triggered the event, for example, "System Account" or "Rita Smith".
UserLoginName	Gets the login name of the user who triggered the event in the format of DOMAIN\\Username. For example "DEV\\Rita.Smith"
Web	Gets the Web object for the Web site in which the event was triggered.
WebId	Gets the GUID of the Web site in which the event was triggered.

Table 17.2 *SPWebEventProperties (continued)*

Property	Description
Cancel	Gets or sets a boolean value that determines whether the event is cancelled. `properties.Cancel=true` means that the current event is cancelled. The `Cancel` property is only available to synchronous (pre-processing) events, such as `WebDeleting, WebMoving and SiteDeleting`. The `Cancel` property inherits from the `SPEventBaseProperties` class
ErrorMessage	Gets or sets the error message that SharePoint displays to the user when an event is cancelled. The `ErrorMessage` property is only available to synchronous (pre-processing) events, such as `WebDeleting`, `WebMoving` and `SiteDeleting`. The `ErrorMessage` property inherits from the `SPEventBaseProperties` class
ReceiverData	Returns a string that contains data about the event.
Status	Gets or sets the status of the event. For example, if the event sets the `Cancel` property to true then the status property will change from `Continue` to `CancelWithError`. The `Status` property inherits from the `SPEventBaseProperties` class

Figure 17.4 shows the Autos pane in Visual Studio 2005 that represents a custom event handler executing the code listed in Figure 17.2. This should give you an idea of the data that is available for use when writing your event handlers. The Type column shows you the data type for each property. For example, the Web property returns the `Microsoft.SharePoint.SPWeb` object for the current site, which you can use directly in your event handler logic. As we proceed through this chapter, we will provide examples of how to leverage these properties in your code.

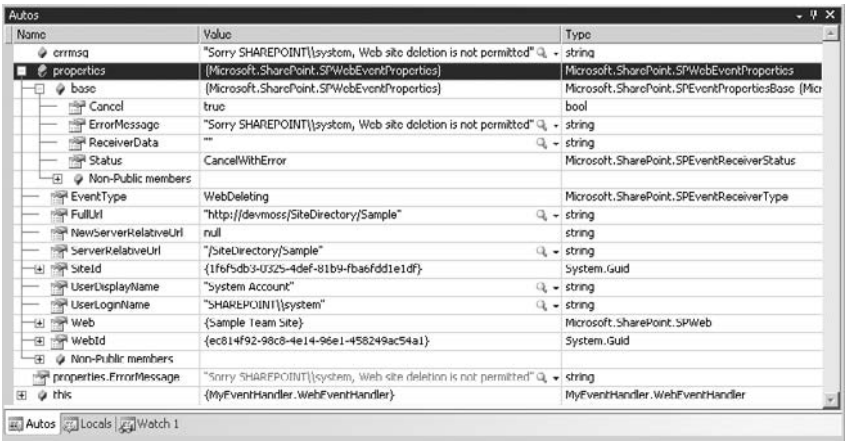


Figure 17.4 *SPWebEventProperties sample values.*

17.1.3 SPItemEventReceiver

The `SPItemEventReceiver` class provides a set of methods for handling events that occur to SharePoint items, such as adding an attachment, deleting an item or checking in an item. These methods are listed in Table 17.3, along with a brief description of each. Many of the methods are self explanatory, for example the `ItemAdded` method fires when an item is added to a SharePoint list to which the event handler is registered, and the `ItemDeleted` method fires when an item is deleted from a list to which the event handler is registered.

Table 17.3 *SPItemEventReceiver Methods*

Method	Description
ItemAdding	A synchronous (preprocessing) event that is triggered when an item is *being* added to a SharePoint container, such as a list or document library. This event fires when new items are created through the browser interface, through the object model, or through a Web service.
ItemAdded	Asynchronous (postprocessing) event that is triggered *after* an item has been added to a SharePoint container, such as a list or document library.
ItemAttachmentAdding	Synchronous (preprocessing) event that is triggered when an attachment is *being* added to an item.
ItemAttachmentAdded	Asynchronous (postprocessing) event that is triggered *after* an attachment has been added to an item.

Table 17.3 *SPItemEventReceiver Methods (continued)*

Method	Description
ItemAttachmentDeleting	Synchronous (preprocessing) event that is triggered when an attachment is *being* deleted from an item.
ItemAttachmentDeleted	Asynchronous (postprocessing) event that is triggered *after* an attachment has been deleted from an item.
ItemCheckingIn	Synchronous (preprocessing) event that is triggered when an item is *being* checked in to a SharePoint container, such as a document library.
ItemCheckedIn	Asynchronous (postprocessing) event that is triggered *after* item has been checked in to a SharePoint container, such as a document library.
ItemCheckingOut	Synchronous (preprocessing) event that is triggered when an item is *being* checked out of a SharePoint container, such as a document library.
ItemCheckedOut	Asynchronous (postprocessing) event that is triggered *after* item has been checked out of a SharePoint container, such as a document library.
ItemDeleting	Synchronous (preprocessing) event that is triggered when an item is *being* deleted.
ItemDeleted	Asynchronous (postprocessing) event that is triggered *after* an item has been deleted from a SharePoint container.
ItemFileMoving	Synchronous (preprocessing) event that is triggered when an item is *being* moved.
ItemFileMoved	Asynchronous (postprocessing) event that is triggered *after* an item has been moved from a SharePoint container.
ItemUncheckingOut	Synchronous (preprocessing) event that is triggered when an item's checkout is *being* discarded.
ItemUncheckedOut	Asynchronous (postprocessing) event that is triggered *after* item's checkout has been discarded.
ItemUpdating	Synchronous (preprocessing) event that is triggered when an item is *being* updated.
ItemUpdated	Asynchronous (postprocessing) event that is triggered *after* an item has been updated from a SharePoint container.

Once again, the synchronous events provide the capability to cancel the event and display an error message to the user indicating why the event was cancelled. Let's assume that we want to restrict the ability to create folders to the members of a specific SharePoint group. We can achieve this functionality by creating an event handler that inherits the `SPWebItemReceiver` class and overrides the `ItemAdding` method. Figure 17.5 shows some sample code that implements this functionality. Within the `ItemAdding` method we retrieve the content type of the item (lines 18-23), and if it is a folder, we check to see if the current user is a member of the appropriate group (lines 29-33). If the user is not a member of the group, we simply cancel the event and display a message to the user informing them that they do not have the appropriate permissions to perform the function.

Figure 17.5
Overriding the ItemAdding method (C#).

```
12        public override void ItemAdding(SPItemEventProperties properties)
13        {
14            //We want to prevent unauthorized users from creating folders in the
15            //Document Library.  They can add documents.
16            //We can check the content type by inspecting the AfterProperties of the Item
17
18            string curContentType = "Folder";
19            SPItemEventDataCollection aprops = properties.AfterProperties;
20            if (aprops["ContentType"] != null)
21            {
22                curContentType = aprops["ContentType"].ToString();
23            }
24
25            using (SPSite curSite = new SPSite(properties.WebUrl))
26            {
27                using (SPWeb curWeb = curSite.OpenWeb())
28                {
29                    if ((curContentType == "Folder") | (curContentType == "Classified Folder"))
30                    {
31                        bool ismember = curWeb.IsCurrentUserMemberOfGroup(curWeb.Groups["Classified Users"].ID);
32                        string errmsg = "You are currently logged on as: " + properties.UserLoginName;
33                        if (ismember == false)
34                        {
35                            properties.Cancel = true;
36                            errmsg = errmsg + "You do not have the appropriate permissions to add: " +
curContentType + " content from this document library";
37                            properties.ErrorMessage = errmsg;
38                        }
39                    }
40                }
41            }
42        }
```

In this example, we leverage the item event properties returned by the `SPItemEventProperties` object and listed in Table 17.4. These properties are accessible to each method in the `SPItemEventReceiver` class. You'll notice that many of the properties listed for the item event were also listed in the Web event properties. For example, the `UserLoginName` property is accessible to both property sets and provides access to the username information for the user that triggered the event. In our code example, we use the

`UserLoginName` property once again when creating the error message to display to the user. Other properties that the item event receiver introduces are specific to items.

Table 17.4 *SPItemEventProperties*

Property	Description
AfterProperties	Gets a hash table of properties consisting of key/value pairs that map to the properties of the triggering `SPItem` object *after* the event has occurred.
AfterUrl	Gets the URL of the item *after* the event has occurred.
BeforeProperties	Gets a hash table of properties consisting of key/value pairs that map to the properties of the triggering `SPItem` object *before* the event occurred.
BeforeUrl	Gets the URL of the item *before* the event has occurred.
CurrentUserId	Gets the `ID` of the user that triggered the event.
EventType	Gets the type of event that occurred, such as `ItemAdding`.
ListId	Gets the `GUID` of the list in which the event was triggered.
ListItem	Gets the `SPListItem` object for the item that triggered the event.
ListItemId	Gets the `ID` of the list item that triggered the event.
ListTitle	Gets the title of the lists in which the event was triggered.
RelativeWebUrl	Gets the server-relative URL for the Web site in which the event was triggered.
SiteId	Gets the `GUID` of the site collection in which the event was triggered.
UserDisplayName	Gets the display name of the user that triggered the event—for example, "System Account" or "Rita Smith".
UserLoginName	Gets the login name of the user that triggered the event in the format of DOMAIN\\Username. For example "DEV\\Rita.Smith".
WebUrl	Gets the absolute URL for the Web site in which the event was triggered.
Zone	Gets the URL zone for the request.

Table 17.4 *SPItemEventProperties (continued)*

Property	Description
Cancel	Gets or sets a Boolean value that determines whether the event is cancelled. `properties.Cancel=true` means that the current event is cancelled. The `Cancel` property is only available to synchronous (preprocessing) events, such as `ItemAdding`, `ItemUpdating`, and `ItemDeleting`. The `Cancel` property inherits from the `SPEventBaseProperties` class
ErrorMessage	Gets or sets the error message that SharePoint displays to the user when an event is cancelled. The `ErrorMessage` property is only available to synchronous (pre-processing) events, such as `ItemAdding`, `ItemUpdating`, and `ItemDeleting`. The `ErrorMessage` property inherits from the `SPEventBaseProperties` class
ReceiverData	Returns a string that contains data about the event.
Status	Gets or sets the status of the event. For example, if the event sets the `Cancel` property to `true`, then the status property will change from `Continue` to `CancelWithError`. The `Status` property inherits from the `SPEventBaseProperties` class

An extremely useful property is the `ListItem` property, which provides you with access to the item object that triggered the event. Once you have the object, you can manipulate the properties as necessary. However, it is important to keep in mind that the availability of the properties varies depending on the method accessing the property and how the event was actually triggered. For example, the `ListItem` and `ListId` properties are set to null and 0 respectively when accessed from the `ItemAdding` method. If you think about it, this makes perfect sense, because the item has not officially been added to the list; therefore, an object doesn't really exist as yet. Once the item has been added to the list, it is written to the content database, which in turn triggers the ItemAdded event. Now, if you access the `ListItem` and `ListId` properties from the `ItemAdded` method, in most cases you will find that both properties are populated. What do we mean by *most* cases? Well, one would expect the `ListItem` and `ListId` properties to be set at all times in the `ItemAdded` method, but unfortunately that's not the case. The `ListItem` and `ListItemId` properties are set to null and 0 respectively when you create a folder through the Web Explorer view, which is very

inconvenient when you are trying to write code that supports both the browser and Web Explorer scenarios. Interestingly enough, when you add a *document* to a document library through the Web Explorer view, the `ListItem` and `ListId` properties are populated as expected and available for use in your code.

Figure 17.6 shows the Auto pane in Visual Studio 2005 that represents a custom event handler executing the code listed in Figure 17.5. This should give you an idea of the data that is available for use when writing your event handlers. Once again, the Type column shows you the data type for each property—for example, the `ListId` property returns the `GUID` for the list that contains the item that triggered the event, which you can use directly in your event handler logic. Notice that the `ListItem` and `ListItemId` properties are set to null and 0, as discussed in the previous paragraph.

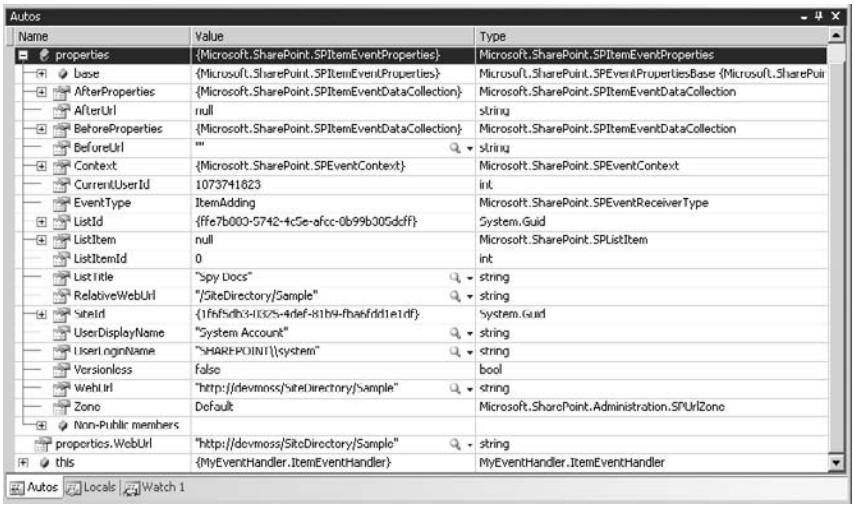


Figure 17.6 *SPItemEvent Properties sample values.*

So, how do you access properties associated with the item when overriding the `ItemAdding` method? The answer lies in the use of another property named `AfterProperties`. The `AfterProperties` and `BeforeProperties` properties each return a hash table of properties consisting of key/value pairs that map to the properties of the triggering `SPItem` object *before* and *after*, respectively, the event occurs. This is great news for our particular example. Remember, we are trying to determine whether the user is creating a folder object and whether they have the appropriate permissions to do so. To determine the type of item being created, we need access to the content type of the item. Each item has a `ContentType` property that contains this information, and we can access the `ContentType` property directly through the `AfterProperties` property set.

Once again, the availability of the properties varies depending on the method accessing the property and how the event was actually triggered. If you refer back to our `ItemAdding` code listing in Figure 17.5 you will notice that the logic to capture the content type of the item happens in lines 18 through 23. As part of this logic, we initially set the content type to `Folder`, which means that if we cannot access the content type property, then we should err on the side of caution and assume the user is trying to add a folder. In what scenario would the content type property be inaccessible? If you initiate the `ItemAdding` event through the Web Explorer view, the `AfterProperties` property returns an empty hashtable, which means that you cannot access properties such as the content type property. Figure 17.7 shows the `AfterProperties` for a folder that was added through the browser; notice that the `ContentType` field is populated. Figure 17.8 shows the `AfterProperties` for a folder that was added through the Web Explorer view; notice that the `ContentType` field is *not* available in this instance. The main point we are making here is that when writing your event code, you need to ensure your code accommodates the erroneous behavior experienced when working in Web Explorer view.

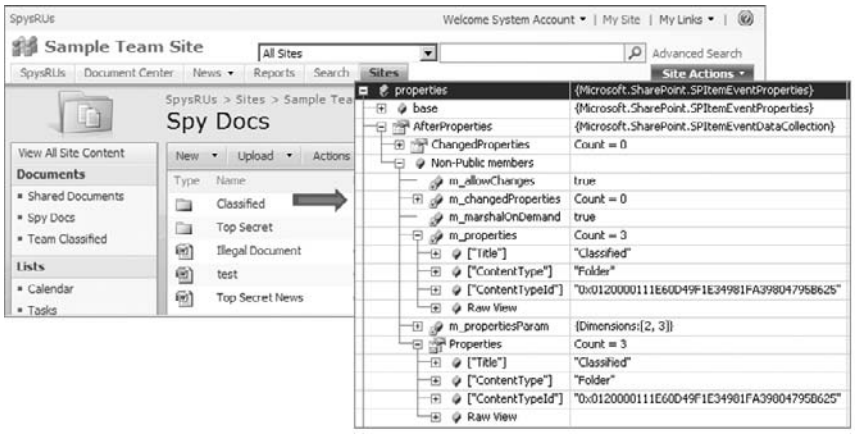


Figure 17.7
Browser folder item AfterProperties.

The actual properties that are returned in the `AfterProperties` and `BeforeProperties` also vary depending on the item type. For example, a folder object returns three `AfterProperties` properties, whereas a Microsoft Office Word 2007 document returns thirteen `AfterProperties` properties. Figure 17.7 shows the folder properties returned, and Figure 17.9 shows the document properties returned.

An easy way to iterate through the properties in the `AfterProperties` and `BeforeProperties` hashtables is to use the `DictionaryEntry` object, as shown in the code listing in Figure 17.10. The code listing returns the key/value pair for each returned property.

Figure 17.8 *Web Explorer folder item AfterProperties.*

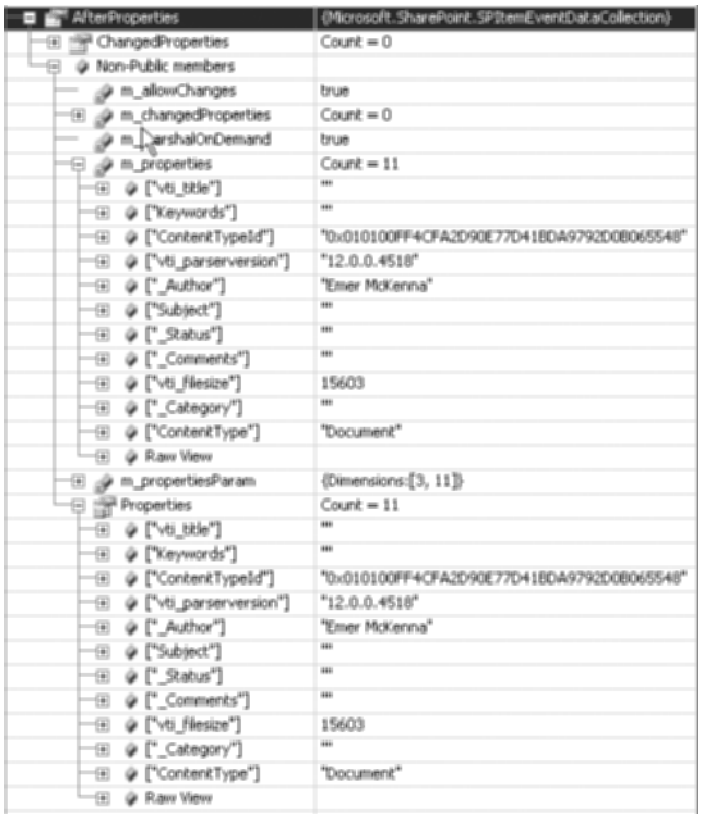


Figure 17.9 *Document AfterProperties.*

```
foreach (DictionaryEntry de in properties.AfterProperties)
{
    string strKey = de.Key.ToString();
    string strValue = de.Value.ToString();
}
```

Figure 17.10 *Iterating through AfterProperties Hashtable*

17.1.4 SPListEventReceiver

The `SPListEventReceiver` class provides a set of methods for handling events that occur to SharePoint lists, such as adding or deleting a column. These methods are listed in Table 17.5, along with a brief description of each. Many of the methods are self-explanatory; for example, the `FieldAdded` method fires when a column is added to a SharePoint list for which the event handler is registered, and the `ItemDeleted` method fires when a column is deleted from a list for which the event handler is registered.

Table 17.5 *SPListEventReceiver Methods*

Method	Description
FieldAdding	Synchronous (preprocessing) event that is triggered when a field (column) is *being* added to a SharePoint list.
FieldAdded	Asynchronous (postprocessing) event that is triggered *after* a field (column) has been added to a SharePoint list.
FieldDeleting	Synchronous (preprocessing) event that is triggered when a field (column) is *being* deleted from a SharePoint list.
FieldDeleted	Asynchronous (postprocessing) event that is triggered *after* a field (column) has been deleted from a SharePoint list.
FieldUpdating	Synchronous (preprocessing) event that is triggered when a field (column) is *being* updated in a SharePoint list.
FieldUpdated	Asynchronous (postprocessing) event that is triggered *after* a field (column) has been updated in a SharePoint list.

Yet again, the synchronous events provide the capability of canceling the event and displaying an error message to the user indicating why the event was cancelled. Let's assume that we want to lock down a SharePoint list so that columns cannot be added or removed from the list. We can achieve this functionality by creating an event handler that inherits the `SPListEventReceiver` class and overrides the `FieldAdding` and `FieldDeleting` methods. Figure 17.11 shows some sample code that implements this functionality. Notice that the code within each method is similar to that used in our previous examples. Within the `FieldAdding` method, we define our error message, set the

`ErrorMessage` property, and then set the `Cancel` property to `true` (lines 10-15), which in turn displays the message shown in Figure 17.12 to the user when he or she tries to add a field to the list.

Each method passes a `SPListEventProperties` parameter that gives you access to properties associated with the `SPListEventReceiver`. The available properties are listed in Table 17.6.

Figure 17.11 *Overriding the FieldAdding and FieldDeleting methods (C#).*

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using Microsoft.SharePoint;
5
6  namespace MyEventHandler
7  {
8      public class ListEventHandler : SPListEventReceiver
9      {
10         public override void FieldAdding(SPListEventProperties properties)
11         {
12             string errmsg = properties.UserLoginName + " this list is locked, ↵
       you cannot add or remove additional columns.";
13             properties.ErrorMessage = errmsg;
14             properties.Cancel = true;
15         }
16
17         public override void FieldDeleting(SPListEventProperties properties)
18         {
19             string errmsg = properties.UserLoginName + " this list is locked, ↵
       you cannot add or remove additional columns.";
20             properties.ErrorMessage = errmsg;
21             properties.Cancel=true;
22
23         }
24     }
25 }
```

Figure 17.12 *FieldAdding error message.*

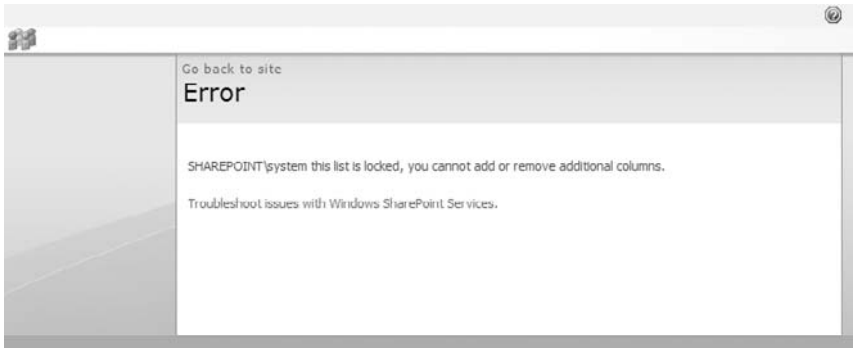


Figure 17.13 shows the Auto pane in Visual Studio 2005 that represents a custom event handler executing the code listed in Figure 17.11. This should give you an idea of the data available for use when writing your list event handlers. Once again, the Type column shows you the data type for each property. For example, the `Field` property returns the `SPField` object

Table 17.6 *SPListEventProperties*

Property	Description
Field	Gets the `SPField` object that triggered the list event.
FieldName	Gets the name of the field that triggered the list event.
FieldXml	Gets the XML definition of the field that triggered the list event.
List	Gets the `SPList` object of the list in which the event was triggered.
ListId	Gets the `GUID` of the list in which the event was triggered.
ListTitle	Gets the title of the list in which the event was triggered
SiteId	Gets the `GUID` of the site collection that contains the list in which the event was triggered.
UserDisplayName	Gets the display name of the user that triggered the event—for example, "System Account" or "Rita Smith".
UserLoginName	Gets the login name of the user that triggered the event in the format of DOMAIN\\Username. For example "DEV\\ Rita.Smith".
Web	Gets the `SPWeb` object of the Web site that contains the list in which the event was triggered.
WebId	Gets the `GUID` of the Web site that contains the list in which the event was triggered.
WebUrl	Gets the URL of the Web site that contains the list in which the event was triggered.

of the field that triggered the event, which you can use directly in your event handler logic. Notice that the `EventType` is set to `FieldDeleting`, which means that a user has attempted to delete a field in a list to which our event handler is registered. The base `SPListEventProperties` `Cancel` property is set to `true`, the `ErrorMessage` property is set to our custom error message, and the `Status` property is set to `CancelWithError`, which in turn matches the logic in our code listing shown in Figure 17.11(lines 19 thru 21). These settings cause SharePoint to cancel the delete attempt and display an error message back to the user explaining why his or her attempt failed.

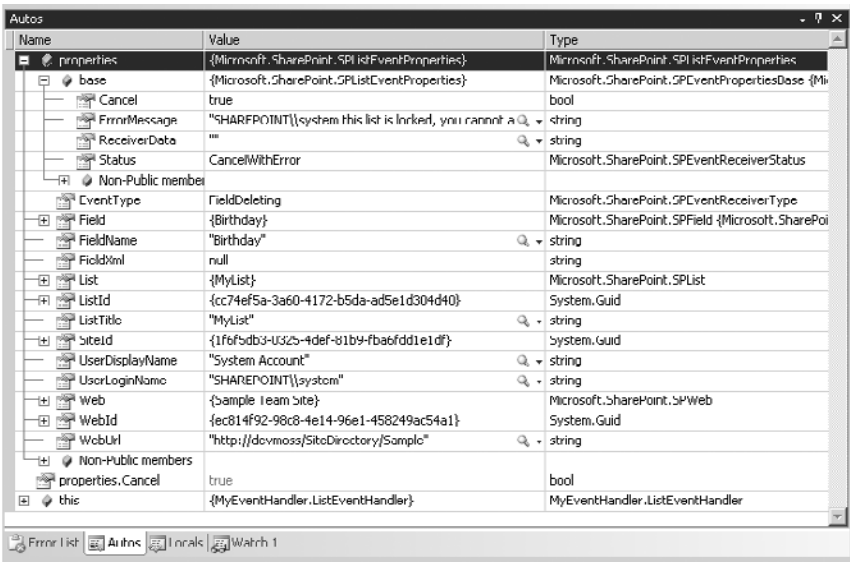


Figure 17.13 *SPListEvent Properties sample values.*

17.1.5 Additional Event Receivers

WSS 3.0 provides additional classes for use when building your solutions, and although technically they don't inherit from the same SPEventReceiverBase class, they do allow you to trap and handle specific actions. For example, in Chapter 16, Features we showed how you can leverage the concept of Feature receivers to trap when a Feature is installed, activated, deactivated, and uninstalled. Our Feature receiver inherits the `SPFeatureReceiver` class, an abstract class that provides four methods specific to the Feature lifecycle: `FeatureInstalled`, `FeatureActivated`, `FeatureDeactivating`, and `FeatureUninstalling`. In essence, Feature receivers are event receivers; while they do not actually inherit the `SPEventReceiverBase` class, they are still examples of event receivers because they allow you to trap a specific event and perform custom actions accordingly.

The `SPEmailEventReceiver` class provides an `EmailReceived` method that allows you to trap when a list receives an e-mail and then perform an appropriate action. The `EmailReceived` method returns a set of parameters that includes an `SPList` object representing the list that received the e-mail, and an `SPEmailMessage` object representing the e-mail message that triggered the event. The `SPEmailMessage` gives you access to all the data associated with the e-mail message, such as the sender, the body of the message, and the collection of attachments to the message. You can then use this information when determining what to do with the message. For example, you can implement logic that causes all attachments from a specific person to be

filed in a designated folder. The WSS 3.0 SDK provides sample code of how to create an event handler that inherits the `SPEmailEventReceiver` class.

17.1.6 Support for WSS 2.0 Events

So far we have talked about the event receivers that are available in WSS 3.0, but what happens to the event handlers that you wrote for the previous version of SharePoint? The good news is that Microsoft provides backward compatibility for WSS 2.0 events. If you examine the object model, you will find that there is still support for `IListEventSink` interface. The `IListEventSink` interface provides one method, the `OnEvent` method, which is used to respond to changes that occur to a document stored in a document library. The `OnEvent` method receives the events as they occur in the form of an `SPListEvent` object. As a developer, all you need to do is insert code into the `OnEvent` method that listens for the specific event types in which you are interested and process them accordingly. Table 17.7 lists the event types that are supported with this method.

Table 17.7 *Event Types Supported by SPListEvent*

Event Type	Triggered when…
CheckIn	A document is checked into the document library.
CheckOut	A document is checked out of the document library.
Copy	A document in the document library is copied
Delete	A document in the document library is deleted
Insert	A new document is saved to the document library
Move	A document in the document library is moved or renamed.
Delete	A document in the document library is deleted.
UncheckOut	The CheckOut of a document in the document library is undone.
Update	A document in the document library is edited.

Note that if you code your event handlers using the `IListEvenSink` class, then your handler will only support asynchronous events.

17.2 Working with Events

So now that we understand the type of event handlers that can be written, let's look at the actual process involved in creating and implementing an event handler. The process in and of itself is quite straightforward and involves three main stages: creation, deployment, and registration. The creation stage typically involves a developer creating an event handler assembly that overrides the relevant event receiver methods. Once the assembly file has been created, the developer can then package the assembly into a solution file for the administrator to deploy on the appropriate Web servers. Once the event handler is installed all that remains is for the event receivers to be registered with the appropriate SharePoint containers, such as a specific Web site or a set of document libraries. To make the administrator's life a lot easier, and make the deployment process more streamlined, the developer can include the registration process as part of the solution package.

The following section will take you through the process of implementing a simple event handler that prevents items from being deleted from a specific document library within a specific SharePoint Web site, regardless of the user's permission level.

17.2.1 Creating an Event Handler

The first step in the process is to create the event handler. The event handler needs to be able to trap when a user attempts to delete a Web site within our site collection; therefore, our event handler will need to inherit the `SPItemEventReceiver` class and override the `ItemDeleting` method. To follow along with the example, create a new C# Class Library project in Visual Studio 2005, add a reference to `Microsoft.SharePoint`, and add the code listed in Figure 17.14. Once the code is entered, you can build the project assembly.

Figure 17.14
ItemDeleting code.

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

namespace MyEventHandler
{
    public class MyItemEventHandler : SPItemEventReceiver
    {

        public override void ItemDeleting(SPItemEventProperties properties)
        {
            string errmsg = properties.UserLoginName;
            properties.ErrorMessage = errmsg + ", sorry you do not have delete permissions.";
            properties.Cancel = true;
        }
    }
}
```

SharePoint event handler assemblies must be strongly named and deployed to the global assembly cache (GAC) before they can be used. As a quick refresh, you can use the Strong Name Tool (sn.exe) to strongly name your assembly or, alternatively, use the Signing tab available in Visual Studio 2005. If you are using the Signing tab, then your assembly is strongly named as part of the build process.

At this stage the assembly file has been created; next we need to deploy the file and register the event handler.

17.2.2 Deploying an Event Handler

Once again, event handler assemblies must be strongly named and deployed to the GAC before they can be used. A very simple way to deploy an assembly file to the GAC is to drag the strongly named .dll build file from its project directory and drop it into the GAC, which is usually located in the `\WINDOWS\assembly` directory. This particular method of deployment is fine for demonstration purposes, but when it comes to production, the recommended method of deploying event handlers is to leverage the Solution Deployment framework. Doing so allows you to inherit all the benefits of the framework, such as providing a central place for deployment to all appropriate Web applications and Web front-end servers, providing a central place for upgrades and for retracting the solution, and being able to package the deployment and registration of the event handler. Chapter 11, Introduction to SharePoint Development, explains the steps involved in packaging and deploying a Web Part solution; since an event handler is just a .dll, the same technique can the applied to deploy the event handler to the GAC of all the appropriate Web front-end servers.

As previously mentioned, the registration of the event handler can be incorporated into the deployment process by including a registration mechanism in the solution deployment files. For example, the solution deployment package could include a Feature that, upon activation, would register the event handlers to all the relevant SharePoint containers; upon deactivation, the same Feature would unregister the event handlers. More information on how to create Features that handle activation events can be found in Chapter 16, Features.

17.2.3 Registering an Event Handler

The registration of an event handler refers to the process that takes a specific method that a custom event handler overrides and binds it to a SharePoint object such as a list, document library, or even a content type. For example, in our scenario we can bind the `ItemDeleting` method from our assembly to a specific SharePoint list. If you examine the object model you will see that

objects such as `SPWeb`, `SPList`, `SPFile`, `SPWorkflow`, and `SPContentType` each contain an `EventReceivers` property. The `EventReceivers` property enumerates the collection of event receiver definitions that are currently registered with that object. An event receiver definition is the registration of a specific event receiver method that is being overridden by a specific event handler. When registering your own custom event handlers, you can add the registrations to the default event receiver definition collection associated with that object, or you can create a new event receiver definition collection object and add all your registrations to it. The `EventReceivers` property is a great way to find out what events are currently registered on your site. Let's take the example of an out-of-the-box SharePoint team site that has the Publishing Feature activated. Figure 17.15 shows the code for a simple Web Part that enumerates all the event receivers registered at the Web site level, and all the event receivers registered for document libraries within the Web site.

Figure 17.15 *Web Part code enumerating registered events (C#).*

```
namespace RegisteredEvents
{
    [Guid("1fd77bf8-1fc9-4b2e-8aaf-4679e987e9f2")]
    public class RegisteredEvents : System.Web.UI.WebControls.WebParts.WebPart
    {
        protected override void Render(HtmlTextWriter writer)
        {
            SPWeb curWeb = SPContext.Current.Web;

            writer.Write("<B>Registered Web Event Receivers</B><br>");
            SPEventReceiverDefinitionCollection webdefcoll = curWeb.EventReceivers;
            foreach (SPEventReceiverDefinition wdef in webdefcoll)
            {
                writer.Write(wdef.Name + " - " + wdef.Type + "<BR>");
            }

            writer.Write("<B>Registered Document Library Event Receivers</B><br>");
            SPListCollection docLibs = curWeb.GetListsOfType(SPBaseType.DocumentLibrary);
            foreach (SPList docLib in docLibs)
            {
                writer.Write("<U>List Name:" + docLib.Title + "</U><BR>");
                SPEventReceiverDefinitionCollection libdefcoll = docLib.EventReceivers;
                foreach (SPEventReceiverDefinition ldef in libdefcoll)
                {
                    writer.Write(ldef.Name + " - " + ldef.Type + "<BR>");
                }
            }
        }
    }
}
```

The results of this Web Part are shown in Figure 17.16. There are five event handler methods registered at the Web site level, three of which are registered under the same event receiver definition, named `AreaEventReceiver`. If we examine further down the list presented in the Web Part, we see that there is a document library named Pages that has five event handler methods registered under the event receiver definition named `PagesListEventReceiverName`. An event receiver method can only be registered once to an object. For example, the event handler methods registered to the Pages document library include `ItemDeleting`, `ItemAdded`, `ItemUpdating`, `ItemDeleted`, and `ItemCheckedIn`. While the `PagesListEventReceiverName` definition's

`ItemDeleting` event is registered to the Pages document library, you cannot register another `ItemDeleting` event to that same document library, regardless of which event receiver definition it belongs. Also, notice that the document libraries titled Classified Documents and Unclassified Documents do not have event handlers registered to them as yet. We will revisit these specific document libraries as we proceed through the registration process.

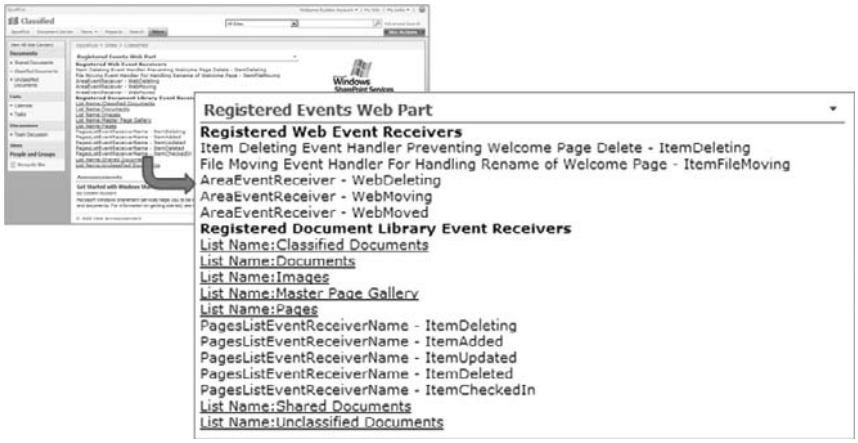


Figure 17.16 *Registered events Web part.*

An event handler can be registered in one of three ways: through using a Feature associated with a specific list type, through a content type Feature definition, or through code. Each method requires a set of standard properties that describe the registration. These properties include the assembly information, the class information, the receiver type, the receiver name, and the sequence number. The assembly information contains the strong name of the assembly used to handle the events. The class information contains the name of the class that implements the receiver. The receiver type defines the type of event that is being registered, for example `ItemDeleting`. The receiver name specifies the name of the event receiver to be registered. If multiple event handlers are being registered, the sequence number specifies the order. So while there are a number of methods of registering event handlers, each method involves providing this information to SharePoint. Let's examine how each method registers an event handler and why you would use that particular method.

17.2.4 Registering Event Handlers using Features

If you would like to bind your event handler to a specific list type, such as a document library or a discussion board, you can do this very easily through Features. Simply create a folder in the `\TEMPLATE\FEATURES` directory with the name of your event handler, and then create a Feature.xml file

with appropriate Feature `ID`, scoping, and reference to an element manifest file. Next, create a matching element manifest file that contains the registration data for the event handler. Features that register receivers may only be scoped at the Web site level, meaning that they are installed and activated in the site features gallery. Figure 17.17 shows the Feature.xml file associated with our sample event handler. The Feature.xml file references the element manifest file elements.xml. The content of the elements.xml file is shown in Figure 17.18.

Figure 17.17
Feature.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<Feature  Id="9BC88D61-8292-4cb6-9E7D-B1D8A67D3EC2"
          Title="My Event Handler Registration"
          Description="My custom event handler registration"
          Version="1.0.0.0"
          Scope="Web"
          Hidden="FALSE"
          xmlns="http://schemas.microsoft.com/sharepoint/">
    <ElementManifests>
        <ElementManifest Location="elements.xml" />
    </ElementManifests>
</Feature>
```

The element manifest uses the `Receivers` element to register the event handler. In our particular example, the `Receivers` element contains a single `Receiver` child element, which in turn contains child elements that define the registration, including elements that map to the aforementioned common properties such as the assembly and class information. You can register multiple event handlers by simply including an additional `Receiver` child element for each registration.

Figure 17.18
elements.xml.

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Receivers
      ListTemplateId="101">
    <Receiver>
      <Name>My Custom Receiver</Name>
      <Type>ItemDeleting</Type>
      <SequenceNumber>10001</SequenceNumber>
      <Assembly>MyEventHandler, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=9dd303c4e2c03c80</Assembly>
      <Class>MyEventHandler.MyItemEventHandler</Class>
      <Data></Data>
    </Receiver></Receivers>
</Elements>
```

The `ListTemplateId` attribute of the `Receivers` element defines the list type with which you wish to register your event handler. Using Features,

you can register your event handlers to any of the default list templates, or to your own custom list template. Table 17.8 lists a subset of the default list templates that come standard with SharePoint, along with their associated template ID. If you examine the code in our elements.xml example, you will notice that we have set the `ListTemplateId` attribute to 101, which means that we are registering our custom event handler to *all* the document libraries in the Web site in which the Feature is activated. The `Name` element is a child element of the `Receiver` element and specifies the name of the event receiver definition to which the registration is to be added. If this element is left blank, the event handler is added to the default event receiver definition collection. In our example, the event handler is registered to a custom event receiver definition named My Custom Receiver, and the event type that is registered is the `ItemDeleting` event type.

Table 17.8 *List Template IDs*

List Template	Template ID
Generic List	100
Document Library	101
Survey	102
Links List	103
Announcements List	104
Contacts List	105
Events List	106
Tasks List	107
Discussion Board	108
Picture Library	109
Site Template Gallery	111
Web Part Gallery	113
List Template Gallery	114
XML Form Library	115
Web Page Library	119

Now all we need to do is install and activate the Feature. Figure 17.19 shows the stsadm.exe commands that will install and activate a Feature in SharePoint. You do not necessarily have to activate the Feature through the

command line; you can simply use the Activate button on the Site Features page of the desired Web site. Note that in order to activate Features through the browser UI, the Feature `Hidden` attribute must be set to `False`.

Figure 17.19
Installing and activating the feature.

```
stsadm.exe –o installfeature –name featurefoldername

stsadm.exe –o activatefeature –name featurefoldername –url http://devmoss/sites/samplesite
```

Once the Feature is activated, the custom event handler is registered to all the document libraries within that Web site. If the Feature is subsequently deactivated, the event handler is automatically unregistered from the appropriate lists. Figure 17.20 shows the registered events Web Part for our sample Web site. Notice that the Classified Documents and Unclassified Documents document libraries now have our custom event handler registered; in fact, the event handler is registered to *all* the document libraries in the site. Furthermore, any new document libraries that are created in this site, while the event handler registration Feature is activated, will automatically inherit the event handler registrations.

Figure 17.20
Custom event handler registered to document libraries.

Registering event handlers through Features allows you to register event handlers to a specific *type* of list, such as a document library, or discussion board; however, it does not provide an easy mechanism for registering an

event handler to a *specific* list, such as the Classified Documents document library. For that you would need to use the object model.

17.2.5 Registering Event Handlers using Content Types

If you want to bind your event handler to a specific type of content, you can do so easily by modifying the XML definition of the associated content type. For example, suppose that you only wanted to prevent items from being deleted from a specific type of folder within a document library, but not from the document library itself. To achieve this you would create a Feature that contained the folder content type definition. The folder content type definition would in turn contain an event handler registration for the `ItemDeleting` event. Content types are implemented as Features, and the element manifest file that contains the content type definition includes a special `XMLDocuments` element in which you can register event handlers. The `XMLDocuments` element contains a set of `XMLDocument` elements that allow you to include additional information in the form of XML nodes, such as registering the event handlers you would like to associate with a given content type. Figure 17.21 shows the Feature.xml file for a custom folder content type. The file references an element manifest file named folderctype.xml that contains the content type definition. Figure 17.22 shows the corresponding element manifest for the custom folder content type that contains an event handler registration for the `ItemDeleting` event. The `Receiver` element contains the same information that was required when registering an event handler using Features, which makes sense given that the content type definition is actually a Feature.

Figure 17.21 *Custom content type Feature.xml.*

```
<?xml version="1.0" encoding="utf-8"?>
<Feature  Id="CB74E862-854D-4d89-8AA2-62BAB17C372B"
          Title="Custom Folder"
          Description="$Resources:core,ctypesFeatureDesc"
          Version="1.0.0.0"
          Scope="Site"
          Hidden="FALSE"
          DefaultResourceFile="core"
          xmlns="http://schemas.microsoft.com/sharepoint/">
    <ElementManifests>
        <ElementManifest Location="folderctype.xml" />
    </ElementManifests>
</Feature>
```

Registering event handlers to content types provides more control over when the event handler actually fires by allowing you to be more specific with your registration. As previously mentioned, registering an event handler through Features alone only allows you to target an event handler to a list *type*, which means the event handler is registered for *all* lists of that type. Additionally, by using the Feature method, you can only scope the event

Figure 17.22
Custom content type element manifest.

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
 <ContentType ID="0x012020"
        Name="Classified Folder"
        Group="$Resources:Folder_Content_Types"
        Description="Create restricted folder to contain classified content"
        Sealed="FALSE"
        Version="0">
        <FieldRefs>
            <FieldRef ID="{fa564e0f-0c70-4ab9-b863-0177e6ddd247}"
                      Name="Title" Required="FALSE"
                      Hidden="TRUE"/> <!-- Title -->
            <FieldRef ID="{8553196d-ec8d-4564-9861-3dbe931050c8}"
                      Name="FileLeafRef" Hidden="FALSE"
                      Required="TRUE"/> <!-- FileLeafRef -->
        </FieldRefs>
        <XmlDocuments>
            <XmlDocument NamespaceURI="http://schemas.microsoft.com/sharepoint/events">
             <spe:Receivers xmlns:spe="http://schemas.microsoft.com/sharepoint/events">
                <Receiver>
                    <Name>My Custom Receiver</Name>
                    <Type>ItemDeleting</Type>
                    <SequenceNumber>10000</SequenceNumber>
                    <Assembly>MyEventHandler, Version=1.0.0.0, Culture=neutral,
                              PublicKeyToken=9dd3d3c4e2cd3c8d</Assembly>
                    <Class>MyEventHandler.MyItemEventHandler</Class>
                    <Data></Data>
                </Receiver>
             </spe:Receivers>
            </XmlDocument>
        </XmlDocuments>
 </ContentType>
</Elements>
```

handler registration to the Web site level. This means that for the event handler registration to take effect, it needs to be activated at each individual Web site. In contrast, content type Features can be scoped at the site collection level, which means that once the feature is activated at the site collection level, the content type becomes available to all Web sites within that site collection. Content type Features are installed and activated in the same way as all regular Features; Figure 17.19 shows the necessary stsadm.exe commands. Once again, you do not have to activate the Feature through the command line; instead you can simply use the Activate button on the Site Collection Features page of the desired site collection.

Figure 17.23 shows the registered events Web Part after the new content type has been bound to the Classified Documents document library. At first glance everything looks fine; we can see that our custom event handler has registered successfully for the `ItemDeleting` event. Since this event handler is registered to the Classified Folder content type, it would not be remiss to expect the `ItemDeleting` event handler code to fire *only* when an item of that content type was deleted. However, this is not the case; there are several anomalies with respect to the behavior of events and where they are registered. In this instance, when the `ItemDeleting` event handler is registered to a content type, and that content type is bound to a document library then the `ItemDeleting` event handler fires when *any* item, regardless of the content type, is deleted from the document library. If you register an

`ItemAdding` event handler to the same content type that is bound to a document library, you would find that the `ItemAdding` event handler only fires when an item of that specific content type is added to the document library. This anomaly, and others, is documented in the WSS 3.0 SDK under the title Event Troubleshooting. If, when developing your own custom event handlers, you run into a scenario where your code is not producing the expected results, then you should reference the troubleshooting section to see if it is a known scenario.

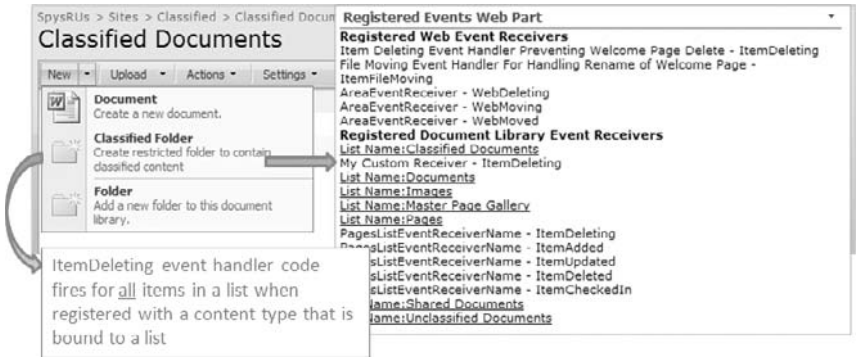


Figure 17.23
Content type with registration bound to Classified Documents library.

To remove an event handler registration from a content type definition, simply remove the appropriate `Receiver` element definition from the content type element manifest, reset IIS, and reinstall the content type Feature using the `–force` switch. Keep in mind, that if the content type is bound to a document library prior to the removal of the registration, then the content type will continue to use the old content type definition and therefore the event handlers will still run. In this instance you have a couple of options: you can remove the old content type version from the document library and then add it back again, thereby inheriting the new definition, or you can remove the registration through custom code. You cannot remove a content type association from a document library until all objects of that specific content type are deleted from the document library. So, in our particular `ItemDeleting` scenario, removing the registration through code is our easiest option.

It is important to note that it is not recommended to modify the out-of-the-box content type definitions supplied by Microsoft. Doing so could land you in unsupported territory. Instead, you should create a custom content type based on the default content type.

17.2.6 Registering Event Handlers using the Object Model

At the beginning of this section, we mentioned that we would take you through the process of implementing a simple event handler that prevents items from being deleted from a specific document library within a specific SharePoint Web site, regardless of the user's permission level. So far we have shown you how to create the custom event handler and how to register it to a specific list type and a content type. We have not, as yet, discussed how to register event handlers to a specific list, such as the document library titled Classified Documents as shown in Figure 17.16. With respect to registering event handlers, the object model provides you with the most flexibility. For example, using the object model, you can register event handlers to a specific list or library and, as shown in our enumeration example, you can leverage the object model to inform you of the current event handler registrations associated with a site.

Figure 17.24
Registering event handler with code (C#).

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using Microsoft.SharePoint;
5
6  namespace myEventRegistration
7  {
8      class Program
9      {
10         static void Register(string asmInfo, string sitecollurl, string webrelurl, string listname)
11         {
12             //Register with specific list
13             using (SPSite siteCollection = new SPSite(sitecollurl))
14             {
15                 using (SPWeb webSite = siteCollection.OpenWeb(webrelurl))
16                 {
17                     SPList myList = webSite.Lists[listname];
18                     SPEventReceiverDefinitionCollection lstreceivers = myList.EventReceivers;
19                     SPEventReceiverDefinition rdef = lstreceivers.Add();
20                     rdef.Name = "My Custom Receiver";
21                     rdef.Assembly = asmInfo;
22                     rdef.Class = "MyEventHandler.MyItemEventHandler";
23                     rdef.Type = SPEventReceiverType.ItemDeleting;
24                     rdef.SequenceNumber = 10001;
25                     rdef.Update();
26
27                     Console.WriteLine("Operation successful");
28                 }
29             }
30             Console.ReadLine();
31         }
32         static void Main(string[] args)
33         {
34
35             string AssemblyInfo = "MyEventHandler, Version=1.0.0.0, Culture=neutral, PublicKeyToken=9dd303c4e2c03c80";
36             Register(AssemblyInfo, "http://devmoss/", "/SiteDirectory/Classsified", "Classified Documents");
37         }
38     }
39 }
```

Figure 17.24 shows sample C# code from a simple console application that registers our event handler to the document library named Classified Documents. In this particular example, we are creating a new event receiver definition (My Custom Receiver), adding it to the collection of receiver definitions for the list, and using it to register our event handler (lines 18 through 25). Notice in Figure 17.25 that the custom event handler is only registered to the Classified Documents document library, and the name of the event receiver definition is populated with the data in the code listing that is assigned to the event receiver definition `Name` attribute. Figure 17.26

shows another method of registering the same event handler. This time, instead of creating a new event receiver definition, we are adding the registration to the default event receiver definition (line 18). Figure 17.27 shows the outcome of this type of registration. Notice that the name of the event receiver definition is blank, which indicates that the registration has been added to the default event receiver definition.

Figure 17.25 *Document library event handler registration.*

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using Microsoft.SharePoint;
5
6  namespace myEventRegistration
7  {
8      class Program
9      {
10         static void Register(string asmInfo, string sitecollurl, string webrelurl, string listname)
11         {
12             string strClass = "MyEventHandler.MyItemEventHandler";
13             using (SPSite siteCollection = new SPSite(sitecollurl))
14             {
15                 using (SPWeb webSite = siteCollection.OpenWeb(webrelurl))
16                 {
17                     SPList myList = webSite.Lists[listname];
18                     myList.EventReceivers.Add(SPEventReceiverType.ItemDeleting, asmInfo, strClass);
19                     Console.WriteLine("Operation successful");
20                 }
21             }
22             Console.ReadLine();
23         }
24
25         static void Main(string[] args)
26         {
27
28             string AssemblyInfo = "MyEventHandler, Version=1.0.0.0, Culture=neutral, PublicKeyToken=9dd303c4e2c03c00";
29             Register(AssemblyInfo, "http://devmoss/", "/SiteDirectory/Classified", "Classified Documents");
30         }
31     }
32 }
```

Figure 17.26 *Register to default event receiver definition (C#).*

17.2.7 Removing Event Handler Registrations using the Object Model

So far we have discussed how to remove an event handler registration simply by deactivating a Feature, or removing the event handler registration from a content type definition. Another option for removing event handler regis-

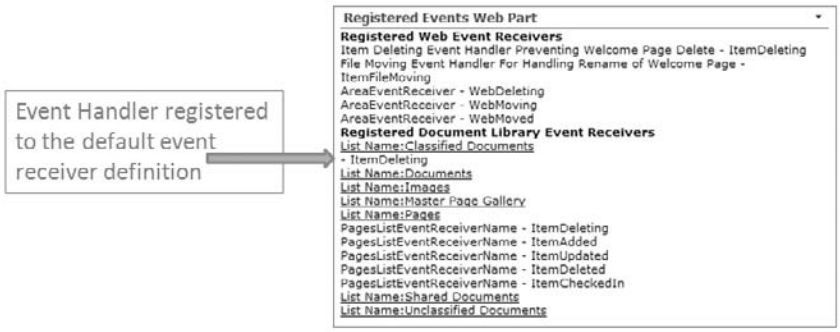


Figure 17.27
Web Part showing default event receiver.

trations is through the object model. When it comes to removing an event handler registration, the object model once again offers the most flexibility. Using the object model, you can remove event handler registrations from many different objects, including sites, lists, document libraries, and content types. Figure 17.28 shows some sample code that iterates through the event receiver definition collection for a given list and deletes any event handlers that are registered with a specific event receiver definition. If the `defName` parameter is an empty string, then the event handlers registered to the default event receiver definition for the list are removed. This example shows how grouping your event handler registrations under the same event receiver definition collection can make manageability of the registrations much simpler.

```
namespace myEventRegistration
{
    class Program
    {
        static void RemoveListReg(string sitecollurl, string webrelurl, string listname, string defName)
        {
            using (SPSite siteCollection = new SPSite(sitecollurl))
            {
                using (SPWeb webSite = siteCollection.OpenWeb(webrelurl))
                {
                    SPList myList = webSite.Lists[listname];
                    SPEventReceiverDefinitionCollection lstreceivers = myList.EventReceivers;
                    List<SPEventReceiverDefinition> deleteItems = new List<SPEventReceiverDefinition>();
                    foreach (SPEventReceiverDefinition redef in lstreceivers)
                    {
                        if (redef.Name == defName)
                        {
                            deleteItems.Add(redef);
                        }
                    }
                    for (int i = 0; i < deleteItems.Count; i++)
                    {
                        deleteItems[i].Delete();
                    }
                    Console.WriteLine("Operation successful");
                }
            }
            Console.ReadLine();
        }
        static void Main(string[] args)
        {
         RemoveListReg("http://devmoss/", "/SiteDirectory/Classsified", "Classified Documents",
                       "My Custom Receiver");
        }
    }
```

Figure 17.28
Code to remove event handler registrations (C#).

There are plenty of tools accessible via the Internet and downloadable for free, that allow you to easily register and unregister event handlers through a GUI type application. Alternatively, you can create your own custom application now that you know how to enumerate, register, and unregister event handlers.

17.3 Working with Document Libraries—Event Exercise

The WSS 3.0 event architecture has improved greatly since its predecessor and provides developers with a great platform for extending the base product functionality. In this final section, we will walk through an exercise that demonstrates the creation of an event handler that overrides the `SPItemEventReceiver` class.

Those of you familiar with managing the permissions and roles for public folders in Microsoft Exchange should be aware that you can assign permissions to determine which users are allowed the ability to create folders. SharePoint Server 2007 and WSS 3.0 gives you the option to turn off the create folder interface, which basically determines whether the "New Folder" option appears on the New menu. This menu option is available on the Document Library Advanced Settings page (Figure 17.29). It does not remove the functionality itself, nor does it apply any permissions-based logic for accessing the Create Folder page. If you know how to navigate to the Create Folder page, by perhaps remembering the URL, then you can create a folder as you would if the menu option existed. Furthermore, out of the box you cannot define which users are allowed the ability to create or delete folders based on their group membership or permissions; however, this functionality can be provided through events.

In this exercise we will create an event handler that ensures only members of the "Classified Users" SharePoint group have the ability to create or delete folders in all the document libraries within a site. Users who are *not* members of the "Classified" SharePoint group may only delete the documents that they created. In addition, when a document is added to a document library, a link to that document is inserted into the "Classified Links" list at the root of the site collection. Upon deletion of a document, we also need to remove any links to that document that have been added to the "Classified Links" list. Please note that the code used throughout this chapter is for demonstration purposes only, as it has not been thoroughly tested and does not contain the appropriate exception handling required of a production application. Nevertheless, it illustrates the power of and some potential uses for leveraging the event architecture when extending and customizing your SharePoint implementation.

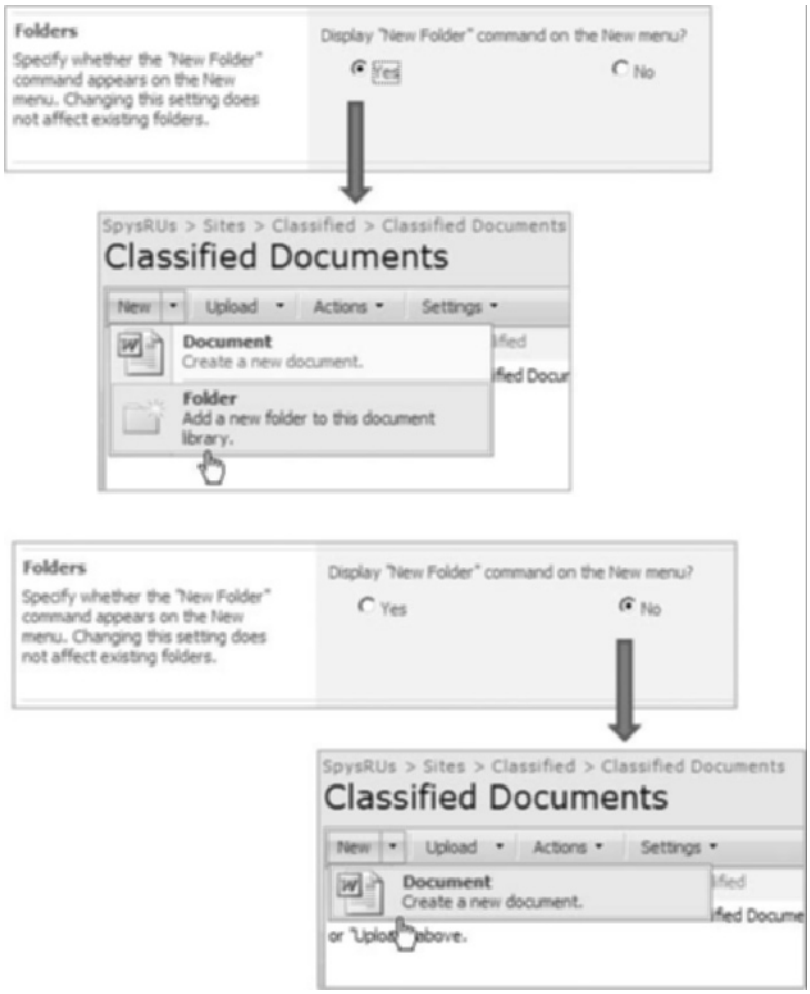


Figure 17.29
Turn folders on and off.

At a very high level, the steps involved include creating an event handler that inherits the `SPItemEventReceiver` class and overrides the `ItemDeleting`, `ItemAdding`, and `ItemAdded` events. The `ItemDeleting` method checks the content type, then checks for group membership to determine whether the user is allowed to proceed. The `ItemAdding` method also checks the content type of the item being added, then checks for group membership to determine whether the user is allowed to create a folder in the document library. Finally, the `ItemAdded` event checks the content type of the item that was just added to the document library, and if it is a document, a link to that document is added to the "Classified Links" list. Once the event handler is created, we will deploy the event handler and register it using a Feature. Note that we are using C# for this exercise.

In preparation for this exercise, create a Links list named "Classified Links" at the root of the site collection, and add a Web Part for this list to the site collection home page. In addition, create a new SharePoint group named "Classified Users" and assign some user accounts to the group. Make sure that you have a user account that is *not* a member of this group, with which you can test the effects of the event handler. All user accounts should have contributor rights to the site and document libraries that you will be testing against.

17.3.1 Create and Deploy the Event Handler

1. Open Visual Studio 2005 and create a new C# class library project.
2. Rename your Class.cs file and you class to a more suitable name, for example "ItemEventHandler."
3. From the Solution Explorer, navigate to the References folder and add a reference to the Microsoft.SharePoint.dll, which is the Windows SharePoint Services component.
4. Add a `using` directive for the `Microsoft.SharePoint` namespace.
5. Set your class to inherit from `SPItemEventReceiver` class.
6. Add the C# code listings shown in Figures 17.30 through 17.35 to your class.
7. Strongly name your assembly. From the Project menu, click Properties. From the Properties dialog, click the Signing tab and sign the assembly by creating a new strong key name file.
8. Save your project.
9. From the Build menu, build your event handler assembly.
10. Using Windows Explorer, navigate to the folder in which your assembly .dll file was generated.
11. Open another Windows Explorer window and navigate to the GAC (`\WINDOWS\assembly`).
12. Drag and drop your assembly file from the project directory into the GAC.
13. Navigate to the assembly file in the GAC, right-click on the file, and display its properties.
14. From the property window, copy the Public Key Token and keep it accessible for registering the event handler (see the next section of this chapter). Alternatively you can download the Lutz

Figure 17.30
ItemAdding code (C#).

```
        public override void ItemAdding(SPItemEventProperties properties)
        {
            //We can check the content type by inspecting the AfterProperties of the Item

            string curContentType = "Folder";
            SPItemEventDataCollection aprops = properties.AfterProperties;
            if (aprops["ContentType"] != null)
            {
                curContentType = aprops["ContentType"].ToString();
            }

            using (SPSite curSite = new SPSite(properties.WebUrl))
            {
                using (SPWeb curWeb = curSite.OpenWeb())
                {
                    if (curContentType == "Folder")
                    {
                        bool ismember = curWeb.IsCurrentUserMemberOfGroup(curWeb.Groups["Classified Users"].ID);
                        string errmsg = "You are currently logged on as: " + properties.UserLoginName;
                        if (ismember == false)
                        {
                            properties.Cancel = true;
                            errmsg = errmsg + "You do not have the appropriate permisssions to add: " + curContentType +
+ " content from this document library";
                            properties.ErrorMessage = errmsg;
                        }
                    }
                }
            }
        }
```

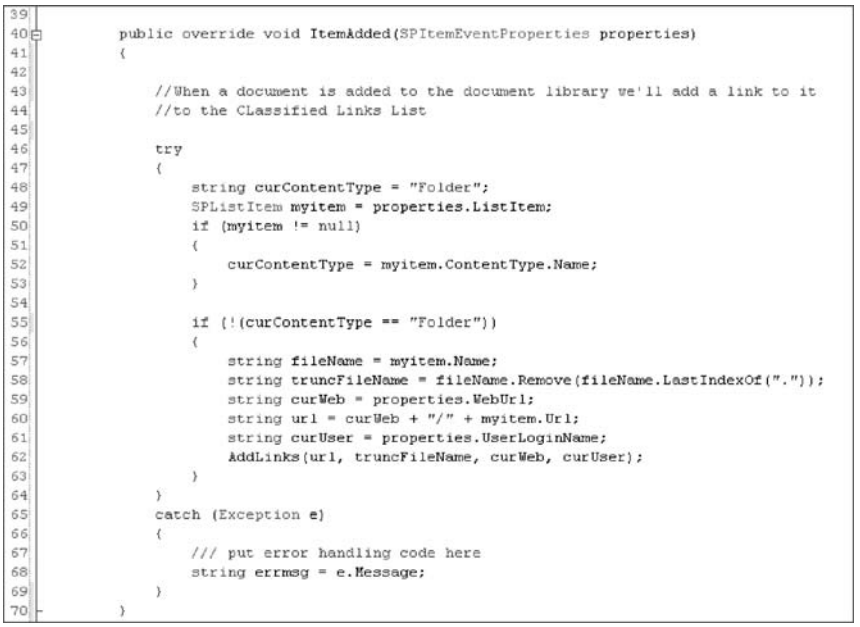
Roeder's .NET Reflector tool from the Internet and grab the assembly information from that program.

Now that we have created and deployed our event handler, all that remains is for us to register the event handler with *all* the document libraries in our site. We can achieve this by registering our event handler using Features targeted to the document library template ID.

17.3.2 Register the Event Handler

1. Navigate to the `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\FEATURES` directory.
2. Create a new Feature folder, for example, myEventHandler.
3. In the new Feature folder, create a file named Feature.xml with the contents shown in Figure 17.36. Replace the Id attribute in the feature element with your own GUID. You can create a GUID using the GUID generator tool in Visual Studio 2005 (guidgen.exe).
4. In the new Feature folder, create an element manifest file named elements.xml, with the contents shown in Figure 17.37. For each receiver definition, replace the assembly information with your

Figure 17.31 *ItemAdded code (C#).*



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

            //When a document is added to the document library we'll add a link to it
            //to the CLassified Links List

            try
            {
                string curContentType = "Folder";
                SPListItem myitem = properties.ListItem;
                if (myitem != null)
                {
                    curContentType = myitem.ContentType.Name;
                }

                if (!(curContentType == "Folder"))
                {
                    string fileName = myitem.Name;
                    string truncFileName = fileName.Remove(fileName.LastIndexOf("."));
                    string curWeb = properties.WebUrl;
                    string url = curWeb + "/" + myitem.Url;
                    string curUser = properties.UserLoginName;
                    AddLinks(url, truncFileName, curWeb, curUser);
                }
            }
            catch (Exception e)
            {
                /// put error handling code here
                string errmsg = e.Message;
            }
        }
```

Figure 17.32 *ItemDeleting code: part A (C#)*

```
        public override void ItemDeleting(SPItemEventProperties properties)
        {
            //If a folder is being deleted we need to check to ensure the user is authorized
            //If a document is being deleted we need to restrict the users to allow them only
            //to delete the documents that they created, unless they are a privileged user.

            using (SPSite curSite = new SPSite(properties.WebUrl))
            {
                using (SPWeb curWeb = curSite.OpenWeb())
                {
                    SPListItem curItem = properties.ListItem;
                    string curContentType = curItem.ContentType.Name;
                    string curOwner = curItem["Created By"].ToString();
                    curOwner = curOwner.Remove(curOwner.LastIndexOf(";"));
                    int curOwnerId = int.Parse(curOwner);
                    int curUserId = properties.CurrentUserId;
                    bool ismember = curWeb.IsCurrentUserMemberOfGroup(curWeb.Groups["Classified Users"]
.ID);

                    if (curContentType == "Folder")
                    {
                        string errmsg = properties.UserLoginName;
                        if (ismember == false)
                        {
                            properties.Cancel = true;
                            errmsg = errmsg + " you do not have the appropriate permisssions to delete
classified content from this document library";
                            properties.ErrorMessage = errmsg;
                        }
                    }
```

assembly information, such as the name of the assembly, the name of the class and the Public Key Token.

5. Install and activate the Feature using the stsadm.exe commands shown in Figure 17.38. Replace the Feature folder name,

Figure 17.33
ItemDeleting code: part B (C#)

```
                    else if ((curOwnerId != curUserId) & (ismember == false))
                    {
                        properties.Cancel = true;
                        string errmsg = "You are not the owner of the item. You are only allowed to
delete items you created.";
                        properties.ErrorMessage = errmsg;
                    }
                    else
                    {
                        //go ahead and delete the item,
                        //but don't forget to remove the link from the Classified Links folder
                        string fileName = curItem.Name;
                        string truncFileName = fileName.Remove(fileName.LastIndexOf("."));
                        string curWebUrl = properties.WebUrl;
                        string relWebUrl = properties.RelativeWebUrl;
                        string url = string.Empty;
                        if (relWebUrl == "/")
                        {
                            url = relWebUrl + curItem.Url;
                        }
                        else
                        {
                            url = properties.RelativeWebUrl + "/" + curItem.Url;
                        }
                        RemoveLinks(url, curWebUrl);
                    }
                }
            }
        }
```

Figure 17.34
AddLinks code (C#).

```

        private void AddLinks(string linkUrl, string linkTitle, string curWebUrl, string curUser)
        {
            using (SPSite curSite = new SPSite(curWebUrl))
            {
                using (SPWeb linkWeb = curSite.RootWeb)
                {
                    SPList linkList = linkWeb.Lists["Classified Links"];
                    SPListItem newlnk = linkList.Items.Add();
                    newlnk["URL"] = linkUrl + ", " + linkTitle;
                    newlnk["Comments"] = "Classified document added by: " + curUser;
                    newlnk.Update();
                }
            }
        }

```

Figure 17.35
RemoveLinks code (C#).

```
        private void RemoveLinks(string linkUrl, string curWebUrl)
        {
            using (SPSite curSite = new SPSite(curWebUrl))
            {
                using (SPWeb linkWeb = curSite.RootWeb)
                {
                    SPList linkList = linkWeb.Lists["Classified Links"];
                  //We need to search the Links List for all entries associated with the deleted document
                  // and remove the link
                  // the query needs to use the relative url, not the absolute url since the absolute url
                  // causes missing results
                        string srchUrl = linkUrl;
                        SPQuery qry = new SPQuery();
                        qry.Query = "<Where><Eq><FieldRef Name='URL' /><Value Type='URL'>" + srchUrl + "</ v
Value></Eq></Where>";
                        SPListItemCollection results = linkList.GetItems(qry);
                        if (results.Count > 0)
                        {
                            for (int i = 0; i < results.Count; i++)
                            {
                                results.Delete(i);
                            }
                        }
                    }
                }
            }
        }
```

myEventHandler, with the name of your Feature, and replace the URL with the URL of the site that you wish to activate the Feature.

Figure 17.36
Feature.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<Feature  Id="7CB8BB95-DF6B-43cc-8952-D048B48E1D22"
          Title="My Event Handler Registration"
          Description="Custom Event Handler to limit folder creation and deletion"
          Version="1.0.0.0"
          Scope="Web"
          Hidden="FALSE"
          xmlns="http://schemas.microsoft.com/sharepoint/">
    <ElementManifests>
        <ElementManifest Location="elements.xml" />
    </ElementManifests>
</Feature>
```

Figure 17.37
Elements.xml.

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Receivers
      ListTemplateId="101">
    <Receiver>
      <Name>My Custom Event Handler</Name>
      <Type>ItemDeleting</Type>
      <SequenceNumber>10001</SequenceNumber>
      <Assembly>MyEventHandler, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=9e7db4db3b276116</Assembly>
      <Class>MyEventHandler.ItemEventHandler</Class>
      <Data></Data>
    </Receiver>
    <Receiver>
      <Name>My Custom Event Handler</Name>
      <Type>ItemAdding</Type>
      <SequenceNumber>10002</SequenceNumber>
      <Assembly>MyEventHandler, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=9e7db4db3b276116</Assembly>
      <Class>MyEventHandler.ItemEventHandler</Class>
      <Data></Data>
    </Receiver>
    <Receiver>
      <Name>My Custom Event Handler</Name>
      <Type>ItemAdded</Type>
      <SequenceNumber>10003</SequenceNumber>
      <Assembly>MyEventHandler, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=9e7db4db3b276116</Assembly>
      <Class>MyEventHandler.ItemEventHandler</Class>
      <Data></Data>
    </Receiver>
  </Receivers>
</Elements>
```

Figure 17.38
Install and activate the feature.

```
stsadm.exe –o installfeature –name myEventHandler

stsadm.exe –o activatefeature –name myEventHandler –url http://devmoss/sites/samplesite
```

17.3.3 Testing the Event Handler

The event handler is now registered against all the document libraries in your site. Revisiting the registered events Web Part (Figure 17.39) shows that our event handler registrations have been applied to all document libraries, and each registration has been added to the event receiver definition named My Custom Event Handler.

Figure 17.39 *Registered events web part displaying new registrations.*

The following steps walk you through the process of testing the event handler registrations.

1. Sign in to SharePoint using the credentials of a user that is a member of the "Classified Users" group.
2. Navigate to the site in which you activated the event handler registration.

3. Create a new document library, for example, Classified Documents.
4. In the document library, create several new folders.
5. If you are logged in as a member of the "Classified Users" SharePoint group, you should be able to create folders.
6. Create or upload some documents in each subfolder and at the root of the document library.
7. The addition of documents to the document library should trigger the `ItemAdded` method, which adds a link to the document in the Classified Links list at the root of the site collection.
8. Navigate to the home page for the root of the site collection and verify that the Classified Links Web Part contains links to the documents that you created. Figure 17.40 shows an example of these results.

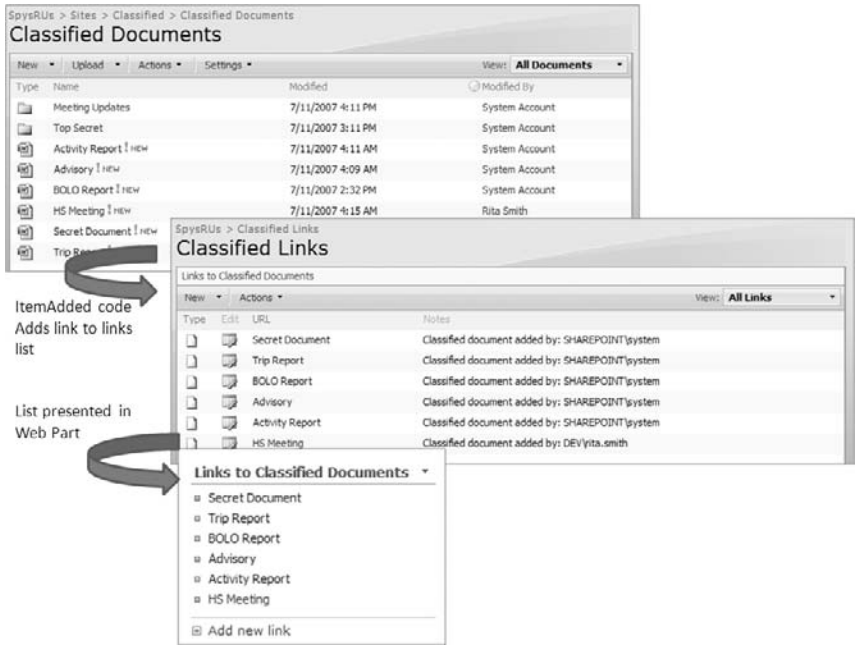


Figure 17.40 *Links added to Classified Links list.*

9. Navigate back to the site in which you activated the event handler registration and sign in as a user who is *not* a member of the Classified Users SharePoint group.
10. Navigate to the document library that you created in Step 2.

11. Try to create a folder. You should receive an error message similar to that shown in Figure 17.41, because the `ItemAdding` event checks to see if the current user is a member of the Classified Users group, in this case the user is not a member and therefore cannot create folders.

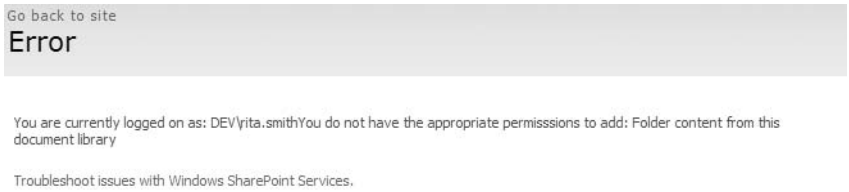


Figure 17.41
Error adding a folder.

12. Create or upload some documents in each sub folder, and at the root of the document library.
13. Once again, the addition of documents to the document library should trigger the `ItemAdded` method which adds a link to the document in the Classified Links list at the root of the site collection.
14. Navigate to the home page for the root of the site collection and verify that the Classified Links Web Part contains links to the documents that you just created.
15. Navigate back to the site in which you activated the event handler registration, remaining signed in as a user who is *not* a member of the Classified Users SharePoint group.
16. Navigate to the document library that you created in Step 2.
17. Try to delete a subfolder. You should receive an error message informing you that you do not have the appropriate delete permissions to perform this operation.
18. Try to delete a document that was created by another user. Once again you should receive an error message informing you that you are not the owner of the item and that you only have the ability to delete documents that you create.
19. Delete a document that was created by the currently logged-in user. Since the user is the owner, or creator, of the document, he or she will be able to delete the document from the document library regardless of the groups to which he or she belongs. Additionally, the deletion of a document triggers the `ItemDeleting`

event, which contains logic that removes any links to the document from the Classified Links list.

20. Navigate to the home page for the root of the site collection and verify that the links to the document that was just deleted have been removed from the Classified Links Web Part.
21. Navigate back to the site in which you activated the event handler registration and sign in as a user who is a member of the Classified Users SharePoint group.
22. Navigate to the document library that you created in Step 2.
23. Try to delete the same subfolder that you attempted to delete in Step 16. Now you should be able to delete the folder.

That completes our events exercise. At this point we hope you agree that the new and improved event architecture that ships with Windows SharePoint Services 3.0 provides an abundance of opportunities for customizing and extending the out-of-the-box functionality. Furthermore, the event architecture, used in conjunction with the other development components of SharePoint such as Features and workflow, provides you with the tools necessary to create sophisticated, flexible, and efficient solutions with minimal code.

18

InfoPath Forms Services and Workflow

So far, we have discussed the fundamental aspects of SharePoint with respect to building and deploying custom solutions. In this final chapter, we take a look at the forms and workflow development facets available with SharePoint Server 2007. We start by introducing you to InfoPath Forms Services, the forms capability built into SharePoint Server 2007. We then look at how to actually create and deploy your own custom form templates within your SharePoint farm. Standalone forms may be enough to meet specific needs within your organization, but chances are that you will want to associate a form with a specific business process or workflow. In the latter part of this chapter, we take a brief look at the workflow capabilities of SharePoint Server 2007 and Windows SharePoint Services 3.0 and show how you can pair this capability with your custom InfoPath forms to automate business processes. At this point, it should be noted that SharePoint Server 2007 workflow is, in general, a very large topic, and a thorough examination of the workflow capabilities is unfortunately beyond the scope of this book.

18.1 What is InfoPath Forms Services?

Microsoft Office InfoPath 2007 is a client application that allows you to develop sophisticated XML-based forms. The InfoPath client was first introduced with the Office 2003 suite, and as such provided integration capabilities with Windows SharePoint Services 2.0 and SharePoint Portal Server 2003. For example, using the InfoPath 2003 client, you can publish an InfoPath form directly into a SharePoint document library, allowing your forms to be accessible to users who have access to the document library. Although the forms technology was exciting and offered integration with SharePoint, it was not widely embraced in the SharePoint Portal Server 2003 arena. One of the factors that impeded its adoption was the fact that, to use the forms published to the SharePoint document library, you needed to have the full-blown InfoPath client installed on your desktop. This limitation alone introduced a number of obstacles for organizations, such as the cost of deploying InfoPath

to every user's desktop and the training costs related to teaching users InfoPath when all they really need to do is fill in a form. The good news is that the InfoPath technology has come a long way since then.

The Microsoft Office 2007 suite now comes equipped with a new version of the InfoPath client, named Microsoft Office InfoPath 2007, which has a host of new bells and whistles that make forms development fairly straightforward. In addition to the client, Microsoft offers InfoPath Forms Services (Forms Services) as part of its Enterprise Edition of SharePoint Server 2007; Forms Services is not available if you only have the SharePoint Server 2007 Standard Edition license. Forms Services is a shared service that allows SharePoint Server 2007 to store and manage forms and form templates from a central location. Forms Services overcomes the limitation of the previous InfoPath 2003 client by allowing form authors to create Web browser–compatible forms that can be stored in SharePoint and viewed and edited through the browser by users with the appropriate permissions. Users no longer need to have InfoPath installed on their desktops to use this functionality. Form templates published to Forms Services are referred to as browser-enabled form templates.

So far, we have mentioned forms and form templates, but we have not really explained the difference between these two terminologies. A form template (.xsn file) contains the basic structure and design of the form, similar in concept to a Word template (.dot) file. A form (.xml file) is an instance of the form template; a single form template can have multiple form instances. For example, let's suppose that you wanted to create a form that others in your organization would be able to access from a specific form library within SharePoint. Using InfoPath 2007, you would create a browser-enabled form template and publish the template to the SharePoint form library. When a user accesses the form library, he or she is able to create new forms based on your form template (Figure 18.1). Form templates can be stored centrally within your server farm or dispersed in various libraries throughout the SharePoint environment. When it comes to enterprise deployments, you may desire more control over the form templates that actually are published to the server. InfoPath Forms Services can be configured to only allow form templates to be deployed and centrally managed by the administrator.

While InfoPath Forms Services provides a mechanism for storing, managing, and serving up forms and form templates, the actual form authoring still takes place on the InfoPath client—specifically, Microsoft Office InfoPath 2007. InfoPath 2007 is tightly integrated with SharePoint Server 2007 and InfoPath Forms Services. For example, from the InfoPath 2007 client you can publish your form template directly to the server and associate the template with a specific content type in SharePoint. InfoPath 2007 comes equipped with a compatibility checker, a tool you can use to ensure that your forms are compatible with InfoPath Forms Services. Additionally, the

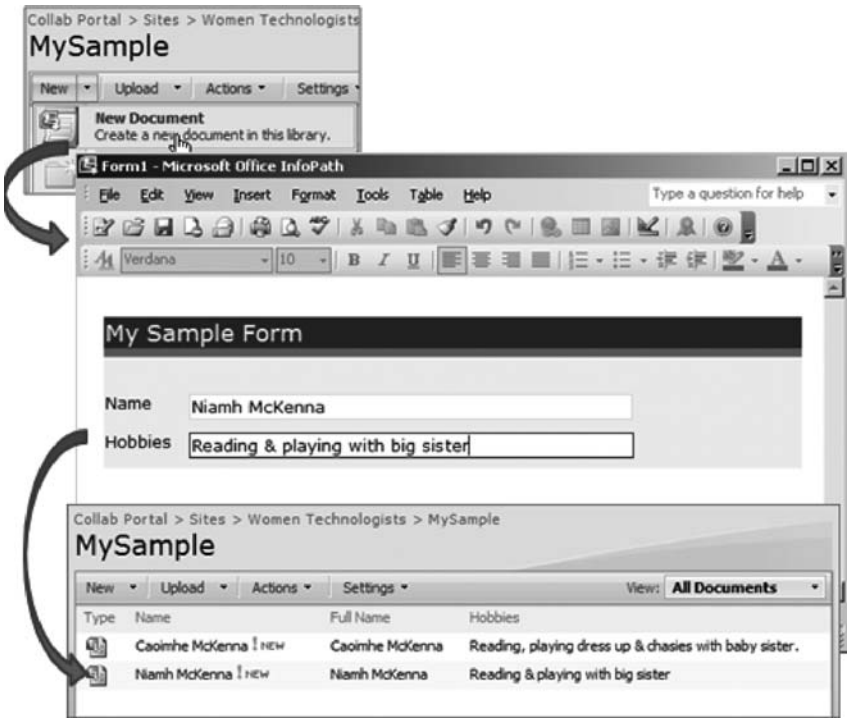


Figure 18.1
Published form template.

forms you develop are accessible from a variety of devices, including a Web-enabled PDA.

InfoPath Forms Services, in conjunction with the workflow capabilities offered with SharePoint Server 2007, provides developers with a powerful development platform from which to build applications and customize SharePoint to suit specific business needs.

18.1.1 Microsoft Office Forms Server 2007

We have mentioned that Forms Services comes with the enterprise edition of SharePoint Server 2007. But what about organizations that do not have SharePoint Server 2007, yet they would still like to access the Web-based forms functionality that Forms Services provides? Microsoft offers a standalone server product named Microsoft Office Forms Server 2007, which comes equipped with Windows SharePoint Services 3.0 and InfoPath Forms Services. However, Forms Server does not have the other service offerings that SharePoint Server 2007 provides, such as Excel Calculation Services, Office Search Server, and the Business Data Catalog.

18.2 Configuring InfoPath Forms Services

InfoPath Forms Services is a shared service that is enabled upon activation of the Office SharePoint Server Enterprise Feature. The Enterprise Feature is scoped at the Web application level and, as such, appears on the Manage Web Application Features page within the SharePoint central administration (Figure 18.2). If the Feature is inactive, then InfoPath Forms Services is not available to the site collection associated with the Web application. Additionally, if you have not yet configured SharePoint Shared Services within your server farm, Form Services is not available. Therefore, from a Forms Services configuration standpoint, the first tasks to perform involve configuring Shared Services and ensuring that the Enterprise Feature is activated. Once these tasks have been accomplished successfully, you can start publishing forms and form templates to Forms Services.

Is that really all there is to configuring Forms Services? Although you can start leveraging Forms Services right away, there are numerous configuration settings that allow you to fine-tune your implementation. For example, Forms Services has the concept of user-deployed form templates and administrator-deployed form templates. Developers, in this instance, fall into the "user" category. By default, users are allowed to deploy relatively simple form templates to Forms Services. What do we mean by relatively simple? A form template must have a specific set of characteristics to require it to be deployed by an administrator, such as a form template that contains managed code. Let's consider an organization that wants to control the form templates published to InfoPath Forms Services by requiring all form templates to be deployed to a central location by an administrator. To accomplish this, the administrator configures Forms Services to disallow users from publishing browser-enabled form templates by simply unchecking the Allow users to browser-enable form templates checkbox from the Configure InfoPath Forms Services page (Figure 18.3). With this setting unchecked, users can no longer deploy form templates to Forms Services. However, they can still deploy form templates that are not browser-enabled to SharePoint. Forms instantiated from these templates are only viewable and editable using the InfoPath client. We will revisit the deployment process for form templates later in this chapter.

Table 18.1 lists the configuration options available from the InfoPath Forms Services configuration page. The sister setting that works in conjunction with the Allow users to browser-enable form templates setting is the Render form templates that are browser-enabled by users setting. If you have had your SharePoint Server 2007 implementation running for quite some time and decide to take control over the browser-enabled form template deployment process to prevent users from being able to publish forms without administrator approval, you may still want to provide users with

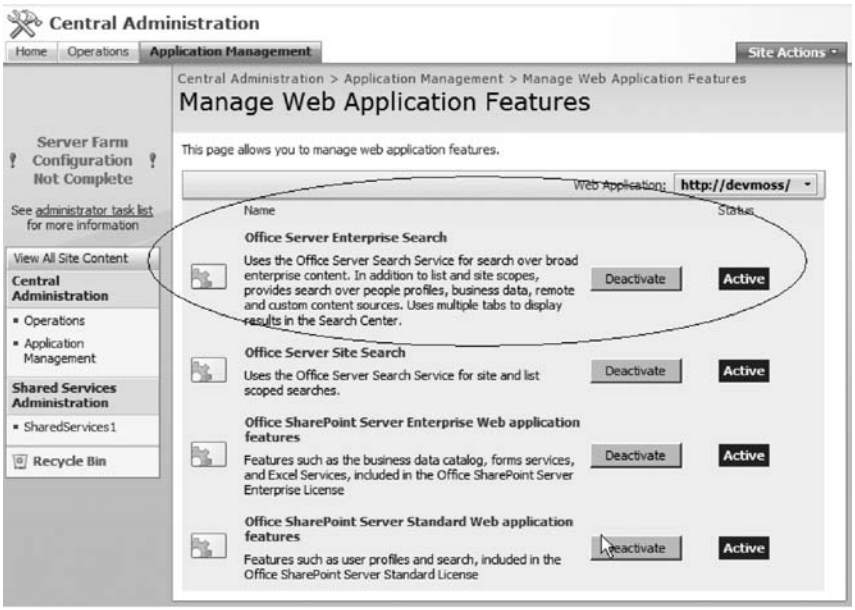


Figure 18.2
InfoPath Forms Services enabled.

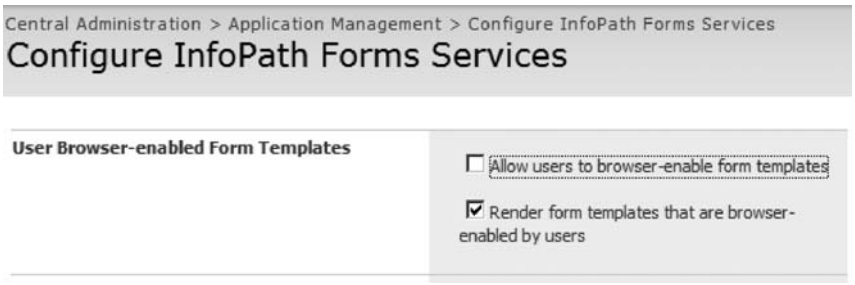


Figure 18.3
Preventing user deployment of form templates.

the ability to render forms that have been instantiated from pre-existing user-deployed form templates. Checking the Render form templates that are browser-enabled by users setting makes this possible. If you were to uncheck both settings, users would not be able to deploy browser-enabled form templates, nor would they be able to view or edit existing forms that were generated from the previously deployed template. It is unlikely that you will configure Forms Services so that the Allow users to browser-enable form templates setting is checked, and the Render form templates that are browser-enabled by users setting is unchecked. Doing so would mean that users are able to deploy browser-enabled form templates to Forms Services, but nobody would be able to view or edit those forms in a browser, which really defeats the purpose of deploying the template to Forms Services in the first place.

Data connections allow you to connect components of a form template, such as a list box, to an external data source, such as an Oracle database or a SharePoint server. When a user instantiates a form from the form template, the data connection retrieves data from the external data source and presents it to the user in the appropriate location on the form. With InfoPath 2007, it is possible to define both submit and query connections. A submit connection is one that would push data to the external data source from a form, whereas a query connection is one that retrieves data from the external data source for presentation in the form. There are a few special concerns with respect to how data connections are handled by form templates deployed to Forms Services. Firstly, the connection to the external data source is established by the server, not the client. This means that the identity used for the connection is the account that runs the Web application, which is ultimately a security risk, as the account that runs the Web application would typically have elevated privileges compared to the user initiating the request. Secondly, data connections require additional resources, which could potentially add load to the server and the network. Lastly, if the data source to which you are connecting is slow, then there is the potential for a denial-of-service attack by overloading the server with connection requests; this security concern can be mitigated by assigning the appropriate data connection timeout settings. The default data connection timeout defines the default timeout for all data connections on all forms; however, this setting can be overridden through code associated with the form template. Since the default data connection timeout can be overridden, administrators need a way of further defining timeouts to prevent denial-of-service attacks. This is where the Maximum data connections timeout setting comes into play; it sets the maximum connection timeout for a form to which all data connections must adhere, regardless of the default data connection timeout configured through code. The Data connection response size setting defines the maximum size of the responses that data connections are allowed to process, the default setting being 1500K. Once again, this helps prevent malicious attacks on the server; it also helps reduce the load on the server, ultimately improving performance.

Typically, connections to external data sources require some means of authentication, such as Windows-integrated authentication or basic authentication. If your data connection requires the use of basic or digest authentication, it means that the password that you use is susceptible to being intercepted; therefore, it is recommended that you use SSL to protect all such data connections. The Require SSL for HTTP authentication to data sources setting is enabled by default, and from a security perspective there's really no reason that you should change this setting. Data connections that use alternative means for authentication, such as Windows-integrated authentication, are not affected by this setting. If a data connection defines an HTTP connection but does not use SSL, the form will fail when the data connection is

invoked. When designing your form templates, you may be tempted to embed SQL usernames and password strings into the form; this practice is highly discouraged; it introduces a security risk, as passwords are exposed during form execution. To protect against this type of behavior, administrators can control whether embedded SQL authentication is allowed by using the Allow embedded SQL authentication setting; this setting is unchecked, or disabled, by default. If the administrator enabled this setting and a form is viewed or edited by a user, Forms Services executes data connections using the saved username and password. Instead of using embedded SQL authentication in your form templates, the best practice is to use instead a trusted connection that uses Windows-integrated authentication to identify the client to the SQL server.

We have briefly discussed the fact you can use data connections to connect to external data sources. Windows SharePoint Services 3.0 and SharePoint Server 2007 provide a special type of document library, known as a data connection library, that allows you to store data connection file definitions. A data connection file takes the format of an XML file with a .udcx or .xml extension and can hold authentication information, such as explicit usernames and passwords. The Allow user form template to use authentication information contained in data connection files setting determines whether Forms Services will permit a user-deployed form template to use the authentication information associated with a specific data connection. This setting does not affect administrator-deployed form templates. By default, user form templates permit access to data within the realm of the domain and zone settings as defined by Internet Explorer. Access to data external to the domain through the use of data connections requires cross-domain access rights. The Allow cross-domain access for user form templates that use connection settings in a data connection file setting is unchecked by default, but when checked, it allows user form templates that have server-defined data connections to access data from another domain.

Postbacks play a key role in forms development, regardless of whether you are developing ASP.NET forms, or InfoPath forms. A postback allows a form to exchange data with the server during form processing, requiring the user to wait for the process to complete. The use of unnecessary postbacks can considerably affect the overall server throughput. As a developer, you should do your best to prevent or eliminate as many postbacks as you can when developing your form templates, as this will improve the overall performance and user satisfaction of your forms. The InfoPath client provides a number of tools that can help you determine and limit the use of postbacks in your form templates. For example, the design checker includes a Show browser optimization messages setting (Figure 18.4) that you can set which lists the possible areas for browser optimization; this list identifies the postbacks that your form requires. Additionally, you can set the postback option

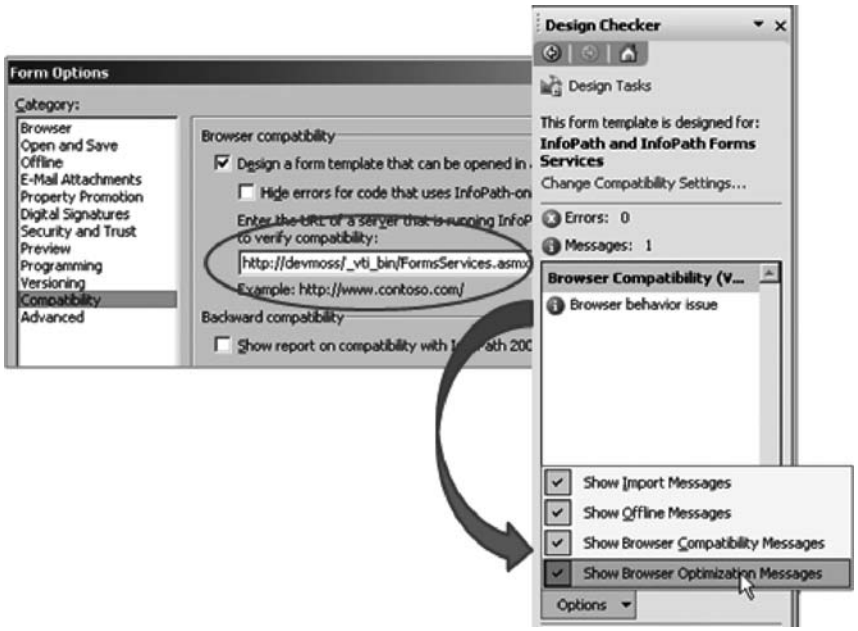


Figure 18.4
Show browser optimization messages.

on individual controls to manage when a postback occurs. Figure 18.5 shows the Browser forms settings tab on a typical Button control.

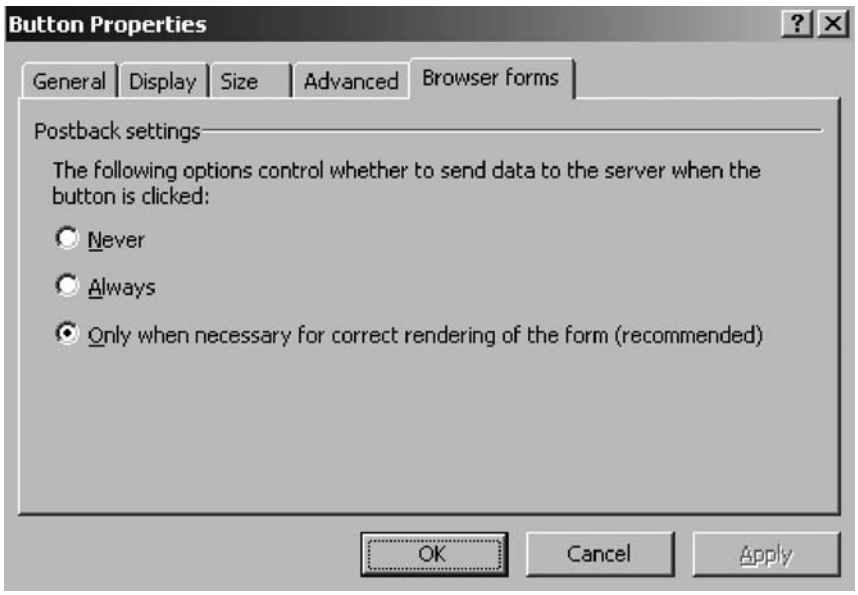


Figure 18.5
Postback settings on button control.

Forms Services provides a number of configuration options that can help an administrator control the form template postbacks in the SharePoint server farm. Administrators can limit the number of postbacks in a given form session using the Number of postbacks per form session state setting; the default setting is 75. A well-designed form template should not require more than 75 roundtrips to the server; in fact, the number should be considerably lower, and well under 30. Using the default setting, as soon as a form executes its 76th postback within a given session, the session is terminated and a server error is generated. In addition, an error is logged in the Windows event log, prompting the administrator to investigate the possibility of a malicious attack.

When a user updates a form in Forms Services, each action is recorded in a temporary log file, which means that the user can continue working on the form without having to wait for the form to reload with every action. During the next postback request, such as a user clicking the Update button, the log file is sent back to the server in the form of a Jscript data array, and the actions are replayed on the server. The server subsequently sends an updated Jscript data array back to the browser. This architecture means that when postbacks are required in Forms Services, they rarely involve a full page postback; instead, the HTML on the page can be updated incrementally, thereby reducing the amount of communication required between the client and the server. An administrator can define the maximum number of actions that can be replayed on the server in a single postback by setting the Number of actions per postback configuration setting. The default setting for this option is 200, which should be more than enough for even the largest form. This limitation helps prevents malicious attacks such as denial-of-service attacks.

By default, Forms Services uses the session state provided by the Shared Services Provider (SSP) to store the temporary data generated when a user fills out a form. This allows the Web front-end servers to remain stateless between roundtrips to the server, relieving the postbacks from carrying large amounts of session state data during form processing. When a new form is created or opened for editing in Forms Services, a new session is created containing information about the form. The *total* length of time that this session is kept alive is determined by the Active sessions should be terminated after ___ minutes configuration setting. The default value for this setting is 1440 minutes (one day), which should be ample time for a user to fill in a form. So why is this setting so high, and what do we mean by total? We have mentioned that the session state capability is provided by the SSP. The SSP maintains the overall session state timeouts. For example, if you navigate to the Configure Session State page (Figure 18.6), accessible from the Application Management page in central administration, you will see that you can specify the timeout associated with a given session. The setting translates to the following: a session will be terminated if it remains inactive for the time

denoted in the setting. So how is this setting different from the previously mentioned *Active sessions* setting? The Timeout setting, as shown in Figure 18.6, is a measurement of how long a session is *inactive*, as opposed to *active*; by default it is set to 60 minutes. When designing forms for optimal performance, it becomes quite difficult to detect when a form is actually inactive, since a user can be happily updating a form, which subsequently logs actions to be replayed during a required postback. This type of activity does not register as activity when it comes to measuring an active session, because it limits the server communication, which then ultimately leads to the perception that a session is idle. Once a session times out, the data associated with that session is lost, which means that the fields that a user has completed will have to be re-entered. Therefore, it is good practice to educate your users to save their forms intermittently as they enter their data; this is particularly important if your form has a lot of freeform text that would be cumbersome to re-enter. Saving the form during a session resets the counter against which the timeout session is measured while maintaining the session; it also initiates a postback. Any activity that requires a postback ultimately resets the session timer, thus preventing timeout. If we take into consideration the default timeout session (60 minutes) in conjunction with the number of postbacks allowed in a given session (75), we find that, based on these settings alone, a session could potentially be kept alive for 4500 minutes (about three days)! To prevent this from happening, Forms Services introduces another level of management in the form of the Active sessions should be terminated after ___ minutes setting. This setting defines the timeout for a session regardless of whether it is active or inactive, which effectively translates to the total session timeout. Obviously, the default setting of 1440 minutes is configurable.

We have mentioned that the session state stores data associated with a given session for a specific amount of time, which is determined by a number of timeout settings, but we have not discussed the size that this data can take. The Maximum size of form session state: ____ kilobytes setting sets a limit on the amount of data that can be stored to maintain a user's session. As far as protecting the server is concerned, this setting is of paramount importance because if it is set too high, the server is at risk of running out of disk space or memory. For example, the File Attachment control is a resource-intensive control and, depending on the size of the file attached, could require quite a lot of session state space. If there are numerous simultaneous sessions, requiring equally large session state resources, there is the potential of a server outage. Configuring the Maximum size of form session state: ___ kilobytes setting to the appropriate number, given your organization's requirements, will prevent such outages from occurring; the default setting of 4MB should be adequate for most environments.

When controlling the management of user sessions, Forms Services can be configured to use the Session State Service (the default setting, provided by the SSP) or the Form view (an ASP.NET view state). The Session State

Service is typically used when you have a large number of potential users or have low-bandwidth users. The Session State Service maintains the session data in the SQL Server database, whereas the data associated with a Form View session is maintained on the client browser, and all the data (up to 40kb by default) is included in each postback to the server. These settings map to the two settings in the Form Session State section of the Configure InfoPath Forms Settings page. Most enterprise deployments will leverage the default Session State Service to maintain the session state. If you have configured the setting to use Form View, and the session data exceeds the specified value (default 40kb), Forms Services will revert to using the Session State Service instead.

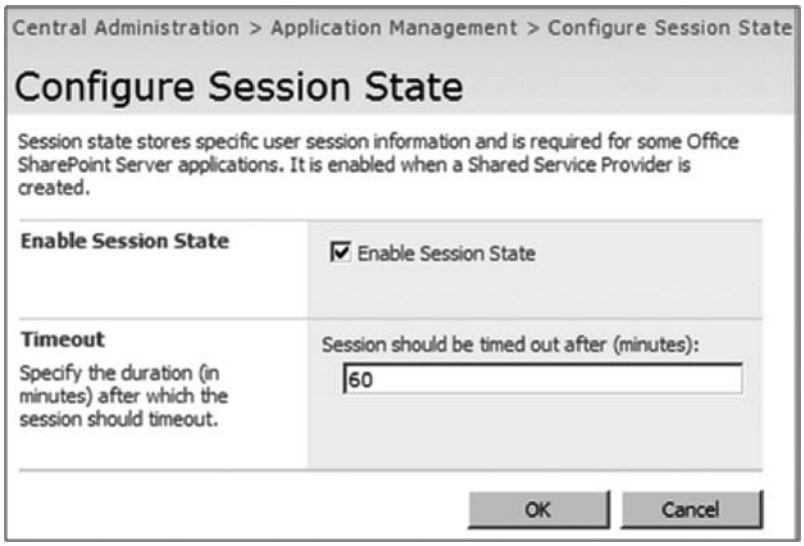


Figure 18.6
SSP Configure Session State.

As a developer, you will find it beneficial to be familiar with the Forms Services configuration settings on the server, as it will help improve the success rate of your forms. For example, if you try to deploy a form template using the InfoPath client and the administrator has disabled this capability in the configuration settings, you will instantly know that you need to package the form for administrator deployment. If the timeout settings have been set too low on the server, and you find that your form is timing out, then you will be able to pinpoint the problem with your administrator and come up with a resolution. When developing form templates for InfoPath Forms Services, it is recommended that you have access to a development SharePoint environment to which you can freely publish your templates to check for compatibility issues. The InfoPath Forms Services configuration settings in the development environment should mimic, within reason, the production environment.

Table 18.1 *Forms Services Configuration Settings*

Setting	Description
Allow users to browser-enable forms Default = enabled	Determines whether users will be allowed to deploy form templates to Forms Services. If this setting is unchecked, users must prepare the form templates for administrator deployment and then provide the template to the administrator.
Render form templates that are browser-enabled by users Default = enabled	Works in conjunction with the previous setting. If this setting is unchecked, users cannot view or edit forms instantiated from a user-deployed form template. If this setting is checked and the previous setting is unchecked, then users can still render forms instantiated from pre-existing user-deployed form templates even though they can no longer deploy form templates to Forms Services.
Default data connection timeout Default = 10000	Defines the default timeout for data connections from browser-enabled forms.
Maximum data connection timeout Default=20000	Defines the maximum timeout limit for browser-enabled forms, regardless of the setting defined for the default data connection timeout.
Data Connection Response Size Default=1500 kilobytes	Defines the maximum size of the payload returned from an external source in a data connection.
HTTP data connections Default=enabled	When enabled, ensures that data connections that use Basic or Digest authentication are required to use SSL encryption.
Allow embedded SQL authentication Default=disabled	When enabled, allows form templates to contain embedded SQL authentication. It is recommended that setting remain disabled; template designers should leverage trusted connections instead.

Table 18.1 *Forms Services Configuration Settings (continued)*

Setting	Description
Allow user form templates to use authentication information contained in data connection files Default=disabled	Determines whether a user-deployed form template can use the authentication information that is defined in a data connection file, such as an explicit user name and password, or SSO id. This setting does not affect administrator-deployed form templates.
Allow cross-domain access for user form templates that use connection settings in a data connection file Default=disabled	Determines whether user form templates are permitted to access data from another domain through the use of data connections.
Number of postbacks per form session state Default=75	Defines the threshold of postbacks for a form session. The form session is terminated if this threshold is exceeded.
Number of actions per postback Default=200	Defines the threshold of the number of actions that can be replayed to the server during a postback request.
Active sessions should be terminated after ___ minutes Default=1440 minutes	Defines the threshold of the amount of time that an active session should be kept alive, regardless of the activity level of the form.
Maximum size of form session state ____ kilobytes Default=4096 kilobytes	Defines the maximum size of the data that the session state maintains for a given session.
Select the location to use for storing form session state: Session State Service (best for low-bandwidth users) Default=enabled	Determines whether to use the Session State Service to store the session state information for the form session.

Table 18.1 *Forms Services Configuration Settings (continued)*

Setting	Description
Select the location to use for storing form session state: Form view (reduces database load on server). Default=disabled + session state size = 40kb	Determines whether to use the Form View (ASP.NET view state) to store the session state information for the form session. If form session state is larger than the specified value, the Session State Service will be used instead.

18.3 Introduction to InfoPath 2007

Microsoft Office InfoPath 2007 is the application to use to develop browser-enabled form templates that can be deployed to InfoPath Forms Services. InfoPath 2007 provides a rich user interface that is intuitive to most people already familiar with Microsoft development tools, such as Microsoft Visual Studio 2005 or even Microsoft Office InfoPath 2003. In addition to designing form templates, InfoPath 2007 can be used to create new instances of forms and update existing forms. If a user has InfoPath 2007 installed on his or her machine and launches a browser-enabled InfoPath form from SharePoint, the form opens by default in the InfoPath 2007 client; this behavior can be changed by the designer. Updates made to the form are then saved back to SharePoint, demonstrating the integration capabilities between the two products.

18.3.1 What is a Form Template?

Before we proceed any further, let's expand upon our earlier definition of a form template. An InfoPath form template is a basically a cabinet (CAB) file with a .xsn file extension; in fact, if you rename an InfoPath .xsn file with a .cab extension, you will be able to examine the files and components that comprise the template. Figure 18.7 shows the contents of a sample form template that has been published to Forms Services and contains multiple views. The manifest.xsf contains the form definition, which is automatically created and maintained by InfoPath and contains all the information about the form template, such as the location of all the files and resources associated with the template. Other files and information contained in the cabinet file include a collection of data, views, and resources, and a schema file. A form template is associated with a core data source, such as a SharePoint list, that stores the data entered onto the form by the user. Additional data sources can be associ-

ated with the form to help populate various components, such as a drop-down list box. A form template contains at least one visual representation of the form data, known as a view. When designing your forms, you may need to provide different views of your form to different users. For example, you may have views that represent different languages or that provide some users with a subset of data and others with a more elaborate view of the data. Views are basically XSL files associated with the form. In addition to the views, data, and schema files, you can associate resources with a form. A resource is content that you would like added to the form, such as images or picture files.

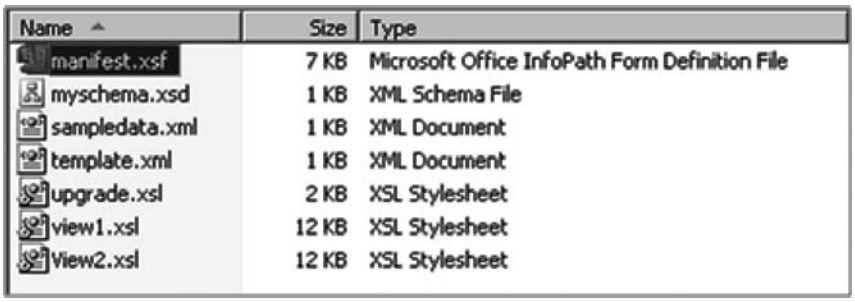


Figure 18.7
XSN file contents.

18.3.2 Form Template Designer

The template designer (Figure 18.8) provides the form designer with a set of design tasks that map to design task panes containing tools to help build the form templates. The design tasks provide a basic high-level task list for the types of activities a typical designer will perform during the template design phase. The first task is to design the layout and the basic structure that the form template will adopt, such as adding tables, columns, and rows. The next step involves adding the appropriate controls to the template, such as text boxes, list boxes, and buttons. After the controls have been added, the designer then checks to make sure all the required mappings with the data source are in place, such as mapping the InfoPath fields to the appropriate site columns in SharePoint. Once the fields have been mapped, the designer can move on to create any additional views required by the form. When the designer is happy with the form template design, he or she then checks the template for potential problems or inconsistencies *prior* to deployment of the form. This functionality is achieved using the Design Checker. When designing browser-enabled form templates, you can configure the browser compatibility settings with the URL of a server running Forms Services to verify the compatibility of your template. This setting enables the Design Checker options, which allow you to select the type of messages you wish the Design Checker to display. For example, the Show browser compatibility messages setting, when enabled, highlights any inconsistencies that it finds with Forms

Services. We visited this setting earlier in the chapter, and Figure 18.4 shows the compatibility options setting along with the Design Checker options. Once the issues highlighted by the Design Checker have been addressed, the form template is ready for deployment. We cover the deployment process later in this chapter.

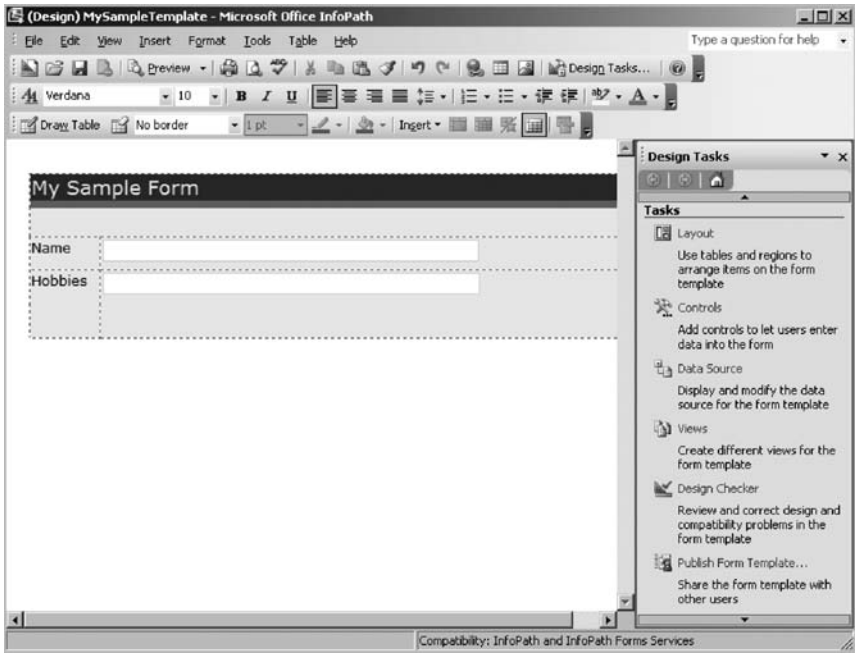


Figure 18.8 *Designing a form template in InfoPath 2007.*

Design once form templates

InfoPath embraces a *design once* philosophy when it comes to designing browser-enabled form templates, meaning that a browser-enabled form template can be used in both the full-blown client or in the browser, with no need to design a separate form to cater for each scenario. If you decide that your form is to only be used in InfoPath and is therefore not browser-enabled, then InfoPath provides access to an extended set of features and tools that can be used on your form. For example, message boxes, user roles, and combo boxes are not supported when designing browser-enabled forms that will run in Forms Services; however, these tools appear in the Controls task pane when designing form templates that are only to be used in InfoPath.

When you launch InfoPath 2007 you are presented with a number of options to get you started, one of which is to design a form template. Clicking the Design a Form Template link loads an associated dialog box from

which you can establish whether your form template will be browser-enabled. Figure 18.9 shows the Design a Form Template dialog box. Notice that toward the bottom of this form, there is a checkbox setting named Enable browser-compatible features only.

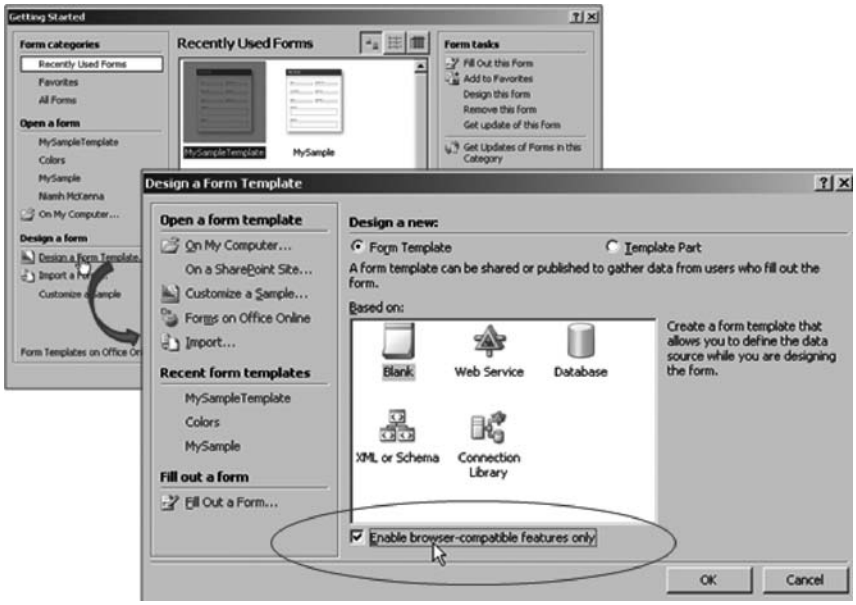


Figure 18.9
Creating a browser-enabled form template.

If you enable the browser-compatibility setting, the form you are designing is a browser-enabled form, and the task panes are filtered to present only those options that are supported by InfoPath Forms Services. Figure 18.10 shows the contents of the Controls task pane for both a browser-enabled form template (left) and a non-browser-enabled form template (right). Notice that there are a lot more controls listed on the right Controls task pane and that the left Controls task pane has a message at the bottom informing the designer that some of the controls have been hidden based on the compatibility settings of the form.

18.3.3 Template Parts

When creating a new form template, the Design a Form Template provides designers with the option to create a template part. A template part is a part of a template that can be reused from one form template to another. For example, if you were developing an automated account registration application that consisted of numerous form templates, you might find that several of the form templates contain a part that gathers address information and that the fields in this part remain consistent across the templates. In this

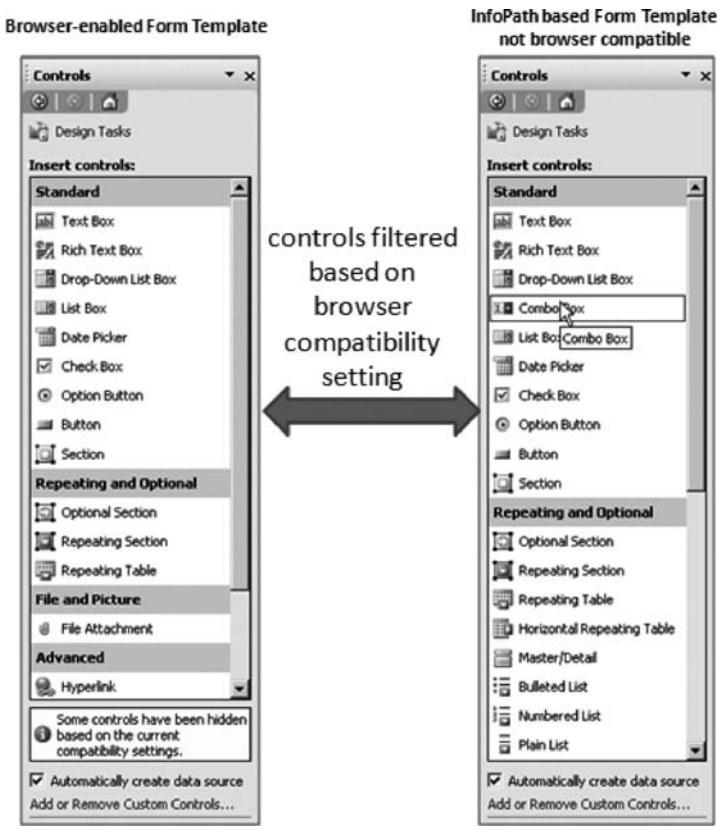


Figure 18.10 *Controls filtered on compatibility settings.*

instance, it would make sense to create a template part containing all the appropriate address fields and then insert this template part into each of the form templates that needed that information.

18.4 Data Connections

InfoPath form templates contain one or more connections to data sources. The default data source is the main data source and stores the data entered onto the form by the user. Additional *secondary* data sources can be associated with the form to help populate controls on the form, such as a drop-down list box. For example, you could have a list box on your form that retrieves its contents from a SharePoint list when the form is opened.

Data sources are configured through the use of data connections, which are also referred to as query connections. The data connections can be stored directly with the form template, or they can be stored in a Data Connection Library. A Data Connection Library (DCL) is a special type of document

library that comes with SharePoint Server 2007 and is used to store data connection files. Two types of data connection files can be stored in the DCL: an Office Data Connection (ODC) file and a Universal Data Connection (UDC) file. InfoPath form templates use UDC-type data connections, and the files generally have a .udcx or a .xml extension. Figure 18.11 shows the contents of a UDC file generated by InfoPath 2007 and stored in a local DCL in a SharePoint site. This specific data connection only retrieves information from the SharePoint list; it does not update data in the list.

Figure 18.11
Universal data connection (.udcx) file.

The UDC file format has been around since FrontPage 2003. If you browse the Microsoft site for information on the schema, you will find that there are actually two versions of the UDC file format; InfoPath uses the V2.0 version. If you want to create a UDC data connection file for use with InfoPath form templates, you can build the .udcx file from scratch using Notepad and the examples provided by MSDN, or you can leverage the Data Connection Wizard that comes with InfoPath 2007 (Figure 18.12). The Data Connection Wizard provides tools with which you can set up the connection and store it directly with the form template. The wizard extracts all the relevant information needed to perform your query.

You can then use the wizard to convert this data connection into a UDC file and store it directly into a DCL on the SharePoint server (Figure 18.13). If you can store the connection information with the form template, why would you want to go to the bother of creating a UDC file? One of the main

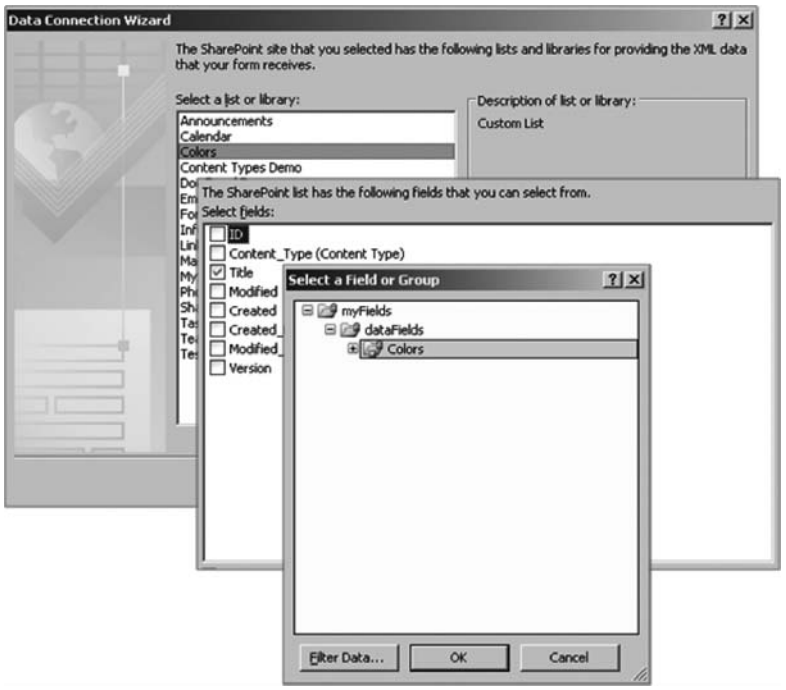


Figure 18.12
Data Connections Wizard.

advantages of using UDC files to store data connection information is that it allows you to share the same data connection information among many form templates, without requiring each template to hold its own copy of the connection data (as was the case with InfoPath 2003). It also greatly improves the maintenance and upkeep of connection information; should the data source move, you only have to update the data connection information in one file instead of editing each form template that leverages the data connection. Additionally, data connections stored in a UDC file support cross-domain connections, which means that you can connect to data outside of the current domain boundary. Such connections are not possible *unless* the data connection uses a UDC file. Figure 18.14 shows the error displayed to the user when a form template attempts to make a cross-domain connection without the use of a UDC file. If we were to replace the data connection causing the error with a data connection that uses a UDC file, we would still receive this error if the cross-domain access setting in the Forms Services configuration settings was not enabled. This setting is disabled by default. However, remember that if you enable cross-domain access for user form templates, it applies to *all* user form templates, which if misused has the potential to cause unscheduled downtime. If your form template needs to make a cross-domain connection, you should either make the template a fully trusted form template or ask your administrator to create a data connection in the centrally managed connection library.

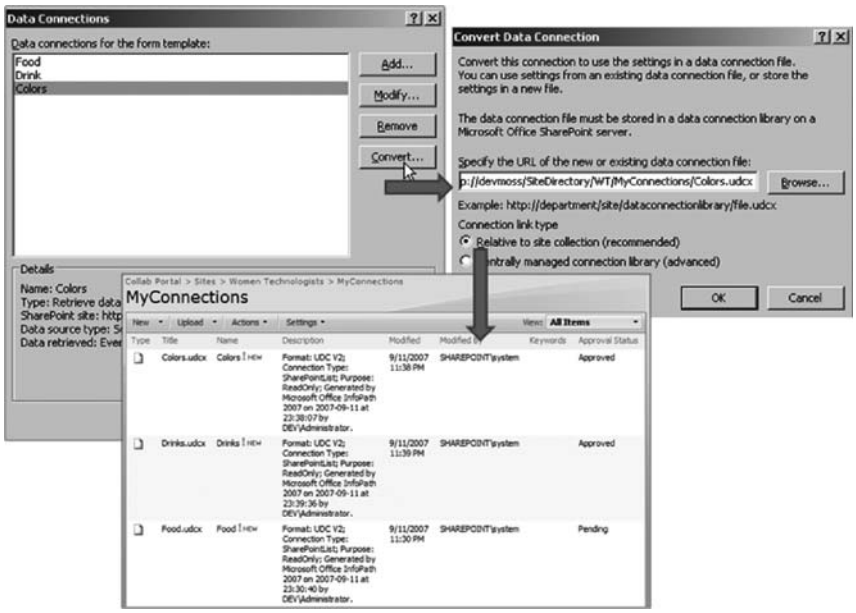


Figure 18.13 *Convert data connection to UDC file.*

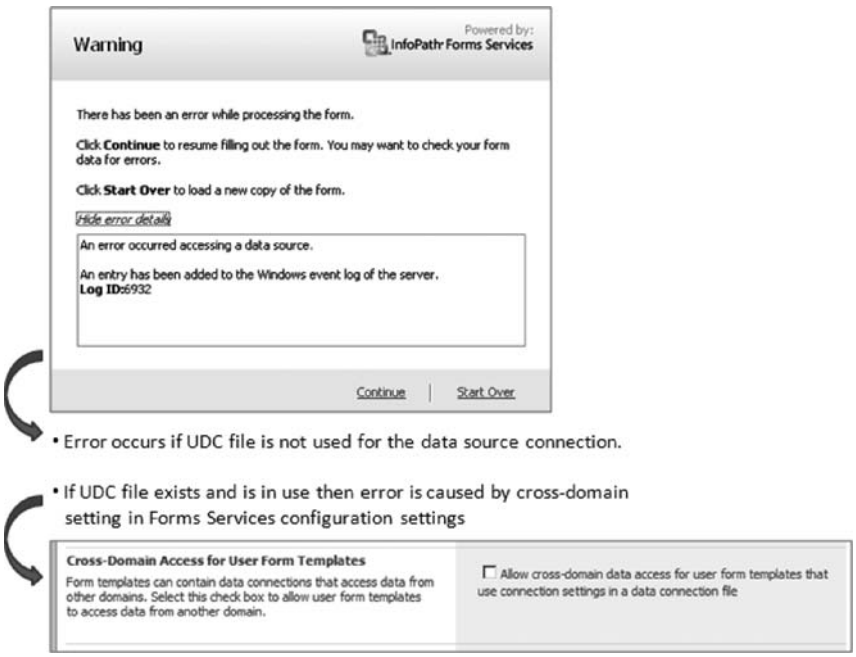


Figure 18.14 *Cross-domain data connection error.*

When you convert a data connection into a UDC file you are given the option of selecting the connection link type. The connection link type to the

DCL can be either a link type that is relative to the site collection or a link type that connects to the centrally managed connection library. The link type you select depends on the type of data connection you are implementing. For example, if you use a relative link, it means that the DCL is not dependent on the site collection. This is an advantage from a portability perspective, as it allows you to deploy your form templates to other site collections as long as that site collection has a DCL with the same name containing a UDC file of the same name. If you need to share a data connection across multiple site collections but some of the connection settings, such as the authentication settings, differ depending on the site collection, the site relative link would be ideal. If, on the other hand, you need to share the same data connection across multiple site collections, with no change in the data connection information between site collections, then using the relative link would result in a lot of unnecessary duplicate data connections across the server farm. A better solution in that instance would be to use the centrally managed connection library, which is accessible from the Application Management page in the central administration Web site.

The centrally managed connection library is managed and maintained by the administrator and provides a central repository for storing data connection files. Data connections defined in the central library have more flexibility than those stored in a site collection DCL. For example, cross-domain data connections defined in the centrally managed connection library can be leveraged by form templates without requiring the form template to be fully trusted. Another advantage is that data connections defined in the centrally managed library are accessible to all site collections across the entire server farm.

While storing data connections in a centrally managed library has many advantages, it does come with some administrative overhead. Only administrators can add data connections to the centrally managed library, and form templates that reference connections in the centrally managed library must be deployed by an administrator. While these constraints may sound cumbersome from a development perspective and do require more coordination, they actually protect the security of your form templates because you do not need to make your form templates fully trusted unless absolutely necessary. Another benefit is that, by default, users who have been granted the contributor role in SharePoint can upload data connection files to a DCL, whereas only farm administrators can upload data connection files to the centrally managed library.

18.5 Creating a Browser-Enabled Form Template

Now that we have reviewed the various capabilities offered by InfoPath Forms Services and InfoPath 2007, let's run through a quick example of cre-

ating a form template In this example, we'll create a simple form (Figure 18.15) that captures the name of the user and his or her hobbies, favorite color, favorite food, and favorite drink. The fields will map to site columns in SharePoint, and the choices for the favorite lists will be pulled directly from lists in SharePoint, thereby demonstrating the use of data connections.

Preparatory steps

Before we begin the exercise, there are a number of preparatory steps that you need to perform:

1. Launch SharePoint Server 2007 and create a new Team site.
2. From the Site Actions menu in the newly created Team site, click Create.
3. From the Libraries section, click Data Connection Library to create a new DCL.
4. Name the DCL MyConnections.
5. In the same Team site, create three custom lists named Color, Drink, and Food.
6. Populate each of these lists with data of your choosing.
7. From the Site Actions menu, click Site Settings.
8. From the Galleries section, click Site columns.
9. Create three new site columns of type single line of text: Color, Drink, and Food.

Step 1: Create the form template layout

1. Launch Microsoft Office InfoPath 2007. The Getting Started dialog should appear.
2. From the Design a Form section, click Design a Form Template. The Design a Form Template dialog appears.
3. In the Design a New section of this dialog, ensure that the Form Template option is selected, the Blank template is selected, and the Enable browser-compatible features only setting is checked.
4. Click OK. The new blank form template should open in design mode.
5. From the Design Tasks task pane on the right of the user interface, click the Layout task.
6. From the Insert Layout Tables section, click Table with Title.

Figure 18.15 *Hobbies and Favorites form.*

7. In the form template window, click on the title of the form and change the title to Hobbies and Interests.
8. Click on the row underneath the title and enter a carriage return.
9. From the Insert Layouts Tables section, click Two-Column table.
10. Right-click the right column of the table and insert four rows below the current row.
11. Click the first cell in the first row of the table and enter the text: Name:
12. Click the first cell in the second row of the table and enter the text: Hobbies:
13. Click the first cell in the third row of the table and enter the text: Favorite Color:
14. Click the first cell in the fourth row of the table and enter the text: Favorite Drink:
15. Click the first cell in the fifth row of the table and enter the text: Favorite Food:
16. Resize the column to the maximum width required for the text to display.

Step 2: Add the secondary data sources

1. From the top of the Layout task pane, click the Design Tasks link.

2. From the Design Tasks task pane, click the Data Source link to open the Data Source task pane.
3. From the Actions section at the bottom of the task pane, click the Manage Data Connections…. link. The Data Connections dialog appears.
4. Click the Add button. The Data Connection Wizard appears.
5. Click the options Create a new connection to: and Receive data.
6. Click Next.
7. Select the SharePoint list or library option as the source of your data.
8. Click Next.
9. Enter the URL to the site that contains the Color list to which you would like to connect.
10. Click Next.
11. From the list of available libraries, click the Color form library.
12. Click Next.
13. From the list of SharePoint fields, select the Title field, and click Next.
14. Leave the Store copy of the data in the form template setting unchecked. Offline query support does not apply to browser-enabled form templates running in the browser. However, if you want to give users who will be using the full InfoPath client the ability to fill in the form offline, then you can check this setting.
15. Click Next.
16. Enter the name of the data connection: Colors
17. Leave the Automatically retrieve data when the form is opened setting checked.
18. Click Finish. The Data Connections dialog should appear and the Colors data connection should appear in the list.
19. Repeat steps 20 through 34 to create a data connection named Drink, to connect to the Drink SharePoint list.
20. Repeat steps 20 through 34 to create a data connection named Food, to connect to the Food SharePoint list.
21. Leave the Data Connections dialog open.

Step 3: Convert the secondary data sources to UDC files

At this point you should have three data connections listed in the Data Connections dialog. Currently, these data connections are stored directly in the form template. We need to convert these templates to UDC files and store them in a data connection library on the SharePoint server.

1. From the Data Connections dialog, select the Colors data connection.
2. Click Convert. The Convert Data Connection dialog appears.
3. In the URL field enter the URL to the MyConnections DCL and suffix the URL with the name of the data connection file. For example: http://devmoss/sitedirectory/myteamsite/MyConnections/Colors.udcx
4. From the Connection link type section, select the Relative to site collection option.
5. Click OK. If you navigate to your MyConnections DCL, you will notice it contains a new data connection file named Colors.udcx with an approval status of Pending.
6. Repeat steps 1 through 5 for the Drink data connection, creating a new UDC file named Drink.udc
7. Repeat steps 1 through 5 for the Food data connection, creating a new UDC file named Food.udc
8. Navigate to the MyConnections DCL and approve each of the data connections that you created.

Step 4: Add controls to the page

Now that we have the data sources configured, let's add the field controls to the form template. At this part of the exercise, we are simply adding and naming the fields. We will revisit the data source mapping tasks in the next step.

1. From the top of the Data Source task pane, click the Design Tasks link.
2. From the Design Tasks task pane, click the Controls link to open the Controls task pane.
3. From the list of standard controls, drag and drop a Text Box control into the second cell in the first row of the table adjacent to the Name text.
4. Right-click the newly added field and click Text Box Properties.

5. Set the Field Name to txtName and check the box to ensure that this field cannot be blank.
6. Click OK.
7. From the list of standard controls, drag and drop a Text Box control into the second cell in the second row of the table adjacent to the Hobbies text.
8. Right-click the newly added field and click Text Box Properties.
9. Set the Field Name to txtHobbies and click OK.
10. From the list of standard controls, drag and drop a drop-down list control into the second cell in the third row of the table adjacent to the Favorite color text.
11. Right-click the newly added field and click Drop-Down List Box Properties.
12. Set the Field Name to lstColor.
13. Click OK. We will revisit this field later to map the data source.
14. From the list of standard controls, drag and drop a drop-down list control into the second cell in the fourth row of the table adjacent to the Favorite Drink text.
15. Right-click the newly added field and click Drop-Down List Box Properties.
16. Set the Field Name to lstDrink.
17. Click OK. We will revisit this field later to set up the data connection.
18. From the list of standard controls, drag and drop a list box control into the second cell in the fifth row of the table adjacent to the Favorite Food text.
19. Right-click the newly added field and click List Box Properties.
20. Set the Field Name to lstFood.
21. Click OK. We will revisit this field later to set up the data connection.

Step 5: Map controls to data sources

The field controls are on the page. If we were to publish/deploy the form as it is, the Color, Drink, and Food fields would be blank. To populate these fields, we need to connect them to the appropriate data source. We will leverage the UDC data connections that we created in step 3.

1. From the form template, right-click the lstColor field, and click Drop-Down List Box Properties.
2. From the List Box Entries section, select the Lookup values from an external data source option.
3. In the Data Source field, drop-down select the Colors data source.
4. Click the button to the right of the Entries field. The Select a field or Group dialog appears.
5. Expand the tree and select the Colors repeating group.
6. Click OK. You will be brought back to the Drop-Down List Box Properties dialog.
7. At the bottom of the dialog, ensure that the Show only entries with unique display names setting is checked. The list box properties dialog settings should be similar to that shown in Figure 18.16.
8. Repeat steps 1 through 7 for the lstDrink field mapping the control to the Drink data source.
9. Repeat steps 1 through 7 for the lstFood field mapping the control to the Food data source.
10. Save the form template to your local computer. Give it the name: PersonalInterests.xsn.

Step 6: Verify the form

Now all that remains is for us to check the form template for any known inconsistencies prior to deployment.

1. From the top of the Controls task pane, click the Design Tasks link.
2. From the Design Tasks task pane, click the Design Checker link to open the Design Checker task pane.
3. Click the Change the Compatibility Settings link. The Form options dialog appears with the Compatibility tab active.
4. In the Browser Compatibility section, ensure that the Design a form template that can be opened in a browser or InfoPath option is checked.

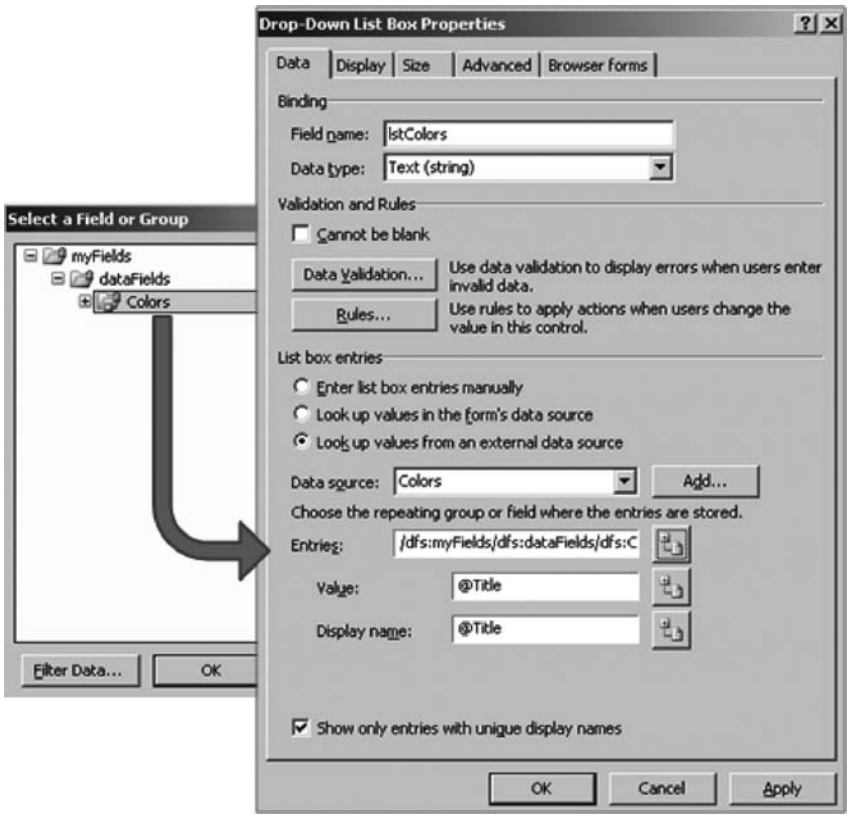


Figure 18.16
Mapping to the Colors data source.

5. In the Browser Compatibility section, enter the URL of the server running Forms Services—for example, http://devmoss/_vti_bin/FormsServices.asmx.
6. Click OK.
7. From the Design Checker task pane, ensure the Verify on Server setting is checked.
8. Click the Refresh button.
9. If the Design Checker detects any potential problems, the Options button will be enabled.
10. If enabled, click the Options button and ensure that the Show Browser Compatibility Messages setting is checked.
11. Address any errors or messages that appear in the Design Checker message area.

You are now ready to deploy your form.

18.6 Deploying and Managing InfoPath Forms

As discussed at the beginning of this chapter, when it comes to deploying form templates to Forms Services, there are two paths you can follow: user-deployed forms or administrator-deployed forms. With respect to form template deployment, developers fall into the "user" category. As long as the Allow users to browser-enable form templates setting (Figure 18.17) is enabled in the Forms Services configuration settings, users can deploy form templates to Forms Services. Form templates that have one of the following characteristics qualify for deployment by an administrator:

- The security level of the form template is set to full trust.
- The form template contains managed code. This goes hand-in-hand with the previous setting, in that browser-enabled form templates require full trust security if they contain code.
- Cross-domain data connections are defined directly in the form template; the form template does not leverage UDC files.
- The data connections used by the form template are located in the centrally managed connection library.

User-deployed form templates are stored in the SharePoint content database, whereas administrator-deployed form templates are stored on the local disk of the Web front-end servers. Consequently, administrator-deployed form templates provide better performance than user-deployed form templates, as they do not have to be retrieved from the SQL database.

Figure 18.17 *User frowser-enabled form templates settings.*

18.6.1 User Form Template Deployment

Assuming that your form template does not adopt any of the aforementioned characteristics, your form template can be deployed to any number of locations. The form template deployment to Forms Services is initiated by clicking the Publish option from the InfoPath File menu, followed by selecting the SharePoint option and entering the URL of the site in which you want to deploy the form template (Figure 18.18). Once you enter the URL and click next, you are presented with a number of deployment options. Table 18.2 lists

Table 18.2 *Deployment Options*

Deployment Option	Description
Document Library	Deploy the form template as a template in a form library (a type of document library) that will store forms based on the form template. Users can open and fill out forms in the library. You can specify which fields in the template will appear as columns in the library.
Site Content Type	Deploying a form template to a site content type means that the template can be used in multiple libraries and sites. As with the Document Library option, you can specify the fields in the template that appear as columns in the library to which the content type is applied.
Administrator-approved Form Template	Allows you to prepare your form template so that it can be subsequently deployed by an administrator.

these options, along with a description of each. Next, we'll walk through each option by deploying the form we created previously.

Deploying to a document library

When deploying a form template to a form library, you can create your form library up front and map the form template to it during deployment; however, in this particular example, InfoPath will conveniently create the form library for us as part of the deployment process.

1. Launch InfoPath 2007 and open the form template that you created in the previous section.
2. From the File menu, click Publish. The Publishing Wizard appears (Figure 18.18).
3. Select the option to publish the form template To a SharePoint Server with or without InfoPath Forms Services.
4. Click Next.
5. Enter the URL of the site in which you want the form library to be created. This site should have access to the site columns created in the preparatory phase.

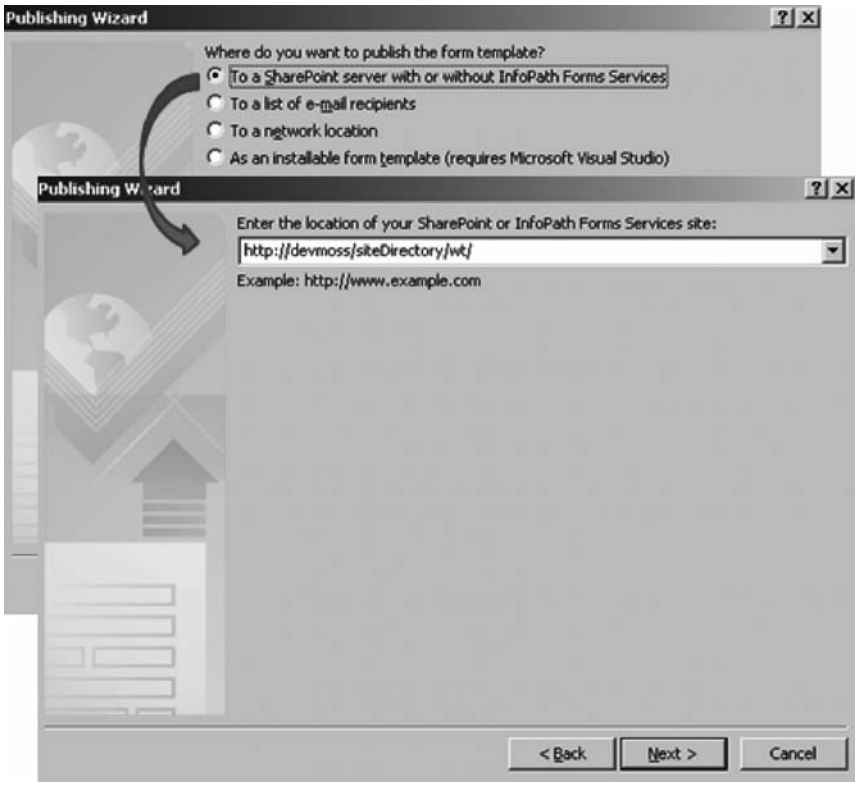


Figure 18.18 *Deploying a form template to Forms Services.*

6. Click Next.
7. Ensure the Enable this form to be filled out using a browser setting is checked.
8. Select the Document Library option.
9. Click Next.
10. From the What do you want to do? section, select the Create a new document library option and click Next.
11. In the name field, enter the name of your form library. For example, Personal Interests.
12. In the description field, enter a description for you form library.
13. Click Next.
14. Click the Add button to add columns to the form library and map them to the form template.

15. Select the txtName field and map it to the Full Name site column, which is located in the Core Contact and Calendar Columns site column group (Figure 18.19).

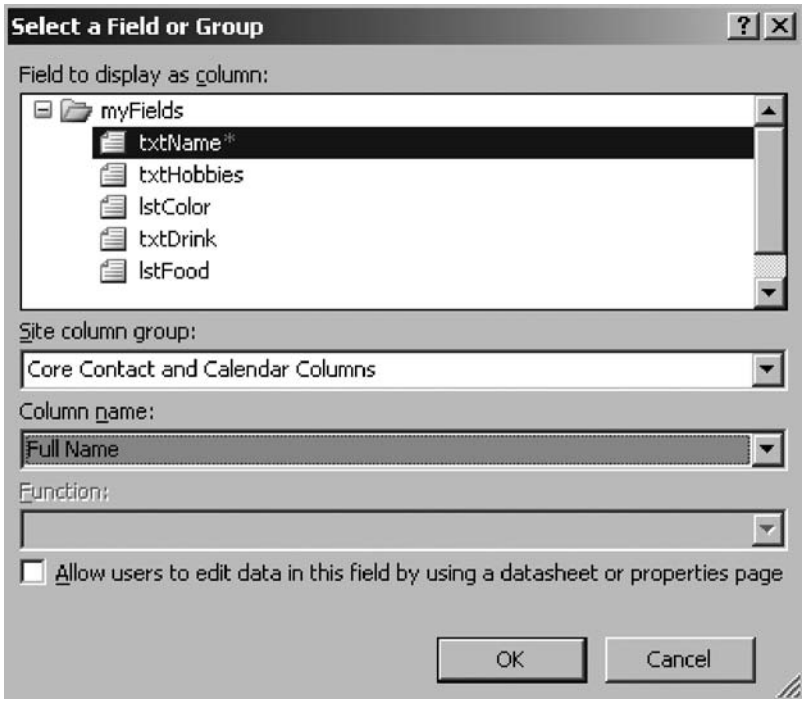


Figure 18.19
Mapping txtName field to the Full Name site column.

16. Click OK.
17. Click Add.
18. Select the txtHobbies field and map it to the Hobbies site column, also located in the Core Contact and Calendar Columns site column group.
19. Click OK.
20. Click Add.
21. Select the lstColor field and map it to the Color site column that you created during the preparatory phase. Custom site columns are generally located in the Custom Columns site column group.
22. Click OK.
23. Click Add.

24. Select the lstDrink field and map it to the Drink site column that you created during the preparatory phase. Custom site columns are generally located in the Custom Columns site column group.
25. Click OK.
26. Click Add.
27. Select the lstFood field and map it to the Food site column that you created during the preparatory phase. Custom site columns are generally located in the Custom Columns site column group.
28. Click OK. The fields mapped should be similar to those shown in Figure 18.20.

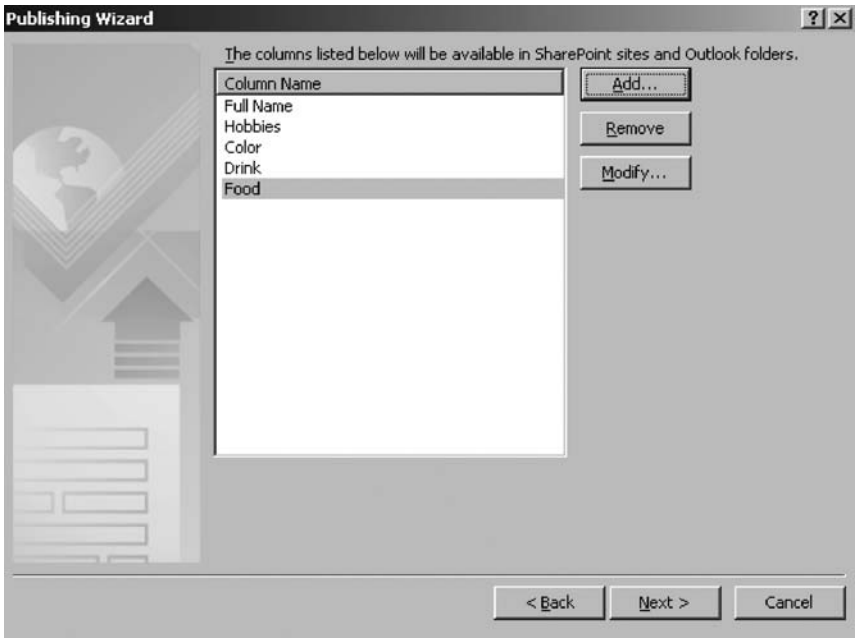


Figure 18.20
Columns added and mapped.

29. Click Next.
30. Verify the form information, and then click Publish.
31. When the form template deployment is completed successfully, the Publishing Wizard presents a confirmation page. Review this page, taking note of the location of the form template.
32. Ensure the Open this document library setting is checked.
33. Click Close.
34. The form library created by InfoPath appears (Figure 18.21).

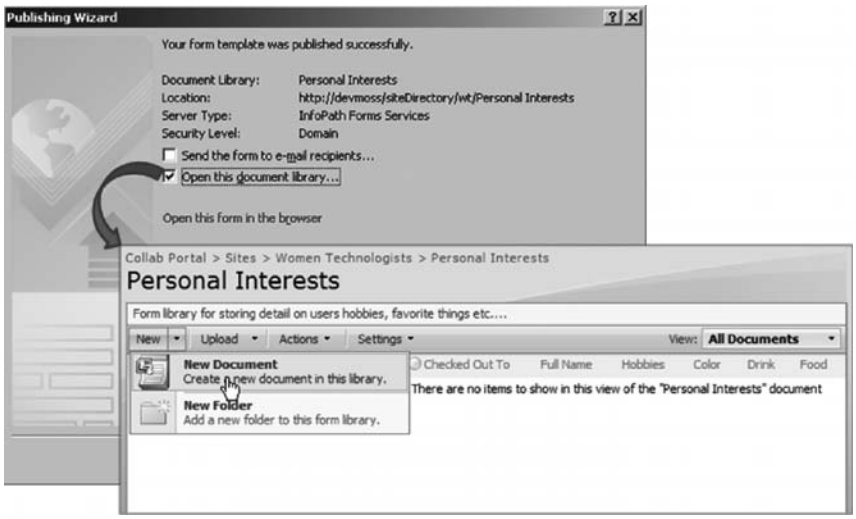


Figure 18.21 *Form template deployed successfully.*

Your form template is now successfully deployed, and you can create and edit forms to your heart's desire. Notice that the appropriate columns have been added automatically to the library, and that when you click the New Document option, a form is instantiated from your form template. At this point, if you have InfoPath 2007 installed on your machine, your new form is opened in InfoPath. An easy way of ensuring that your forms open in the browser, which is great for testing purposes, is to configure the form library to open forms in the browser:

1. Navigate to the form library that hosts your form template.
2. From the Settings menu, click Form Library Settings.
3. From the General Settings section, click Advanced settings. The Advanced Settings page appears.
4. From the Browser-enabled Documents section, click the Display as Web page option.
5. Click OK.

Now when you create or edit a new form, the form will be opened in the browser instead of the InfoPath client (Figure 18.22). Notice that the list box fields are populated with the data retrieved from the SharePoint lists. If you were to add or remove content to the SharePoint lists, these changes would be reflected upon a refresh of the form.

If you are curious as to where the actual form resides, simply set the view settings of the Form Library to the Explorer View and expand the Forms sub-folder. In this folder, you will see a file named Template.xsn, which is actually

Figure 18.22
Form opened in browser; list boxes populated.

your newly deployed form template. InfoPath renames the form at deployment time and uses it to replace the default Template.xsn that is typically associated with a new form library.

Deploying to a site content type

Once again, when deploying a form template to a site content type you can create the content type up front and map the form template to it during deployment. However, in this particular example, InfoPath will conveniently create the site content type for us as part of the deployment process.

1. Create a document library to store the form template to which your site content type form will be linked.
2. Launch InfoPath 2007 and open the form template that you created in the previous section.
3. From the File menu, click Publish. The Publishing Wizard appears (Figure 18.18).
4. Select the option to publish the form template To a SharePoint Server with or without InfoPath Forms Services.
5. Click Next.

6. Enter the URL of the site in which you want the form library to be created. This site should have access to the site columns created in the preparatory phase—for example, http://devmoss/sitedirectory/myteamsite/.
7. Click Next.
8. Ensure the Enable this form to be filled out using a browser setting is checked.
9. Click the Site Content Type (advanced) option.
10. Click Next.
11. From the What do you want to do? section, select the Create a new content type option and click Next.
12. In the name field, enter the name of your content type—for example, Personal Interests.
13. In the description field, enter a description for your content type.
14. In the location and file name for the form template field, enter the URL to the document library that you created in the first step in this series of tasks, and suffix this URL with the file name of the form template—for example, http://devmoss/sitedirectory/myteamsite/mydoclib/PersonalInterests.xsn.
15. The Column list should still contain the columns that you mapped when publishing to the document library. You can choose to keep these settings or, if you'd rather repeat the exercise, remove all the columns and follow the same steps as before.
16. Click Next.
17. Verify the form information and then click Publish.
18. When the form template deployment is completed successfully, the Publishing Wizard presents a confirmation page. Review this page, taking note of the location of the form template.
19. Click the Manage this Content Type link to view the newly created content type.
20. Click Close.

You have successfully deployed your form template to a site content type. This site content type can now be associated with a form library, thereby inheriting the columns and template information from the content type. To see the site content type in action, create a new form library through the standard SharePoint browser user interface, and then set the new site content type as the default content type for the library (Figures 18.23 and

18.24). You will need to enable the management of content types for the library before you can actually do this. Notice that the action on the New menu reflects the name of the site content type: Personal Interests; which differs from the experience achieved when deploying a form template to a document library (Figure 18.21).

Figure 18.23 *New Personal Interests site content type.*

Figure 18.24 *New site content type associated with form library.*

Preparing a form template for administrator deployment

If your form template has one of the characteristics mentioned at the start of the deployment section, it needs to be deployed by an administrator. However, you must take certain deployment steps to prepare your form template for administrator approval and ultimate deployment; the publishing wizard walks you through the required tasks.

1. Launch InfoPath 2007 and open the form template that you created in the previous section.
2. From the File menu, click Publish. The Publishing Wizard appears (Figure 18.18).
3. Select the option to create an administrator-approved form template.
4. Click Next.
5. Enter the path to the file share or folder in which you want the form template to be saved—for example, C:\mytemplates\PersonalInterests.xsn. This location should be one that the administrator has access so that he or she can retrieve the form. In addition, the folder must already exist; InfoPath does not create the folder during the process. Once you have saved the form template, you can e-mail it to the administrator instead of requesting the administrator to retrieve the file from a specific location.
6. The Column list should still contain the columns that you mapped when publishing to the document library. You can choose to keep these settings or, if you'd rather repeat the exercise, remove all the columns and follow the same steps as before.
7. Click Next.
8. Verify the form information and then click Publish.
9. When the form template deployment is completed successfully the Publishing Wizard presents a confirmation page (Figure 18.25). Review this page, taking note of the location of the form template, the target site, and the security level.
10. Notify your administrator that the form template is ready for approval.

You have successfully prepared your form template for administrator approval. Now all you have to do is sit back and wait for the administrator to

evaluate your form, approve it, and deploy it in the farm. The next section walks you through the various activities that need to be performed by the administrator to make your form available for use in your target site.

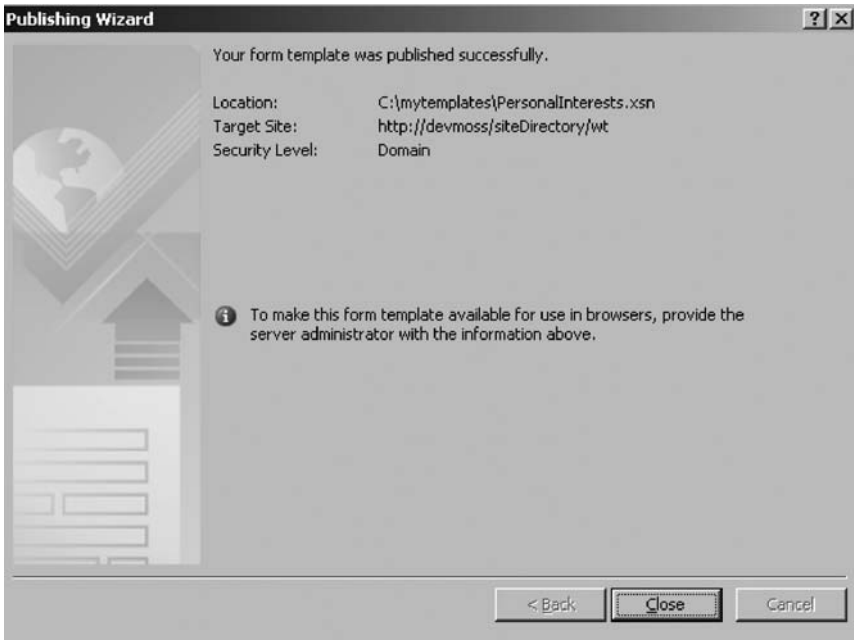


Figure 18.25 *Form template prepared successfully.*

18.6.2 Administrator-Deployed Form Templates

It is the administrator's job to thoroughly evaluate the form templates that he or she may ultimately deploy in the farm. Unless your organization requires *all* form templates to be approved and deployed by an administrator, forms requiring administrator approval generally require higher privileges and are therefore a greater security risk than user-deployed forms. For example, form templates that contain managed code require administrator approval before being deployed, as the contained code could be malicious code.

Once an administrator is certain that the form template is worthy of deployment and will not wreak havoc on the system, then he or she must upload the form template to the server and activate the form template in the required site collection.

Uploading form templates to the server

Form templates can be uploaded to the server via the Upload Form Template page in the browser interface or via a command line using stsadm.exe. The Upload Form Template page is accessible from the Application Man-

agement page within the central administration Web site (Figure 18.26). The administrator selects the form template prepared by the developer and then clicks the Verify button to ensure that there are no errors on the form and that the form will load properly. As an example—if, during the preparation phase, the developer forgot to set the template as a browser-compatible form template, then the error shown in Figure 18.27 will appear during the verification process. If the verification process passes, the administrator is free to proceed with the template upload. If this is the first time the template is being uploaded, there is no need to be concerned with the Upgrade section of the Upload Form Template page; the administrator can click the Upload button to upload the form template to the server. We will revisit the upgrade process shortly.

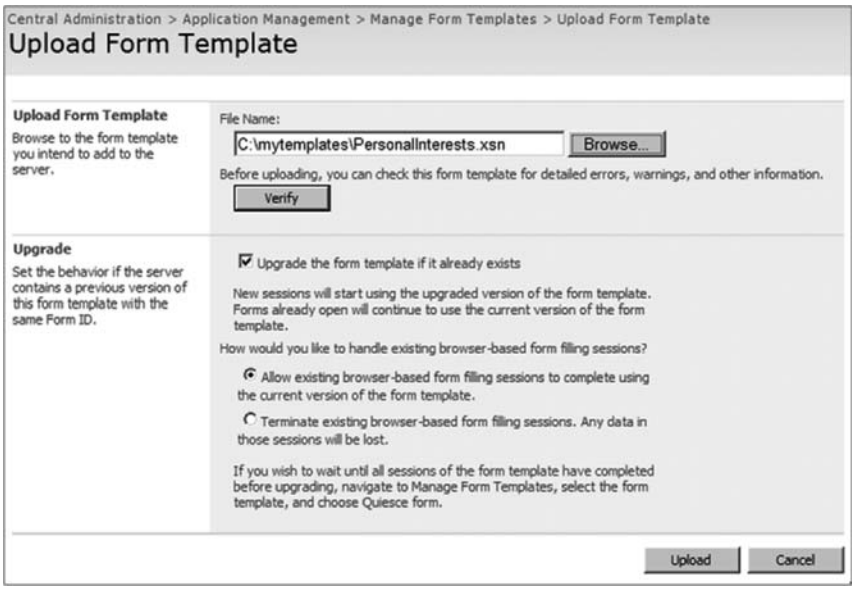


Figure 18.26 *Upload form template from central administration.*

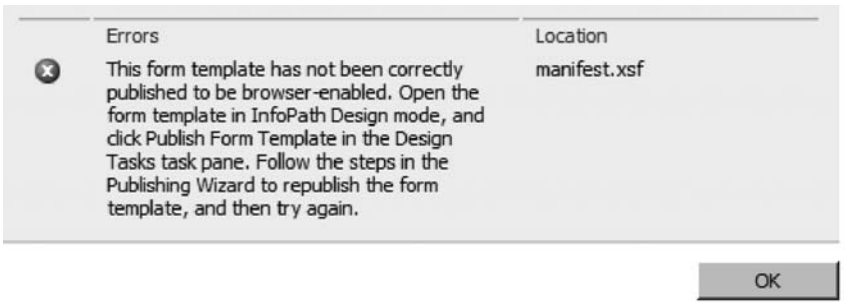


Figure 18.27 *Form upload verification error.*

The browser interface is very convenient for uploading and managing form templates on an individual basis, but when you have numerous form templates to upload and activate at one time, the interface becomes cumbersome. Fortunately, uploads can be performed through the command-line interface using `uploadformtemplate` operation in stsadm.exe. In addition, you can perform the task programmatically by leveraging the upload function in the Forms Services object model. In this chapter, we will focus on the stsadm.exe commands. However, a wealth of information on the Forms Services object model can be found on MSDN. The command to upload a form template is shown in Figure 18.28. If you have more than one form template to upload, you can create a simple batch file and upload all the templates at once.

```
stsadm.exe -o uploadformtemplate -filename c:\mytemplates\PersonalInterests.xsn
```

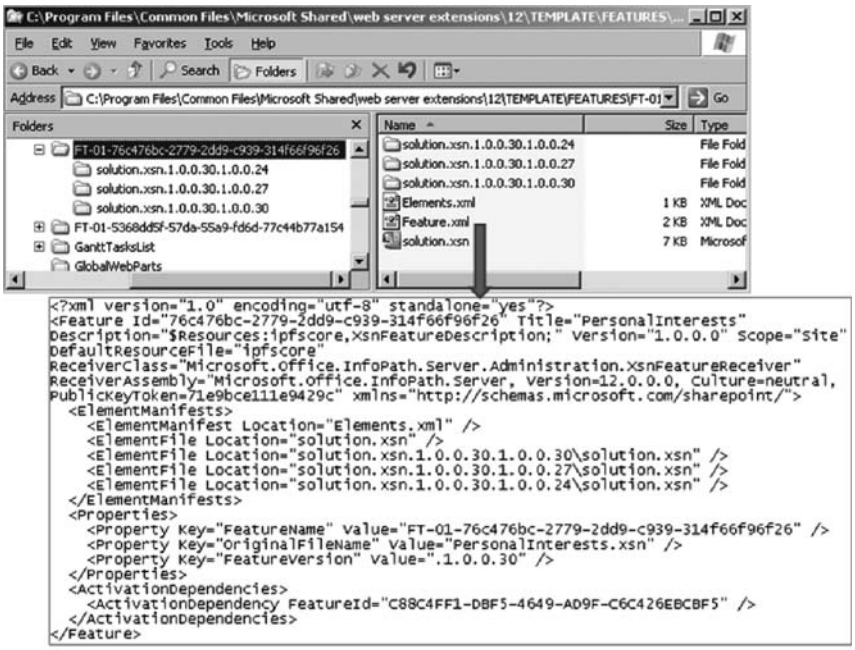
Figure 18.28
Uploading a form template using ststadm.exe.

Activating form templates

We mentioned earlier that user-deployed form templates are stored in the SharePoint content database; whereas administrator-deployed form templates are stored on the file system of the Web front-end servers. So where exactly is this location? When an administrator uploads a form template to the server, the template is implemented as a Feature, and as such is physically located in the Features directory; it will be one of the Feature folders with the name prefixed with "FT-01-" and followed by the Feature GUID. If you examine the Feature definition, you will notice that it is scoped at the site collection level (Figure 18.29) and references an element manifest that, upon Feature activation, provisions the form template into the FormServerTemplates library of the site collection (Figure 18.30). Understandably, the form template is inaccessible to users until it is activated within their site collection.

In addition to populating the site collection's Form Templates library, activating a form template also causes a corresponding site content type to be created (Figure 18.31). Users can then begin to leverage the form template by using the site content type or by referencing the form template directly. Deactivation of the form template results in the site content type being removed.

An administrator can activate a form template for a given site collection by using the Activate Form Template option on the drop-down menu associated with each form template listed in the Manage Form Templates page of the central administration Web site (Figure 18.32).



Feature.xml

Figure 18.29 *Form template feature.*

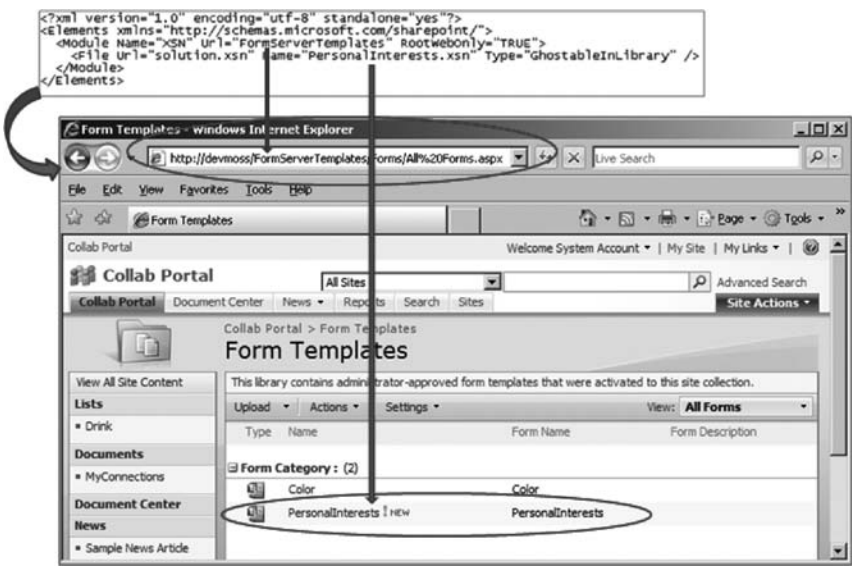


Figure 18.30 *Provisioning the form template to the FormServer Templates library.*

Once again, the browser interface is convenient for activating form templates on an individual basis, but when you have numerous form templates to activate at one time, the interface becomes cumbersome. Activations can

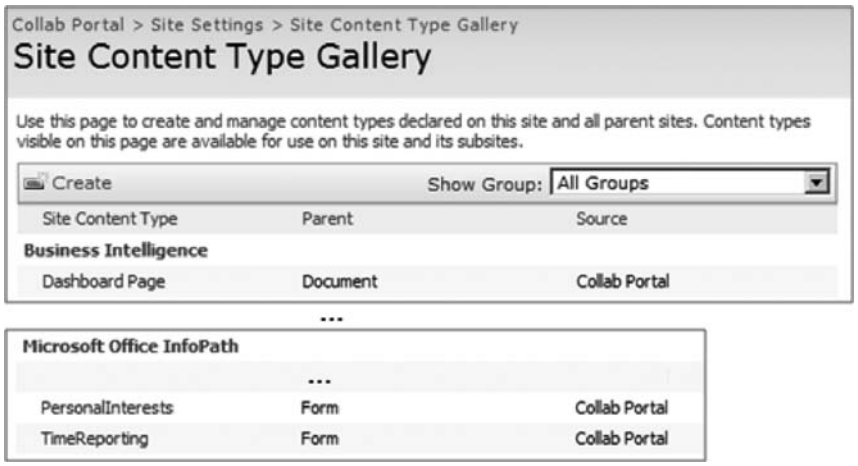


Figure 18.31
Site content type associated with form template.

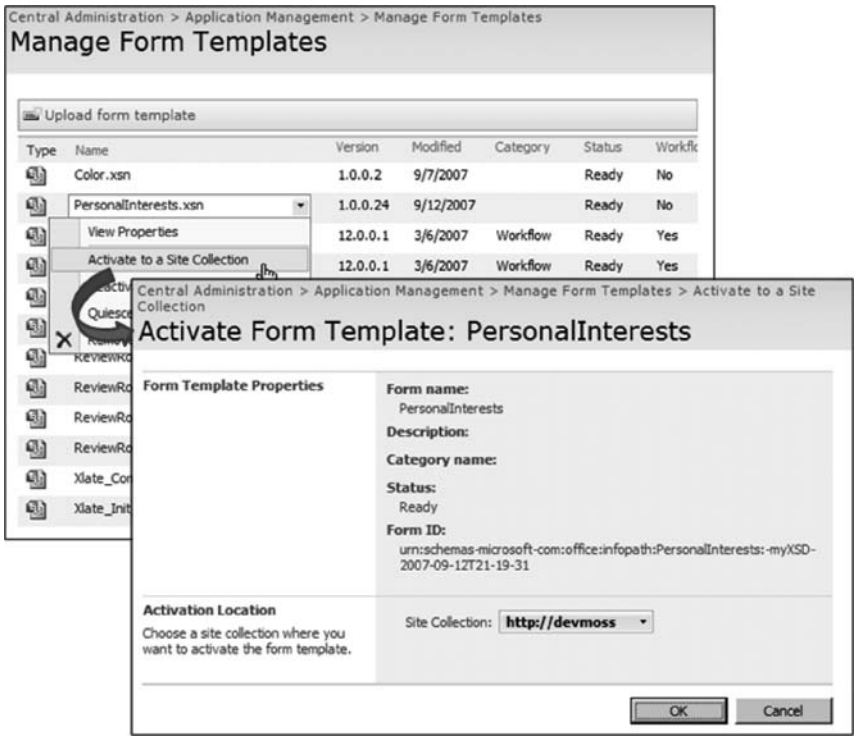


Figure 18.32
Activating a form template in a site collection.

be performed through the command-line interface using the `activate-formtemplate` operation in stsadm.exe. In addition, you can perform the task programmatically by leveraging the Activate function in the Forms Services object model. The command to activate a form template is shown in

```
stsadm.exe -o activateformtemplate -url http://devmoss -filename
  c:\mytemplates\PersonalInterests.xsn
```

Figure 18.33 *Activate a form template using stsadm.exe.*

Figure 18.33. If you have more than one form template to upload, you can create a simple batch file and upload all the templates at once.

Upgrading form templates

Notice on the form there is a special section specifically for handling form template upgrades. Upgrading a form template requires more thought and planning than merely clicking an Upload button. If users are currently using the older form template, then an administrator has several options with respect to how these existing sessions are handled. Administrators can choose to allow all existing browser-based form filling sessions to complete using the older version of the form template, or they can choose to terminate all existing browser-based sessions, resulting in loss of data. While the latter option may provide an expedited route to getting a form template upgraded, it would not be very popular among the end-user community. Figure 18.34 shows the generic error message that a user receives when they try to save their form data if the administrator has chosen to terminate sessions during an upgrade. The key point here is that even though the administrator has terminated the user's session, the user can still fill in data on the form; if the form is considerably large, the user will not be notified that the session has been terminated until they contact the server, whether it's by clicking Save or initiating a postback within the form. Imagine your frustration if you went to the trouble of filling in lots of data on a form, only to find that it's all been lost!

The recommended way of handling upgrades is to wait until the browser-based sessions are complete. This is achieved using a concept called quiescing.

Quiescing allows an administrator to disable a form template gradually, rather than abruptly. Disabling a form template gradually provides users with ample time to save their data and prepare for the upgrade of the template. Only administrator-deployed and activated form templates have quiescing capabilities. The Quiesce Form Template page (Figure 18.35) is accessible from the Quiesce Form Template option on the drop-down menu associated with each form template listed in the Manage Form Templates page in the Central Administration Web site. On the Quiesce Form Template page, the administrator enters the time interval (in minutes) in which the Form Template will be quiesced. Once the administrator clicks Start Quiescing, new form sessions based on that form template are no longer accepted, and the

Figure 18.34
Error message received by user upon form save.

countdown begins for the existing sessions. As a best practice, the administrator would inform all users (either via e-mail or some other notification mechanism) as to when the form will be disabled, giving them ample time to save their current changes. Once the allotted time has passed, the form template is quiesced (Figure 18.36); new sessions are still rejected, existing sessions are terminated, and any unsaved data is lost. When a form template has the quiesced status, it means that there are no open sessions using the template; therefore, the form template can be successfully upgraded without impacting users. To upgrade the form template, the administrator simply uploads the new form template. Since there are no open sessions, the administrator can choose either option from the upgrade section—both have the same effect. Once the new form template has been successfully uploaded, the administrator can release the form from its quiesced status by clicking the Reset Template button on the Quiesce Form Template page.

In addition to being able to quiesce form templates on an individual basis, you can quiesce an entire farm. Why would you want to do such a thing? Let's suppose that your administrator needed to take some servers down for scheduled maintenance. If your organization uses forms heavily and has numerous form templates deployed and activated throughout the entire farm, it is very cumbersome from an administrative perspective to manually, or even programmatically, track each form template and quiesce it before the scheduled downtime occurs. An easier option involves using the Quiesce Farm functionality, available from the Global Configuration settings on the Operation page in the central administration Web site (Figure 18.37).

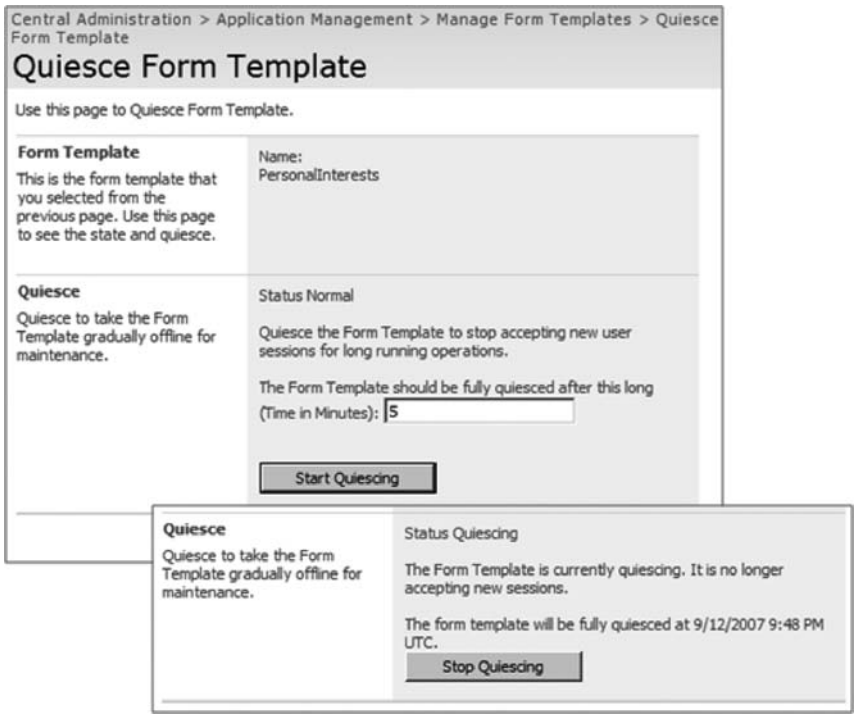


Figure 18.35 *Quiescing a form template.*

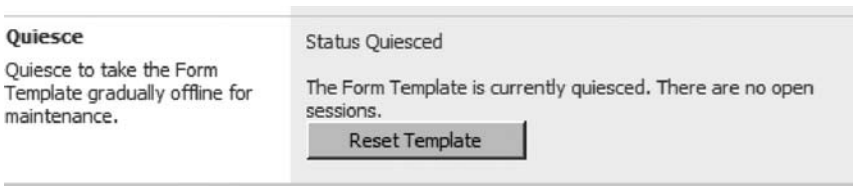


Figure 18.36 *Form template with quiesced status.*

The Quiesce Farm option behaves similarly to the Quiesce Form Template option, but instead of disabling a single form template, it disables all browser-based form templates within the entire farm, regardless of whether they have been deployed by a user or an administrator.

18.7 Document Information Panel

SharePoint Server 2007 and Windows SharePoint Services 3.0 are tightly integrated with the Office 2007 suite. We have seen throughout this chapter that when we deploy a form to a document library or a site content type, the data entered into the fields on the form are promoted to columns in the form library. This type of property promotion is extremely useful

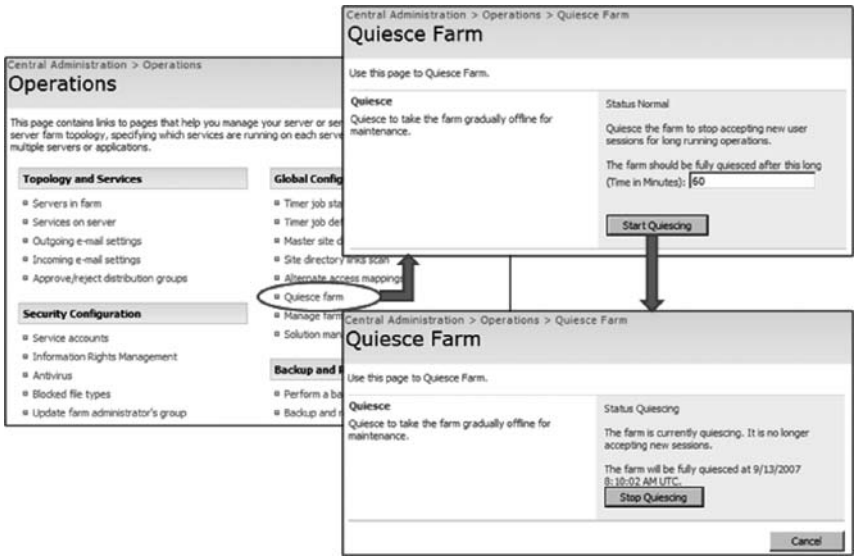


Figure 18.37
Quiesce the entire farm.

when working with Office applications in general, as a user does not have to open the complete document or form to reveal a specific property value. Office applications such as Microsoft Word, Excel and PowerPoint have supported property promotion into SharePoint lists and libraries since the days of SharePoint Portal Server 2001; however, unless the properties were one of the default document properties, such as Title, they were unwieldy to capture and to map to SharePoint. With Office 2007, Microsoft has provided a convenient interface for you to capture content for the custom columns in a SharePoint library that is consistent across all the Office applications. The interface we are referring to is the Document Information Panel, which is basically an InfoPath form embedded into your Office application and mapped to SharePoint content. The Document Information Panel replaces the Document Properties dialog used in earlier versions of Office. Figure 18.38 shows the default panel associated with a Word 2007 document. You can replace the default panel with your own custom Document Information Panel by associating the document with your InfoPath form template.

Replacing the standard Document Information Panel with a custom panel is very straightforward and involves creating a new custom template and associating it with the document site content type associated with the document library. When you navigate to a document library in SharePoint, click on the Document Library Settings from the Settings menu, and then click on the Document content type associated with the library, you are presented with the Document *list* content type settings (Figure 18.39). Because we are accessing the content type settings for the list, any changes that are

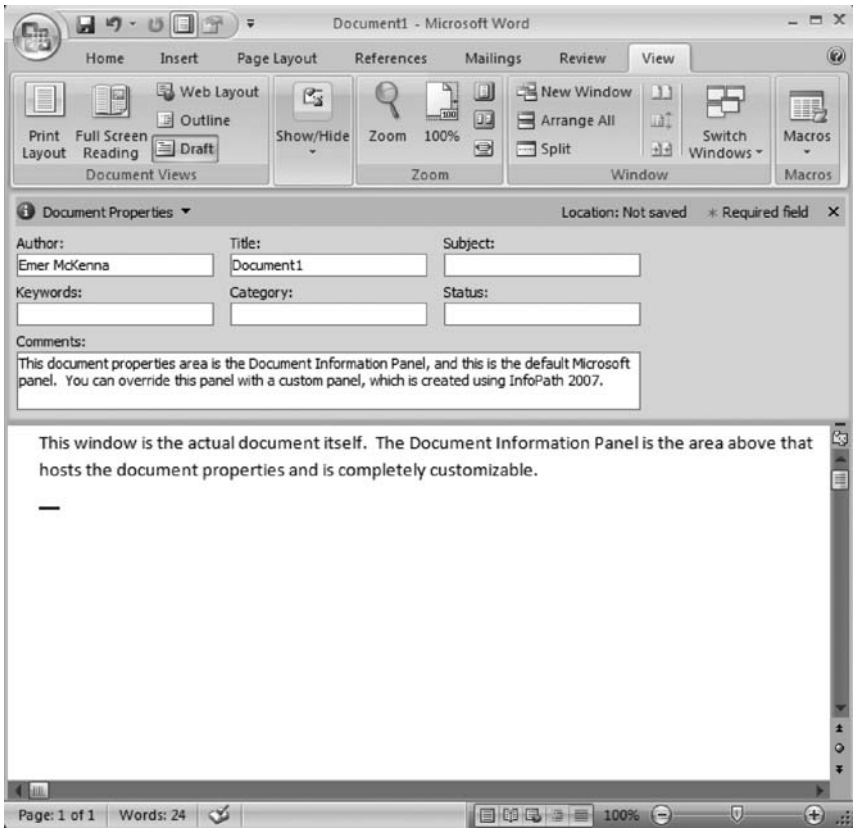


Figure 18.38
Default Word 2007 document information panel.

made apply only to list's version of the content type, the Document site content type remains intact.

On the Document list content type page, there is a link to the Document Information Panel settings associated with the content type. When you access the Document Information Panel Settings page, you will find a link that allows you to create a new custom template. Clicking on this link launches InfoPath 2007, and InfoPath creates a data source connection between the custom template and the library with which the content type is associated. InfoPath also creates some controls on the form template and binds them to columns in the SharePoint library. The form template is in designer mode at this point, and you are free to add and remove layouts, colors, controls, and bindings as needed (Figure 18.40). Once the template has been designed, it is deployed to SharePoint using the publishing wizard. When you initiate the publishing wizard during this process, you will notice an additional option at the bottom of the list of locations that allows you to deploy your form template as a Document Information Panel for a SharePoint site content type or list content type (Figure 18.41). That's really all

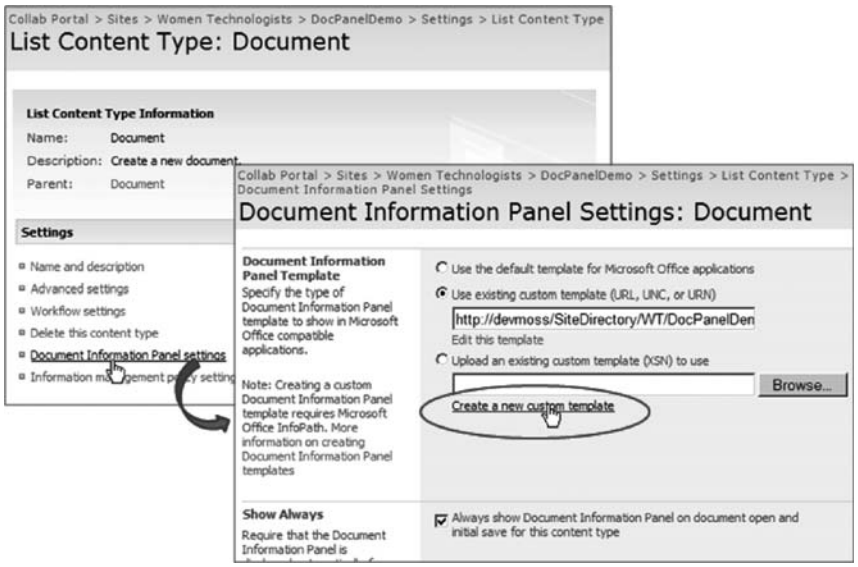


Figure 18.39 *Creating a new document information panel template.*

there is to it; clicking Next simply brings you to the verification dialog for one last check before you decide to deploy the template. Once the template has been deployed, documents created in that document library adopt the new Document Information Panel. Figure 18.42 shows the result of creating a new document in the sample document library to which we deployed the custom template. The same basic process is involved when you want to create a custom Document Information Panel for a site content type, as opposed to a list content type.

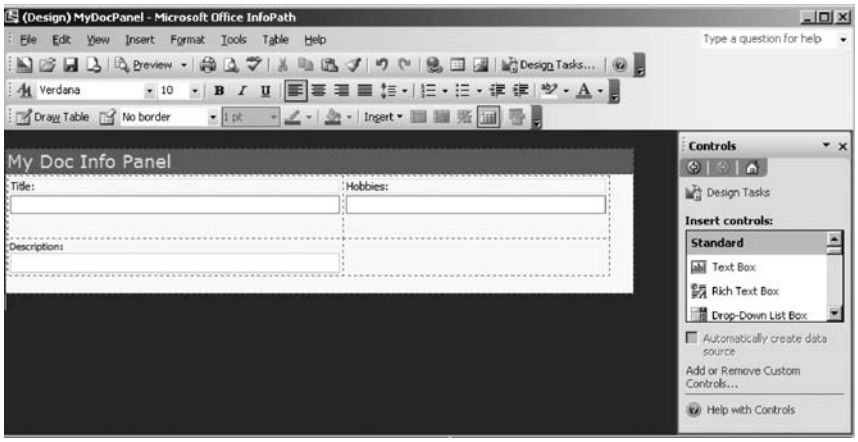


Figure 18.40 *Custom template designed in InfoPath 2007.*

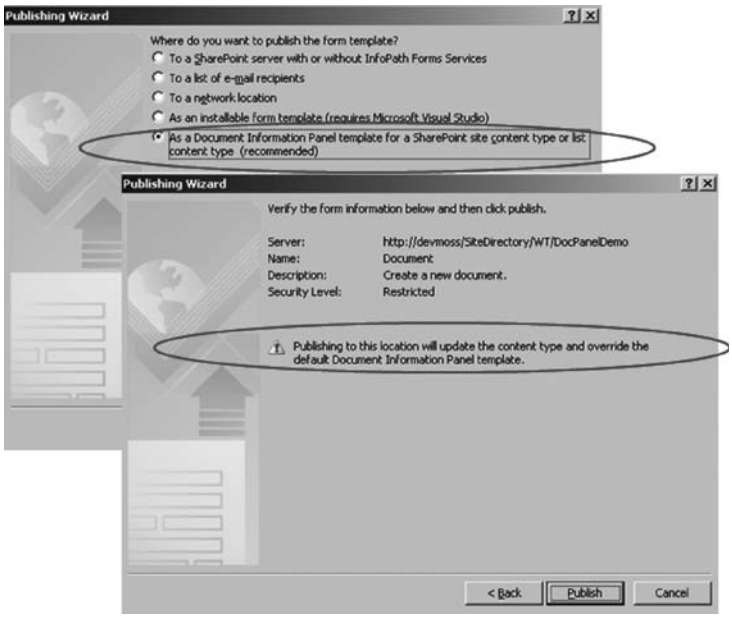


Figure 18.41 *Publishing the custom template.*

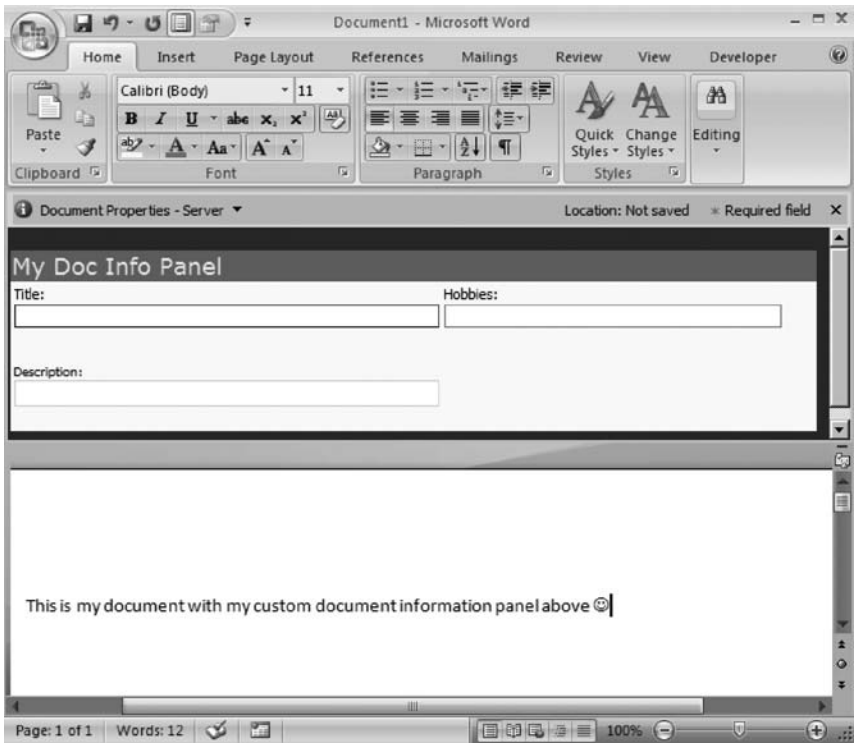


Figure 18.42 *Custom document information panel in action.*

18.8 Adding Code to a Form Template

InfoPath 2007 provides integration capabilities with Visual Studio to allow you to write managed code for your browser-enabled form templates. Visual Studio 2005 is the primary development environment when it comes to extending the SharePoint functionality; therefore, it makes sense that this becomes the standard development environment for adding code to InfoPath form templates. Visual Studio Tools for Applications (VSTA) is tightly integrated with InfoPath and allows you to add code to your form within the confines of the InfoPath client. VSTA is installed as an optional component as part of the InfoPath 2007 client installation. If you already have InfoPath 2007 installed and are unsure if VSTA is installed, you can check quickly by clicking the Programming option from the Tools menu and then clicking the Loading Event option. Either the VSTA environment will launch, or you will be presented with an error message indicating that VSTA is not currently installed. Figure 18.43 shows the VSTA environment presented when the user clicks the aforementioned Loading Event menu option. Notice that the code is set by default to Visual Basic. If you would prefer to code in C#, you can change the language preference from the Programming category on the Form Options dialog (Figure 18.44). This setting should be defined *prior* to launching VSTA for your form template.

The VSTA environment gives you access to the InfoPath object model, in addition to key development tools such as a debugger, IntelliSense, and the object browser. You can think of VSTA as bringing Visual Studio 2005 into the InfoPath 2007 client. It contains subset of the features that Visual Studio 2005 is equipped with, but at the same time provides the fully functional InfoPath client.

Another option for developing code for your form templates comes in the form of Visual Studio 2005 Tools for Office Second Edition (VSTO SE). You can think of VSTO SE as bringing the InfoPath 2007 client into Visual Studio 2005. It contains a subset of the features with which InfoPath 2007 is equipped but at the same time provides the fully functional Visual Studio 2005 environment. For example, if you want to preview your form, then you would still need to use the InfoPath client. Figure 18.45 shows an example of creating a new InfoPath form template with the Visual Studio 2005 integrated design environment. A major advantage of developing your form templates in this manner is that you are able to view multiple windows at one time—for example, you are able to view both the designer window and the code window. The experience is very similar to the SharePoint Designer 2007 development experience.

Let's take a quick look at how easy it is to attach code to your form template. In this example, we'll use VSTO SE to create a very simple form template that contains two textbox controls, txtName and txtHours. We will use

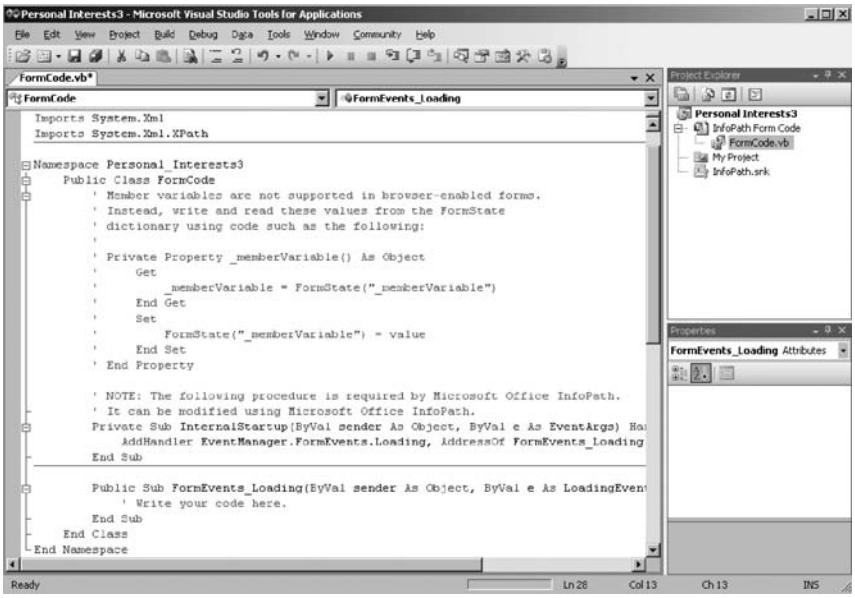


Figure 18.43 *Visual Studio for Tools environment.*

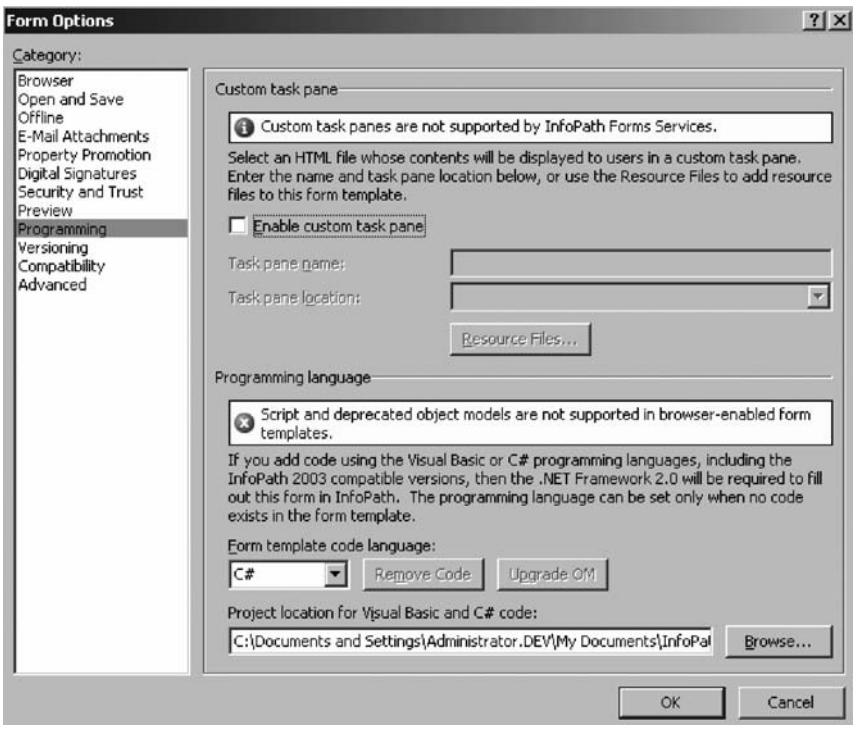


Figure 18.44 *Changing the programming language setting.*

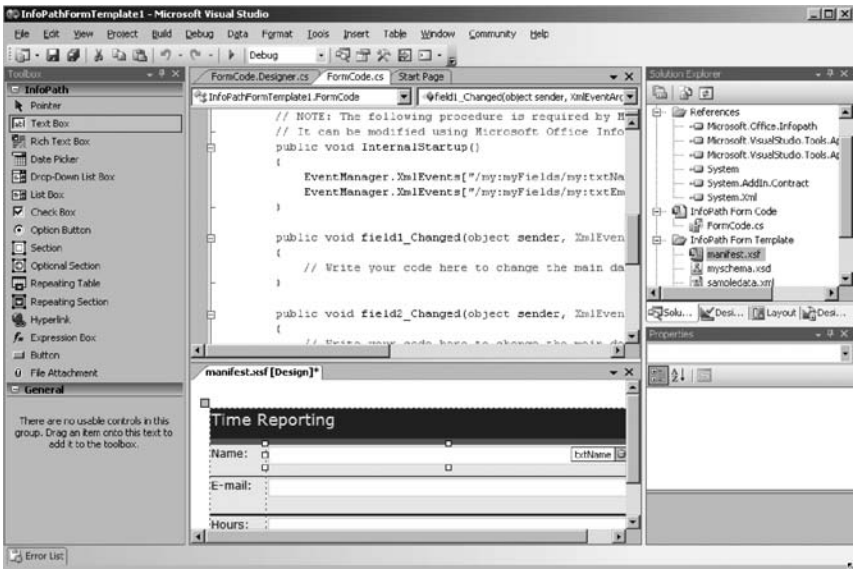


Figure 18.45
Visual Studio 2005 Tools for Office second edition.

code to automatically populate the txtName control with the name of the currently logged on user; that will be the extent of our demonstration. The purpose of this exercise is to walk you through using the VSTO SE development environment and show you what additional steps you need to take when deploying your form template. For example, because your form template contains associated code, the security of the form needs to be set to Full Trust. Full Trust requires the form template to be deployed by an administrator and also requires that the form be digitally signed with a certificate. We'll walk you through this process.

If you have not already done so, now would be a good time to install VSTO SE. VSTO SE can be downloaded from the Microsoft Web site, and is installed by simply double-clicking the setup file on the system in which you have both Office 2007 and Visual Studio 2005 installed. VSTO SE needs to be installed after the InfoPath 2007 client; otherwise, Visual Studio will not be able to present all the InfoPath 2007 project templates. Another factor to keep in mind when installing VSTO SE is that if you have two versions of Office running side by side on the same machine upon which you wish to install VSTO SE, the installation will fail. To resolve this issue, you must remove the earlier version of Office (for example, Office 2003) prior to installing VSTO SE.

Step 1: Attaching code to a form template

1. Launch Visual Studio 2005

2. From the File menu click New, and then click Project. The New Project dialog appears.
3. Expand the Visual C# project type, and select Office (Figure 18.46). Note, do not navigate down to the 2007 Add-Ins project type, as the InfoPath form template is not located in that area.
4. From the Templates window, select InfoPath Form Template (Figure 18.46).
5. Enter the remainder Visual Studio project information.
6. Click OK. The InfoPath Design a Form Template dialog appears.
7. Select the options to design a new form template based on a blank template and enable the browser-compatible features.
8. Click OK. You are presented with the design view of the manifest.xsf in the main development window. The right task pane shows the Layout task, and the left task pane shows the toolbox listing all the available controls. Use these tools to create a form that looks similar to that shown in Figure 18.47.
9. Rename the properties of the text fields as txtName and txtHours, respectively. The txtHours text box field should be of type Whole Number (integer).
10. From the Visual Studio View menu, click Solution Explorer.
11. From the Solution Explorer window, double-click the file named FormCode.cs.
12. The code should appear in the main development window.
13. Enter the code shown in Figure 18.48.
14. Save the project.

The code used in this example retrieves the username from the System.Environment.Username property and binds this data to the field using the XPathNavigator object.

Step2: Setting the security and digitally signing the form template

Since the form template has code behind, we need to digitally sign the template with a certificate. The certificate ensures that the form template is executed with the appropriate permissions on the SharePoint server and ensures the integrity of the form code during processing. It should be noted that the form template will successfully deploy *without* the digital signature. However, when a user tries to create or open a form instantiated from the form

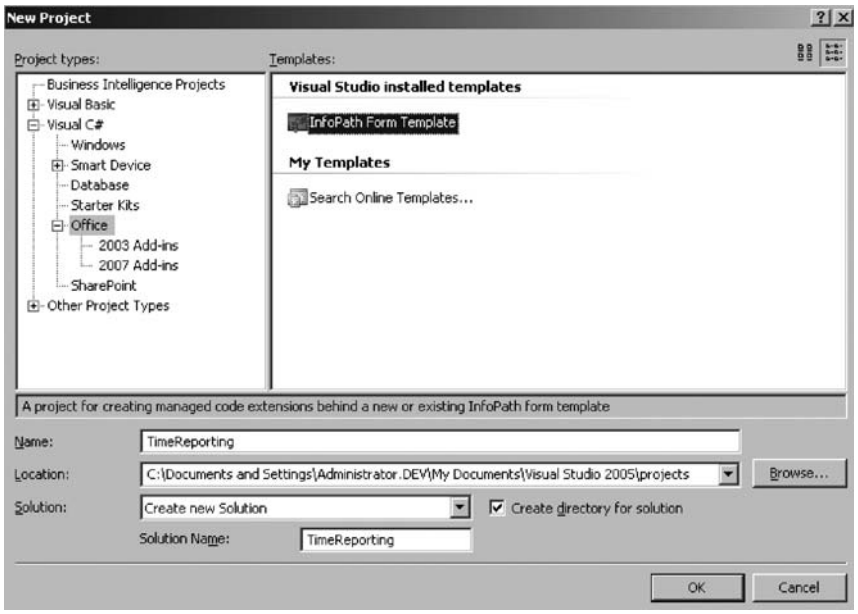


Figure 18.46 *Creating a new InfoPath form template from Visual Studio 2005.*

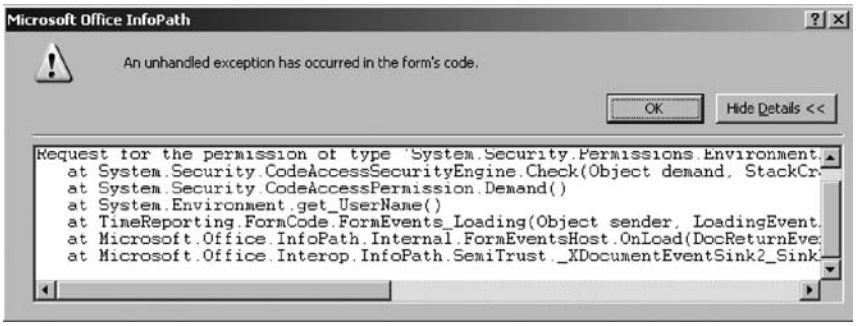
Figure 18.47 *Time reporting form.*

template, an error similar to that shown in Figure 18.49 appears; this is true whether the form is opened using InfoPath or the browser. Digitally signing your form template requires a code-signing certificate from a trusted certificate authority (CA). If your organization is its own CA, as is the case for organizations that have deployed certificate services on Windows Servers running in their Active Directory environment, then you can request a certificate internally and use it to deploy your form template. Many organizations prefer to get their certificates from an external trusted CA, such as those provided by Verisign or Operational Research Consultants (ORC). If your organization leverages an external CA, you can use their certificates when deploying your form template. When you are developing and testing your form template, you can create your own temporary certificate to digitally sign your form template. You can create this certificate directly from the InfoPath interface. For the purposes of our demonstration, we will create our own certificate, but it should be noted that this option should not be used when deploying form templates to a production environment.

Figure 18.48
Form code.

```
using Microsoft.Office.InfoPath;
using System;
using System.Xml;
using System.Xml.XPath;

namespace TimeReporting
{
    public partial class FormCode
    {
        public void InternalStartup()
        {
            EventManager.FormEvents.Loading +=
                new LoadingEventHandler(FormEvents_Loading);
        }

        public void FormEvents_Loading(object sender, LoadingEventArgs e)
        {
            string strUserName = System.Environment.UserName;
            string xpathFieldName = "/my:myFields/my:txtName";
            XPathNavigator usrnameField =
                MainDataSource.CreateNavigator().SelectSingleNode(xpathFieldName,
                NamespaceManager);
            usrnameField.SetValue(strUserName);
        }
    }
}
```

Figure 18.49
Unsigned fully trusted form template.



1. From the Tools menu, click Form Options.
2. Click the Security and Trust category.
3. Set the Security level for the form to Full Trust. Note that you will need to uncheck the Automatically determine security level setting to enable the trust setting.
4. From the Form Template Signature section, set the Sign this form template to checked.
5. Click Create Certificate.
6. A message box appears; click OK. A self-signed certificate has been created (Figure 18.50). Now we need to export and deploy

the certificate on the SharePoint server by logging on locally to the box.

7. Click View Certificate. The Certificate dialog appears.
8. Click the Details tab, and then click Copy to File…. The Certificate Export Wizard appears.
9. Click Next.
10. Click Next again. We will not export the private key.
11. Select the DER encoded binary X.509 (.CER) as the certificate format.
12. Click Next.
13. Enter the file name and location of the certificate that you want to export; export it to a location that you can access from the SharePoint 2007 server to which you will import the certificate—for example, c:\mytemplates\mycert.cer.
14. Click Next.
15. Click Finish.
16. You will receive a message stating that the export was successful. Click OK.
17. Click OK again to close the Certificate dialog. Now we need to import the certificate to the SharePoint 2007 server.
18. Log on locally to the SharePoint 2007 Server.
19. Launch Internet Explorer and bring up the Internet Options dialog.
20. Click the Content tab. We are using IE 7.0, so the area where the certificates are managed may differ depending on the version of your browser.
21. From the Certificates section, click Certificates. The Certificates dialog appears.
22. Click the Trusted Root Certification Authorities tab.
23. Click Import. The Certificate Import Wizard appears.
24. Click Next.
25. In the File Name field, select the certificate that you stored previously in step 13.
26. Click Next.
27. Click Next again to place the certificate in the Trusted Root Certification Authorities store.

28. Click Finish. A Security Warning message box appears asking if you really want to install the certificate.
29. Click Yes to install the certificate.
30. Close all the dialogs and log off the SharePoint server.
31. Go back to VSTO.

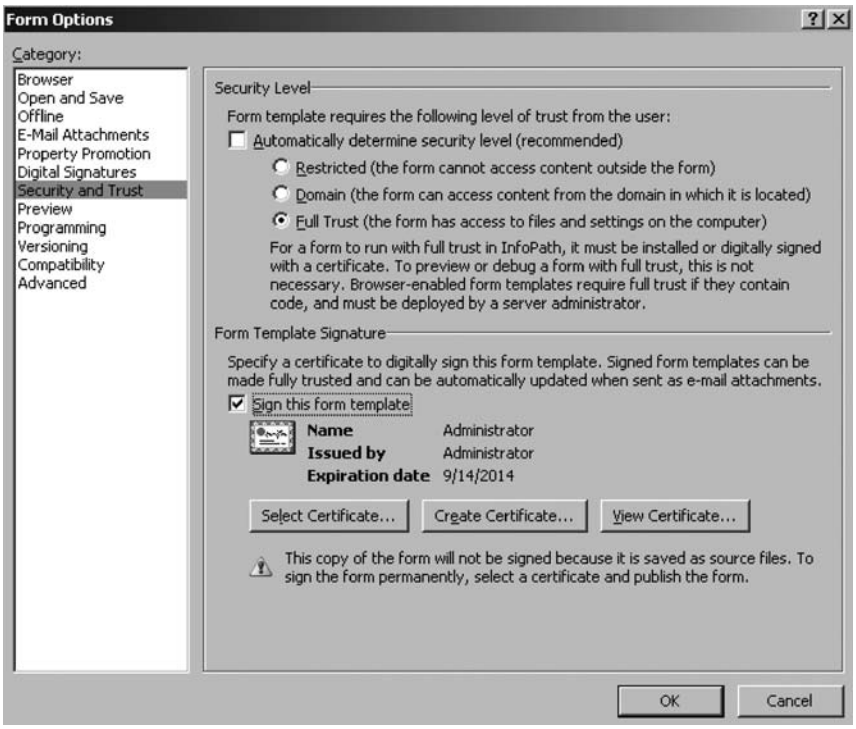


Figure 18.50
Digitally signing a form template.

Step3: Build and deploy the form template

All that remains is to build and deploy the form template. Remember: because the form template contains code, we need to prepare it for later deployment by an administrator.

1. From the Build menu, click Build TimeReporting. Replace the word TimeReporting with the name of your project. This builds the assembly (TimeReporting.dll) that contains the code associated with your form template. During deployment, this .dll file is packaged and stored in the form template's Feature folder.

2. To deploy the template, follow the instructions listed in the earlier section on how to prepare a form template for deployment by an administrator. Then inform your administrator of the template location and ask them to upload the template to SharePoint. If you are working in a development and have the necessary permissions, you can upload the template by following the information provided in the sections that discuss uploading and activating form templates.

A site content type should now be available for you to test your form template's functionality. Remember, the code used in this example was purely for demonstration purposes only and does not contain sufficient code for a production environment. For example, when a user creates a form based on this form template, the Name field is populated by the username with which they are logged onto their computer. If another user were to subsequently edit this form, their name would replace the original author's name—which is obviously not good from a design perspective. Additionally, it would be better if the username data was retrieved from the currently logged-on SharePoint user, as this often times differs from the username used to authenticate the workstation to the domain. Now that you know how to associate code with your form template, you can implement your own code.

18.9 Workflow and SharePoint Server 2007

Before we jump into SharePoint's workflow capabilities, we should spend just a minute reviewing what exactly we mean by the term "workflow." Workflow typically refers to the flow of information that comprises a business process. This flow includes the interactions, activities, and events that occur during the process lifecycle. At a very high level, workflow can be divided into two camps: system workflow and human workflow. System workflow represents business processes that do not involve people, whereas human workflow represents business processes in which people play a key role in the flow of information and in the progress of the overall process. Products such as BizTalk server target system workflows, whereas SharePoint Server 2007, a product designed for human interaction, targets the realm of human workflow. Since this book focuses on SharePoint, from this point on, any time we refer to workflow, we are specifically referring to *human* workflow.

Workflow automation refers to the software or technologies used to model business processes. That is a pretty broad definition, however, and different types of software are better suited for different types of workflow. So let's dissect this a little more:

- *Ad-hoc, or unstructured.* Least complex; this type of workflow automation includes the office routing slip or other such features built into your desktop software that you can use without any special coding or development.
- *Team-process automation.* Slightly more complex; this more structured workflow automation applies across a team of individuals and models team-based business processes, such as the routing of information or the review and approval of documents.
- *Line*-of-business. Still larger in complexity and scale; such applications include an insurance company's claims processing software or a manufacturing company's order production tracking system. The defining characteristic of this type of workflow automation system is that the company's business runs on or depends heavily on it.
- *Extranets, external communications.* Most complex and largest in scale; this type of workflow automation has become more and more common over the last ten years, as companies rely on Internet technologies to perform such tasks as communicating with their suppliers, for example. This type of workflow crosses company boundaries, and is often found in the e-business space.

The relationship between workflow and SharePoint stems back to the original product release of SharePoint Portal Server (SPS) 2001. SPS 2001 shipped with a workflow engine that provided out-of-the-box approval routing workflow capabilities. The design goals for the SPS 2001 workflow engine focused directly on team-process automation. From an extensibility perspective, the workflow architecture provided a workflow designer and an object model specifically designed for handling workflows. This allowed developers to customize and extend the out-of-the-box workflow capabilities to meet the needs of their organization. With the introduction of SharePoint Portal Server 2003 and Windows SharePoint Services 2.0, workflow took a brief hiatus from the SharePoint world. SPS 2003 provided an event architecture through which you could attach a single event handler to a document library. The event handler could trap asynchronous events that occurred in the document library, such as when an item was saved to the library; it could then process the document, effectively creating a workflow-type scenario. However, to call it workflow was a stretch! To have workflow support in SPS 2003, you either had to build it yourself—an expensive option for most organizations—purchase another Microsoft product, such as BizTalk Server, or purchase a third-party workflow product, such as that offered by K2. With the advent of SharePoint Server 2007 and Windows SharePoint Services 3.0, workflow has made a magnificent comeback and is deeply ingrained in the product. In fact, since SPS 2003, Microsoft's overall workflow strategy has become more streamlined and is now a key component of the base Windows

operating system. With this new strategy, the workflow capabilities of SharePoint Server 2007 are ideally suited to many types of workflow, including team-process automation, line-of-business, and e-business scenarios.

18.9.1 Windows Workflow Foundation

In addition to ASP.NET 2.0, WSS 3.0 and SharePoint Server 2007 are built upon Windows Workflow Foundation (WWF) and Windows Communication Foundation (WCF). WWF is Microsoft's workflow platform, which offers a workflow runtime engine that can be hosted in any Windows process along with a common set of tools and programming interfaces that developers can leverage when building their workflow applications. WCF is a set of technologies built around the Web services architecture that allow you to build and deploy connected systems. To install Windows SharePoint Services 3.0 and SharePoint Server 2007, you must first install WWF and WCF. WWF and WCF are both included natively in the .NET Framework 3.0 and in the Windows Vista operating system. Windows Server 2008 also provides native support for WCF and WWF. WWF includes a library filled with a set of building blocks that represent common workflow activities. This library is referred to as an activity library, and a workflow activity represents the building block or discrete unit that can be leveraged by developers. For example, the `TerminateActivity` activity provided by WWF allows you to immediately end the operation of your workflow. You can build your own custom activities to add to the activity library. WWF provides support for both system workflows and human workflows; therefore, from a developer perspective, you can write code to support either scenario. WWF includes a set of developer tools that integrate with the Visual Studio IDE to offer developers a familiar environment with which to automate their business processes. The tools come in the form of Visual Studio Extensions 2005 for .NET Framework 3.0 (Windows Workflow Foundation), which can be downloaded from the Microsoft Web site.

18.9.2 Workflow in Windows SharePoint Services

Windows SharePoint Services 3.0 leverages the WWF's capabilities by hosting the WWF runtime engine and modifies some of the runtime services to better suit the needs of SharePoint. Windows SharePoint Services 3.0 also extends the activity library, providing additional activities such as `CreateTask` and `SendEmail`. The `CreateTask` activity creates a Windows SharePoint Services task for your workflow, the properties of which can be populated through code. SharePoint uses tasks as the mechanism to track workflow actions with the people assigned those actions. Windows SharePoint Services maintains a task list in each SharePoint site that is dedicated to maintaining workflow-associated task information, and developers can

use the `CreateTask` activity to add tasks to this list. The `SendEmail` activity creates and sends an e-mail to specified users. You can review the complete set of workflow activities provided by Windows SharePoint Services 3.0 by using the object browser in Visual Studio 2005 to examine the `Microsoft.SharePoint.WorkflowActions` namespace (Figure 18.51). These activities are leveraged by the predefined workflow templates provided by SharePoint Server 2007, which we'll revisit later. When you develop workflows for Windows SharePoint Services, you basically create your own custom workflow template, install the template on the SharePoint server, and then associate the template with a specific document library, list, or content type. Once this association is in place, a user can then create an instance of the workflow that loads and executes the template in the WF runtime engine hosted by Windows SharePoint Services. This concept is quite similar to the way InfoPath forms work—you build the form template; deploy the form template; and associate it with a document library, list, or content type. A user then creates an instance of your form, which activates the code and logic in the form template.

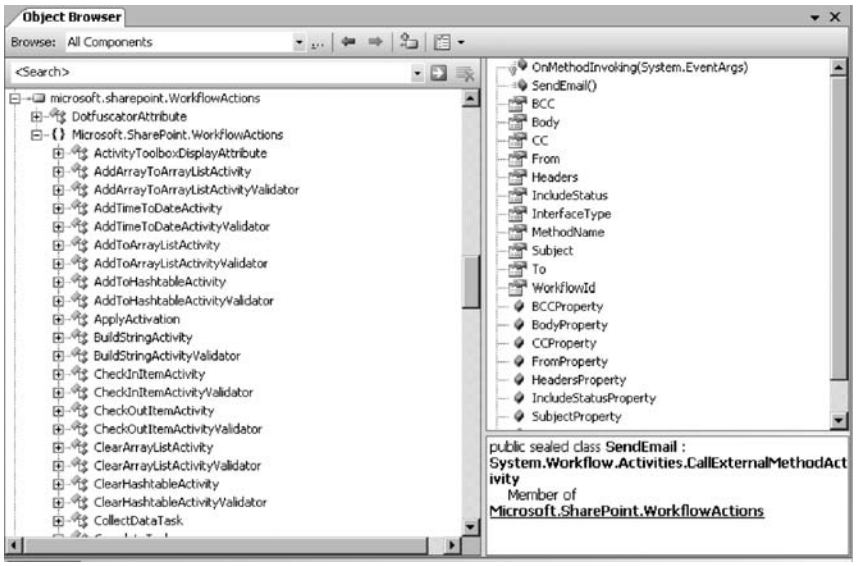


Figure 18.51 *Windows SharePoint Services 3.0 workflow activities.*

Windows SharePoint Services comes with one predefined workflow out of the box: the three-state workflow. As its name implies, this workflow manages an item in SharePoint as it passes through three states: an initial state, an intermediate state, and a final state. Between each state, a specific action can occur. For example, you could have three states: Active, Awaiting Approval, and Complete. When the workflow starts, it is in the Active state, and a participant in the process is assigned a task indicating that they have to create a

budget proposal. When the participant has finished his or her draft of the budget proposal, they mark the task as complete, and the workflow moves on to the next state: Awaiting Approval. At this point, another task is created; another participant is assigned the role of reviewer and sent a request to review the budget proposal. The reviewer decides that the budget is acceptable and approves the item, which changes the status of the item in the workflow. That causes the workflow to move to the final Complete state, at which point the workflow is terminated. The three-state workflow was designed primarily to be used with the Issue Tracking list; however, it can be configured to work with any list that implements a Choice column containing at least three values. The Choice column is used to represents the states that the workflow tracks, so while you can have more than three choices defined, only three can actually be implemented as states in the three-state workflow.

SharePoint workflow templates in general are deployed as Features. If you examine the site collection Features, you will see the three-state workflow template listed. When you activate this Feature, the three-state workflow becomes an option when associating workflow with a list, document library, or content type (Figure 18.52). Figure 18.53 illustrates the choice list associated with the workflow and the configuration of the initial workflow state requesting that workflow participant Rita Smith create a budget proposal. The Choice field selected was the Category column that comes with the Issue Tracking list; this column was populated with the fields that we wanted to be represented as states in the workflow.

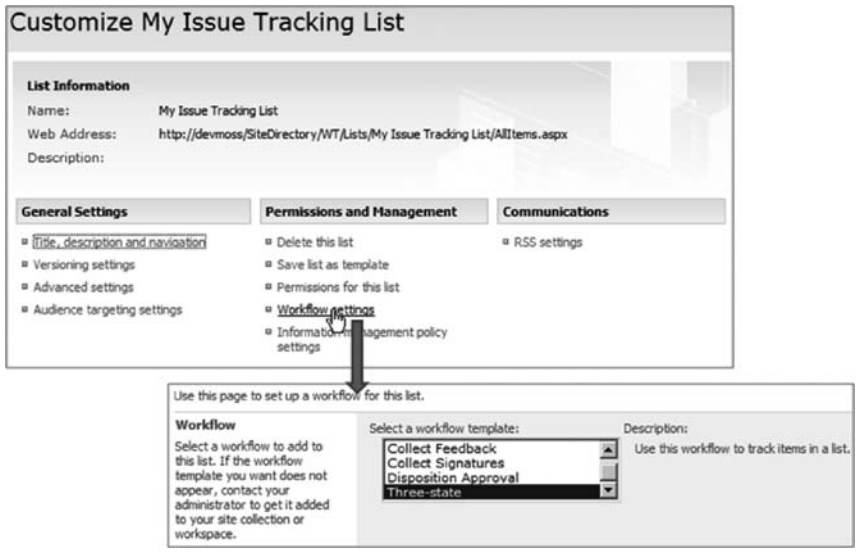


Figure 18.52 *Three-state workflow association.*

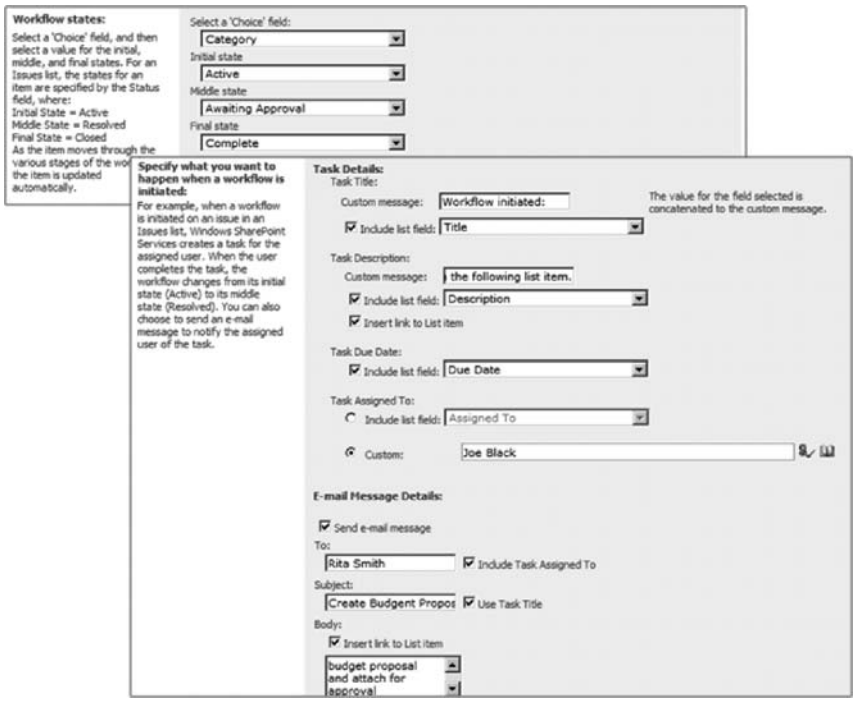


Figure 18.53
Configuring three-state workflow.

As previously mentioned, workflows in SharePoint are implemented as Features. If you navigate to the Features folder, you will notice a folder named IssueTrackingWorkflow (Figure 18.54). The Feature defines the workflow, including the name and description of the Feature to appear in the Features list for activation. Even though the name of the Feature folder in this instance is IssueTrackingWorkflow, the actual name of the Feature that appears through the browser interface is Three-State Workflow. The Feature also defines the assembly and the class that represents the workflow. Workflow assemblies must be strongly named and installed in the global assembly cache. Upon activation, the workflow appears as a selection in the drop-down list when associating a workflow with a document library, a list, or a content type; deactivation of the Feature removes the workflow option from the same drop-down list. If you deploy a workflow, whether it is your own custom-built workflow or that of a third party, and you do not see the workflow as an option in the association drop-down list, you should check to make sure that the Feature for that specific workflow has been activated.

As a developer, when you create workflows in Windows SharePoint Services you are limited to the use of ASPX forms. InfoPath Forms are not supported. Because the workflow is limited to the use of ASPX forms, users must use their Web browsers to interact with the workflow. And aside from sending e-mail messages, there are no integration capabilities between

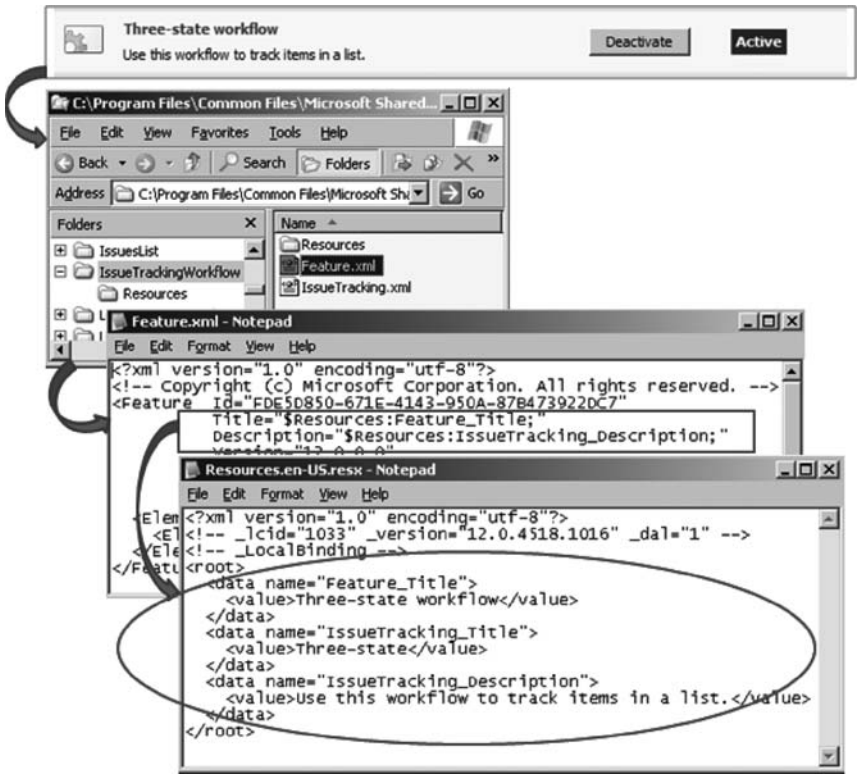


Figure 18.54 *Feature fepresentation of three-state workflow.*

Windows SharePoint Services 3.0–based workflows and Office 2007 client applications, such as Microsoft Office Word 2007. SharePoint Server 2007, on the other hand, supports the use of InfoPath forms in workflows. The support of InfoPath forms means that workflows can be easily integrated with Office 2007 client applications through the use of the Document Information Panel.

18.9.3 Workflow in SharePoint Server 2007

SharePoint Server 2007 provides a set of predefined workflows that can be easily customized to meet the needs of your organization. Table 18.3 lists the predefined workflows and their associated features. All of the listed features are scoped to the site collection, and both the feature name that is displayed in the browser and the associated feature folder name are listed in the table. If you examine the features folder, you will notice that there are a few other folders that have the word workflow in their names. One such folder is the BulkWorkflow folder, a hidden feature that is scoped at the farm level and adds a custom action to the standard menu allowing workflows to be completed in bulk. What does this mean, and why is it useful? From a functional

perspective, this means that as an end user, you will see an additional menu item on the Actions menu, called Process all tasks. If you are a participant in multiple workflows within your organization, the benefits of this capability are quickly realized. When you click the Process all tasks menu option, you are presented with a collated list of all the workflow task types, sorted by workflow template. If you click on a specific task type, you are presented with an editable form, implemented through InfoPath, that allows you to edit the tasks in bulk. Only the workflows that support bulk edits are displayed in the list. Windows SharePoint Services 3.0 does not support the capability to process workflows in bulk.

The predefined workflows for SharePoint Server 2007 leverage InfoPath form templates. If you examine the Feature folders of each of these workflows, you will notice a folder named Forms that hosts the InfoPath forms associated with the workflow. Before we show you how to integrate InfoPath forms with workflow, let's take a quick look at how workflows are authored in SharePoint.

18.10 Workflow Authoring

When it comes to authoring workflows, there are a variety of tools provided for Windows SharePoint Services and SharePoint Server 2007. Which tool to use and when really depends on what you are trying to achieve with your workflow. For example, if you want to create a custom workflow that leverages the use of InfoPath forms, you would choose SharePoint Server 2007

Table 18.3 *SharePoint Server 2007 Predefined Workflows*

Workflow Name	Description	Feature Name in Browser
Approval	Routes a document for approval. Approvers can approve or reject the document, reassign the approval task, or request changes to the document. By default, the workflow assigns tasks to participants sequentially.	Routing Workflows (Folder Name = ReviewWorkflows)
Collect Feedback	Routes a document for review. Reviewers can provide feedback, which is compiled and sent to the document owner when the workflow is completed. By default, this workflow assigns tasks to participants in parallel.	Routing Workflows (Folder Name = ReviewWorkflows)

Table 18.3 *SharePoint Server 2007 Predefined Workflows (continued)*

Workflow Name	Description	Feature Name in Browser
Collect Signatures	Gathers signatures needed to complete a Microsoft Office document. This workflow can only be initiated from within an Office application.	Collect Signatures Workflow (Folder Name = SignaturesWorkflow)
Disposition Approval	Manages document expiration and retention by allowing participants to decide whether to retain or to expire documents.	Disposition Approval Workflow (Folder Name = ExpirationWorflow)
Translation Management	Manages document translation by creating copies of the documents to be translated and assigning translation tasks to the translators.	Translation Management Workflow (Folder Name = TranslationWorkflow)

and Visual Studio 2005 with the WWF extensions and the SharePoint SDK templates applied. If you want to build a workflow application that uses ASPX forms only, then you would choose Windows SharePoint Services and either SharePoint Designer 2007 or Visual Studio 2005 with the WWF extensions applied.

18.10.1 Authoring Workflows in Windows SharePoint Services 3.0

Two options are available for authoring workflows targeted for the Windows SharePoint Services 3.0 platform. The first option involves the use of Visual Studio 2005 with the WWF extensions applied. The Visual Studio 2005 Extensions for .NET Framework 3.0 (Windows Workflow Foundation) extends the native Visual Studio environment to provide support to developers for building workflow applications that leverage Windows Workflow Foundation. This is great news for developers already familiar with developing applications in Visual Studio 2005, as it offers a familiar environment from which to build workflow applications. Once the workflow extensions have been installed, an additional Workflow option appears under both the Visual Basic and C# project types in the New Project dialog (Figure 18.55) The Workflow option offers a number of workflow project templates that you can use as the foundation for your custom workflow solution. The project templates include workflow libraries used for creating sequential workflows, state machine workflow libraries used for creating state machine workflows, and activity libraries used for creating a set of activities that can be reused in future workflow solutions.

The workflow extensions include a Workflow Designer (Figure 18.56), from which you can drag and drop components directly onto the designer page, ultimately generating some code behind the scenes. The first icon in the designer window represents the start of the workflow, and the last icon represents the end of the workflow. The content between these two icons represents the workflow activities and actions to be carried out, and the arrows indicate the business process flow. For the development of a workflow, the first task involves adding the activities that you will need. With the Workflow Designer, this is a simple matter of dragging and dropping the activity from the Toolbox onto the designer window. When developing SharePoint workflows, the `onWorkflowActivated` activity must be the first activity in the workflow, as it acts as a gateway to SharePoint. It is this activity that allows an association to a list or document library to be configured through the browser.

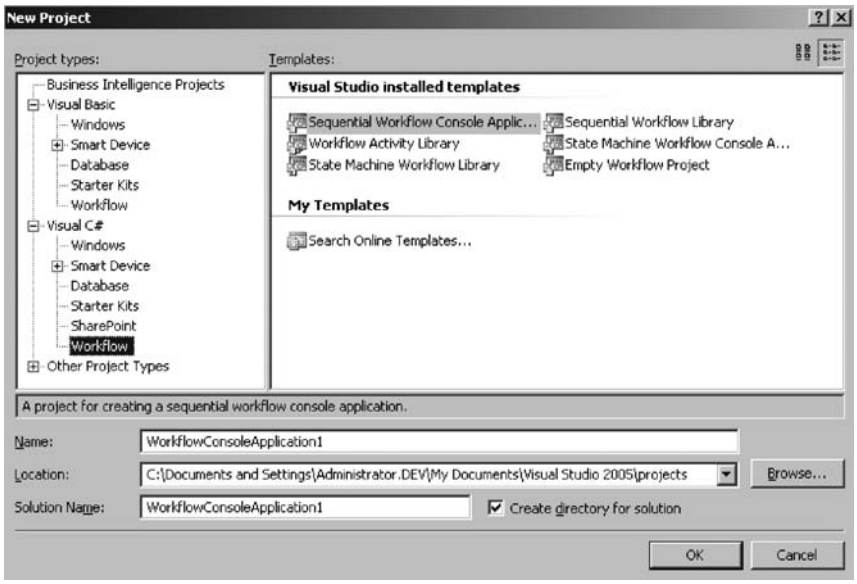


Figure 18.55 *Workflow project templates added to Visual Studio.*

To access the activities provided by Windows SharePoint Services, you must first add a reference to the Microsoft.SharePoint.WorkflowActions.dll assembly, then right-click the Toolbox and select the Choose Items menu option. The Choose Toolbox Items dialog appears, and from it, you can select all the activities related to the assembly (Figure 18.57). Numerous activities are available for use, many of which are used to manage the task information in the workflow task list. Examples of such activities include Create Task, OnTaskChanged, CompleteTask, and DeleteTask. As a developer, you have free rein to write any code you choose to satisfy the needs of your workflow. If you find that the activities provided by WWF and Win-

dows SharePoint Services do not meet your needs, you can create your own library of custom activities that can be later be reused in your workflows. In addition to defining the business logic associated with the workflow, the ASPX forms used by the workflow also need to be developed.

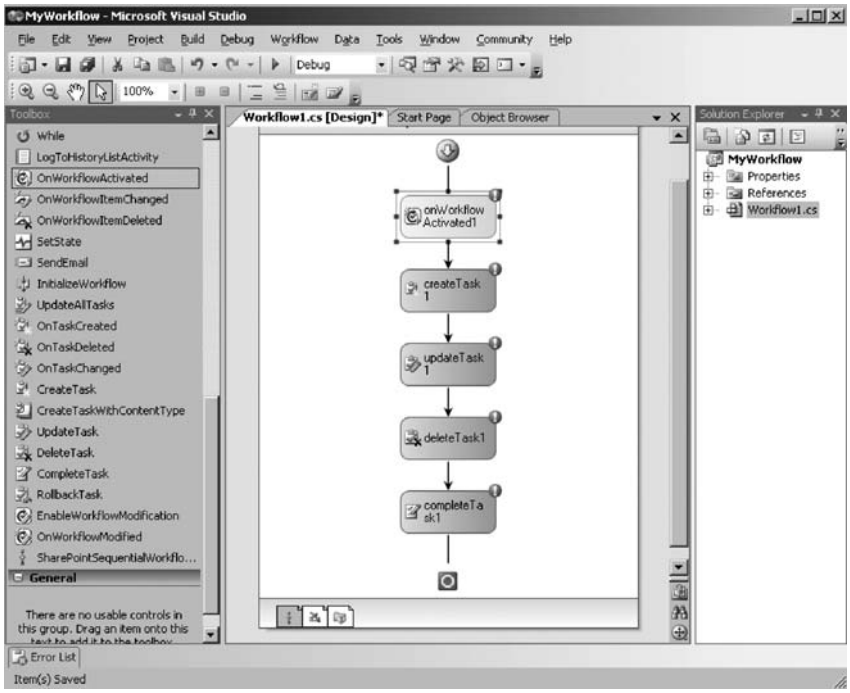


Figure 18.56
Visual Studio Workflow Designer.

Once you have all your workflow activities defined, the code behind the activities written, and the ASPX forms developed, all that remains is for you to compile and deploy the workflow. From a compilation perspective, you simply build and strongly name your assembly and install it in the global assembly cache. Once that is achieved, you need to create a Feature to deploy the workflow to SharePoint. For a basic workflow that does not include InfoPath forms, all you need are two files: your Feature definition file (Feature.xml) and your element manifest (workflow.xml). Note that the element manifest does not have to be named workflow.xml. The Feature definition file registers the SharePoint Feature receiver that is used to handle the deployment of workflows. In addition, it references the location of the element manifest files, which would include the location of any associated InfoPath forms if we were dealing with SharePoint Server 2007 workflows. Figure 18.58 shows the Feature.xml for the out-of-the-box SharePoint Server 2007 review workflows. Notice that this Feature definition references two element manifests that represent two separate workflow

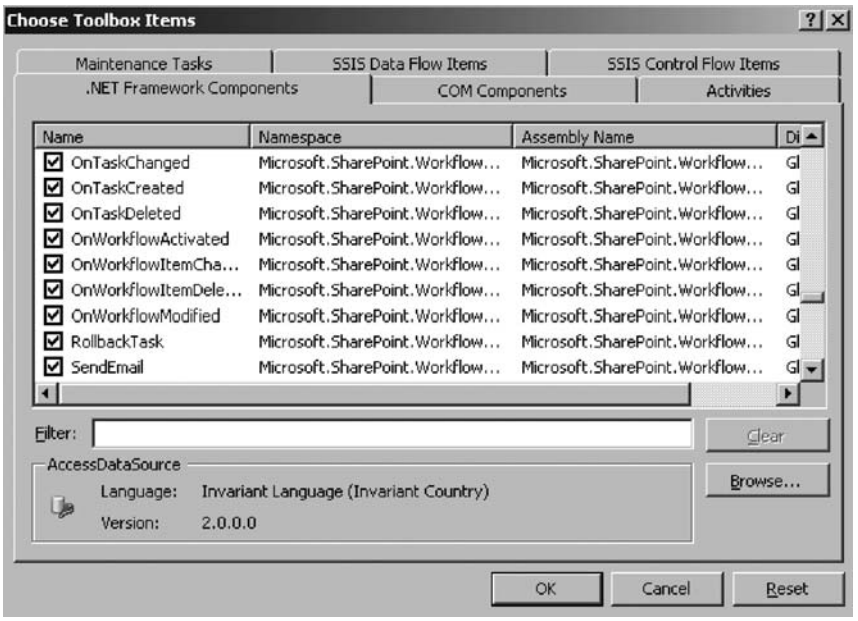


Figure 18.57 *Add SharePoint activities to Toolbox.*

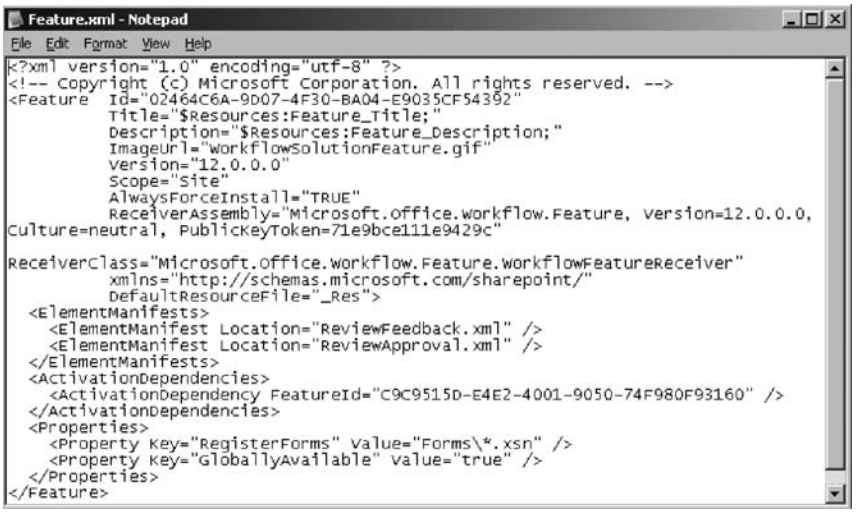


Figure 18.58 *Workflow feature definition.*

templates. Figure 18.59 shows the ReviewApproval.xml element manifest. The element manifest contains a Workflow element that is used to describe the workflow. It contains information such as the name of the workflow template; a description of the workflow, the assembly, and class that contains the code associated with the workflow; and the information relative to the forms required by the workflow. As with previous projects that we have

created throughout this book, when working with Features it is recommended that you create a folder structure in your Visual Studio project that mimics the TEMPLATE\Features path, with a subfolder containing the Feature you would like to deploy. You then leverage the solution deployment framework to deploy the assembly and Feature to the appropriate locations on the server. Developing your workflows in this way allows you to keep all your code in one place and also to package your workflows for deployment by an administrator in your production environment.

Figure 18.59 *Workflow element manifest.*

```
<?xml version="1.0" encoding="utf-8" ?>
<!--  Copyright (c) Microsoft Corporation. All rights reserved.  -->
- <Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  - <Workflow Name="$Resources:ReviewApproval_Title;"
      Description="$Resources:ReviewApproval_Description;" Id="C6964BFF-
      BF8D-41ac-AD5E-B61EC111731C"
      CodeBesideClass="Microsoft.Office.Workflow.ReviewRouting"
      CodeBesideAssembly="Microsoft.Office.Workflow.Routing,
      Version=12.0.0.0, Culture=neutral,
      PublicKeyToken=71e9bce111e9429c"
      TaskListContentTypeId="0x01080100C9C9515DE4E24001905074F980F93160"
      AssociationUrl="_layouts/CstWrkflIP.aspx"
      InstantiationUrl="_layouts/IniWrkflIP.aspx"
      ModificationUrl="_layouts/ModWrkflIP.aspx">
      <Categories />
    - <AssociationData>
      + <my:myFields xmlns:xsi="http://www.w3.org/2001/XMLSchema-
          instance"
          xmlns:my="http://schemas.microsoft.com/office/infopath/2003/myXSD"
          xml:lang="en-us">
      </AssociationData>
    + <MetaData>
    </Workflow>
</Elements>
```

18.10.2 Authoring Workflows using SharePoint Designer 2007

Authoring workflows using the Visual Studio 2005 workflow designer is easily within the capabilities of developers already familiar with working with Visual Studio 2005. While there are quite a few moving parts to get used to initially, it's not exactly complex coding. However, there are other people besides developers who would like the capability to create workflows; this is where Microsoft Office SharePoint Designer 2007 comes into play. SharePoint Designer allows users with the appropriate permissions the ability to create no-code workflows to automate business processes.

Windows SharePoint Services 3.0 and SharePoint Designer 2007 combined provide a lot of capabilities when it comes to creating workflows and associating those workflows with content stored in SharePoint. SharePoint Designer provides a graphic workflow editor that allows you to define the various steps that comprise a workflow, along with a set of conditions for

each step and its associated actions. For example, Figure 18.60 shows a very rudimentary compliance workflow that check to see if a document meets the 28CFR compliance requirement. The workflow has two steps: Review Document and Shred Document. The Review Document step has two conditions defined; the first states that if the document has been checked in and the comment field is set to "28CFR Compliant," then the approval status is set to Approved. The second condition states that if the document has been checked in and the comment field is not set to "28CFR Compliant," then the approval status is set to Rejected. The Shred Document step of the workflow has one condition associated with it; this condition basically checks the approval status, and if the approval status equals "Rejected," it deletes the document.

To create a workflow in SharePoint Designer, simply click the File menu, and then click the New menu option, followed by the SharePoint Content menu option. The New dialog box appears with the SharePoint content options. Select the Workflow option, select the Blank workflow as the template to use, and then click OK. The Workflow Designer will appear and you are free to create your workflow. The first screen that appears in the designer requests the name of the workflow, the list or library with which you want to associate the workflow, and the various ways your workflow can be invoked. SharePoint Designer workflows can be invoked in three ways: manually, through the item menu; automatically, when an item is created; and automatically, when an item is changed. The next screen that appears is the crux of the workflow definition, as this is where you define your workflow steps, conditions, and actions.

When you have finished configuring the workflow, the next step is to check and make sure no errors exist. The Workflow Designer includes an option that checks your workflow for any errors. For example, if your workflow checks the approval status of an item, but the library that the item is stored in does not have Content Approval enabled, then the Check Workflow function highlights the problem areas in your form and notifies you of the problem. If your workflow is error-free, then it is ready for deployment. The deployment process couldn't be easier; it is a simple matter of clicking the Finish button in the designer window. Once you click Finish, SharePoint Designer creates the necessary XOML files that represent the workflow and stores them in the Workflows folder at the root of your site (Figure 18.61).

At this point, if you navigate to the SharePoint document library with which you originally associated your workflow, you should be able to initiate the workflow automatically, by either creating a new item or changing an existing item, or manually, through the browser interface (Figure 18.62).

The SharePoint Designer workflow editor is much more simplistic in nature than the workflow editor provided by the Visual Studio 2005 Designer for Windows Workflow Foundation, and is geared towards the

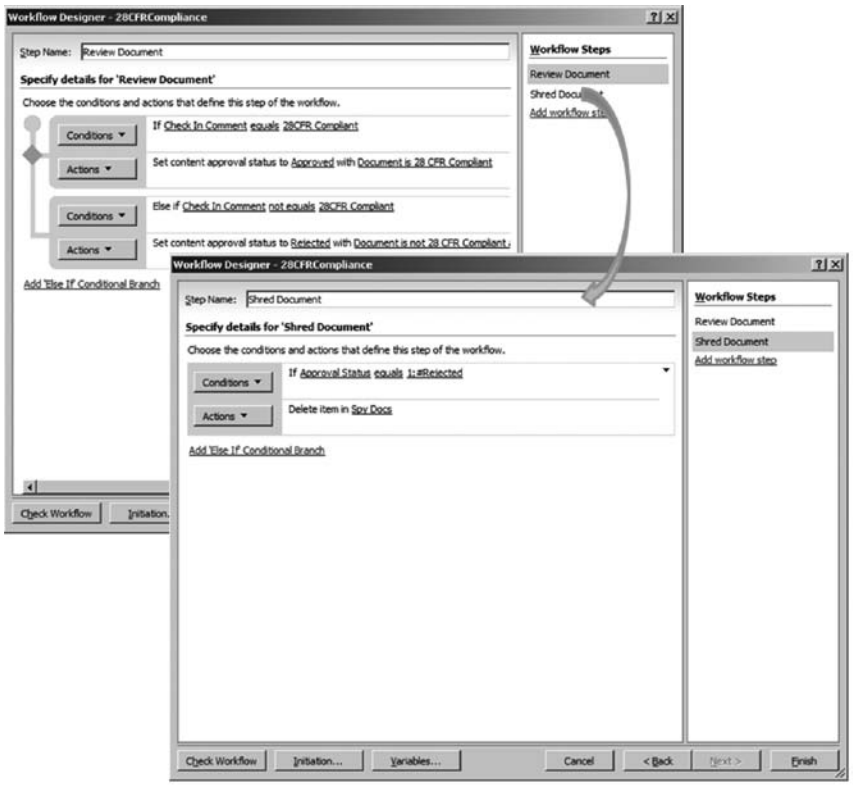


Figure 18.60
Workflow Designer.

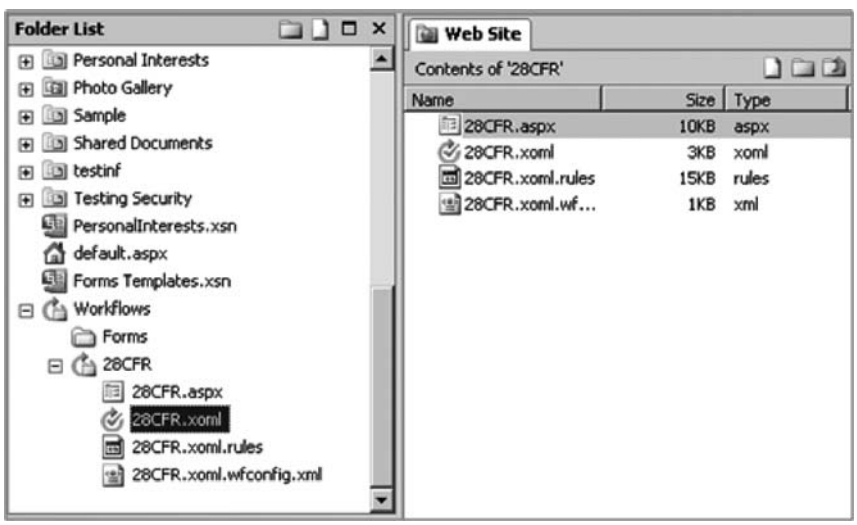


Figure 18.61
Workflow files generated by SharePoint Designer.

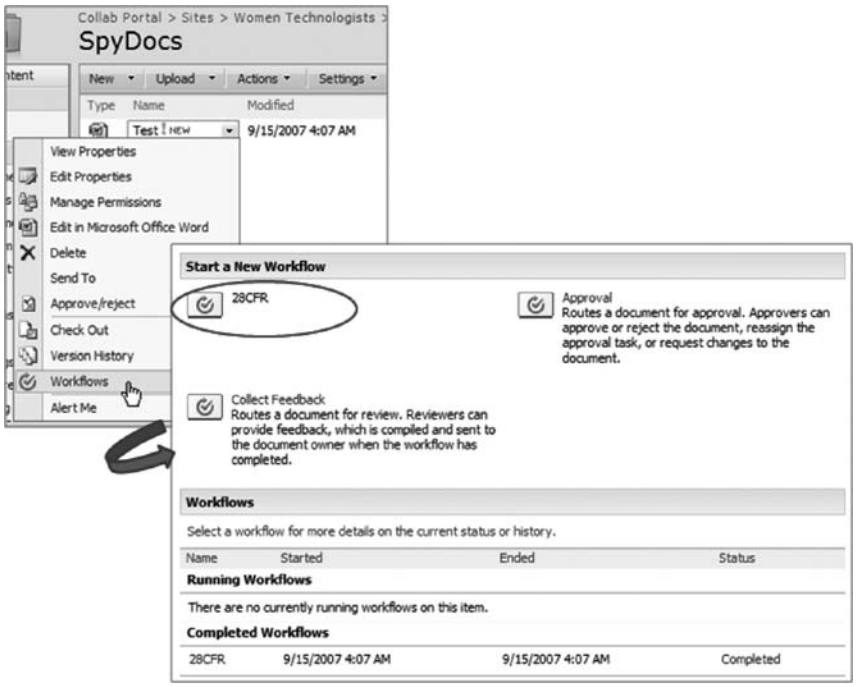


Figure 18.62
Initiating workflow in browser.

rapid development of workflows that do not require custom code or custom activities.

18.10.3 Authoring Workflows in SharePoint Server 2007

SharePoint Server 2007 is built upon Windows SharePoint Services 3.0, and as such, inherits many of the functions and capabilities of the base platform. Therefore, it should come as no surprise to hear that the same two options exist for authoring workflows targeted at SharePoint Server 2007: namely, Visual Studio 2005 with WWF extensions applied, and SharePoint Designer 2007. Everything we have discussed thus far with respect to these tools is applicable to SharePoint Server 2007, with one exception. SharePoint Server 2007 supports the use of InfoPath Forms in its workflow templates. Therefore, any workflow template that you author using Visual Studio that is targeted to SharePoint Server 2007 can include InfoPath Forms. SharePoint Designer 2007 is still limited to the use of ASPX forms only. It should be noted that the integration with InfoPath that we are referring to here is using InfoPath forms to display information to the user throughout the workflow process, such as providing them with instructions on the task that they must perform in the workflow process. We are not referring to the ability to associate a workflow with an InfoPath content type or a form library. InfoPath

Forms installed into a form library are treated just like documents are in a document library; therefore, you can use the out-of-the-box workflows to route an InfoPath form through a workflow process simply by associating the form library with your workflow of choice. In this instance, you can leverage SharePoint Designer to route InfoPath forms.

The Microsoft Office SharePoint Server 2007 SDK is another great resource to add to your workflow authoring toolkit. When you install the SDK, you will find a whole suite of workflow templates with sample code and forms that you can leverage in your own SharePoint workflows. It is highly recommended that you review these templates and the associated documentation before you begin your own custom workflow development, as you never know what treasures you may find. For example, the SDK includes a set of Visual Studio workflow project templates specifically designed for building SharePoint workflows. These workflow templates set up your environment in Visual Studio by creating your folder hierarchy for deploying your workflow, loading the SharePoint activities into the toolbox, providing XML code snippets to aid with the creation of the Feature definition and the element manifest XML files, and providing post-build commands and .bat files to aid deployment. In addition, the template creates your .ddf file and the necessary manifest.xml file required for leveraging the solution framework for deploying your workflow as a solution (Figure 18.63).

Figure 18.63
Project files and folder hierarchy.

18.11 Integrating InfoPath Forms with Workflow

In this section, we examine the various components of integrating InfoPath forms with workflow. To help you better understand the concepts that we

discuss, we recommend that you install the SharePoint Server SDK and open up the HelloWorldSequential workflow project. The HelloWorldSequential project is a simple example that illustrates the integration between InfoPath and workflow. Having the project open will be helpful, because as we discuss different aspects of the integration, you will have a point of reference and a visual representation of the topic. Let's begin.

You can create a number of forms for SharePoint workflows, depending on which part of the workflow process you would like to customize. The forms include Association forms, Initiation forms, Modification forms, and Task forms. Association forms are the forms that administrators, or privileged users, use to associate a workflow template with a specific list, document library, or content type. Figure 18.64 shows a section of the Association form used to associate the out-of-the-box Approval workflow with a document library. Initiation forms are the forms presented to a user when they want to initiate a workflow; in other words, when they want to create a workflow instance. Figure 18.65 shows the Initiation form of the same Approval workflow. So what is the difference between these two forms? The Association form is used to configure the default settings of the workflow *before* it is instantiated. Data, such as the list of participants, can be captured and provided to the Initiation form for further processing.

Modification forms are forms with which a user can modify the workflow after it has been initiated. For example, if we look again at the out-of-the-box Approval workflow, it has an associated Modification form that allows a user to add additional approvers to the workflow process while the workflow is actually running (Figure 18.66). Finally, the Task form type is a special type of form used to manage the workflow tasks created in the task list during the workflow process. For example, if a user edits an item in the workflow task list, they are presented with a corresponding Task form, and they can use this form to make changes to the workflow item (Figure 18.67).

When navigating to each of the aforementioned forms in the browser, you will notice that the URL references an ASPX page each time. Association forms use the CstWrkflIP.aspx page, Initiation forms use the IniWrkflIP.aspx page, Modification forms use the ModWrkflIP.aspx page, and Task forms use the WrkTasIP.aspx page. Figure 18.68 shows the element manifest for the Approval workflow which contains three attributes (`AssociationUrl`, `InstantiationUrl`, and `ModificationUrl`) that inform SharePoint which custom ASPX page to use for a given stage in the workflow. The `MetaData` element in the element manifest contains attributes that define the URN for the InfoPath forms to be loaded at each stage. The `Association_FormURN` attribute contains the URN of the InfoPath form to be used as the Association form; the `Initiation_FormURN` attribute contains the URN of the InfoPath form to be used as the Initiation form; the `Modification_GUID_FormURN` attribute contains the URN of the InfoPath form to be used as the Modification form; and

Figure 18.64
Approval workflow Association form.

finally, the `TaskId_FormURN` contains the URN of the InfoPath form to be used as the Task form. The Approval workflow leverages multiple Modification and Task forms.

The custom ASPX pages work generally in the same way the FormServer.aspx page does to display InfoPath forms in a form library. When SharePoint Server 2007 displays a custom InfoPath form associated with some stage in the workflow process, such as the initiation stage, it references the workflow attributes to determine the custom ASPX page to be used, such as IniWrkflIP.aspx. SharePoint then references the corresponding attribute, such as the `Initiation_FormURN` attribute, within the `MetaData` element to retrieve the URN of the form to be loaded. SharePoint calls the Form Server control within the custom ASPX page, which loads the InfoPath form into the page for rendering in the browser. Task forms are processed a little bit differently, in that a workflow task is basically represented as a SharePoint item that is assigned a specific content type. Given that content types can have InfoPath forms associated with them, it is the content type that determines the forms to use for the

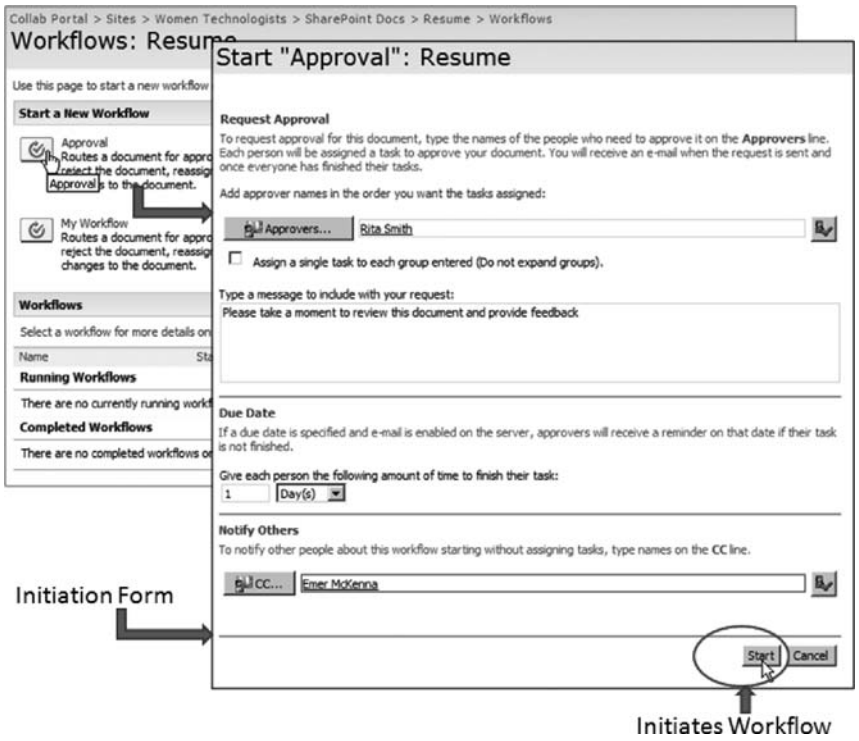


Figure 18.65
Approval workflow Initiation form.

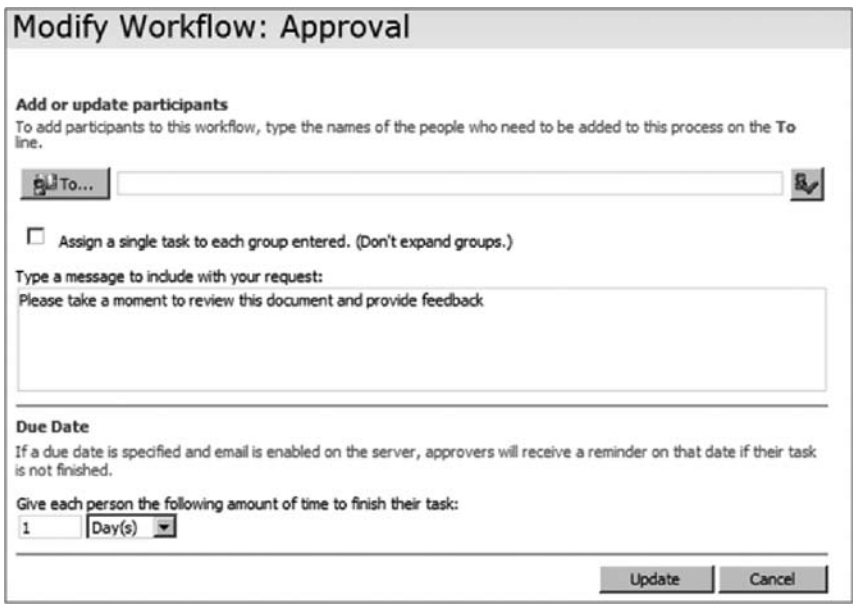


Figure 18.66
Approval workflow Modification form.

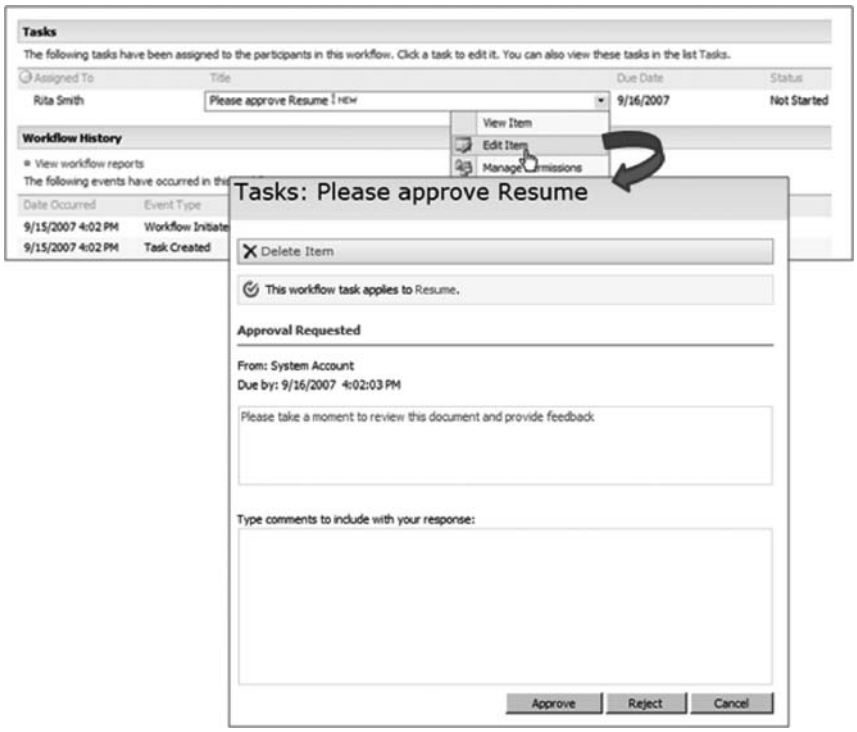


Figure 18.67
Approval workflow Task form.

specific task types in your workflow. When you want to use a custom InfoPath form as a Task form, you must set the `TaskListContentTypeId` attribute of the `Workflow` element to a specific ID that uses custom task display and edit forms, including the Form Server control, for rendering InfoPath forms. The Workflow element within the element manifest of the workflow contains the `TaskListContentTypeId` attribute. The `TaskListContentTypeId` that attribute SharePoint you are using custom InfoPath forms in your workflow. If you examine the element manifest of the Windows SharePoint Services 3.0 three-state workflow, you will notice that the `TaskListContentTypeId` attribute is not defined; the reason is that the three-state workflow does not use InfoPath forms in its workflow.

During the form load, the Form Server control passes any data captured at the association stage to the form in XML format. For example, when the user enters data into the Association form of the Approval workflow, he or she can populate the lists of reviewers that will review the document. This list can then be passed onto the Initiation form, which instantly populates some of its fields based on this data. The fields that you want to capture from the Association form are listed in the `AssociationData` element of the workflow's element manifest. Figure 18.70 shows the `AssociationData` element as defined in the element manifest for the Approval workflow.

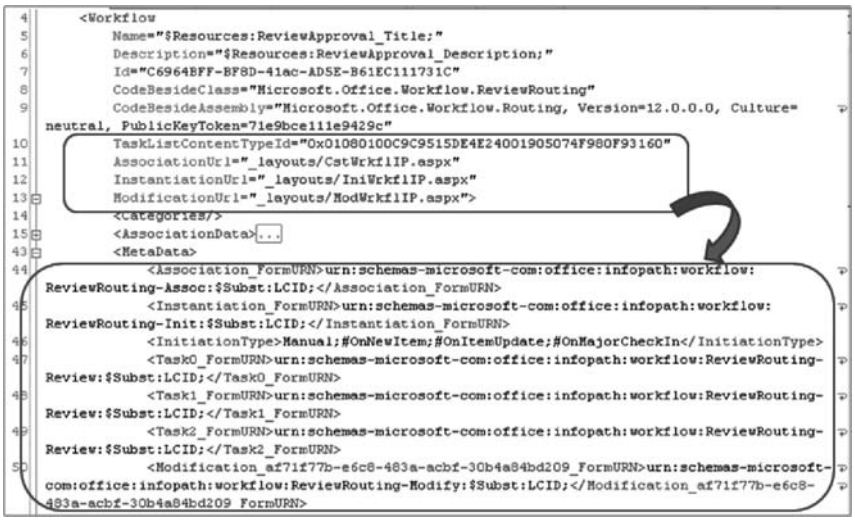


Figure 18.68
Approval workflow element manifest.

The Form Server control data-binds any elements in the workflow to controls on the form that share the same name. For example, if you have an element in your workflow named Grade, and wanted to bind this data to a control on your InfoPath form, you would also name the control Grade. In this scenario, there is no need to create secondary data sources, as we did earlier in this chapter; the Form Server control "magically" handles it all for you behind the scenes. The key takeaway here is that if you want data from a field to be shared between your InfoPath forms and the workflow, you *must* use the same field name in both areas.

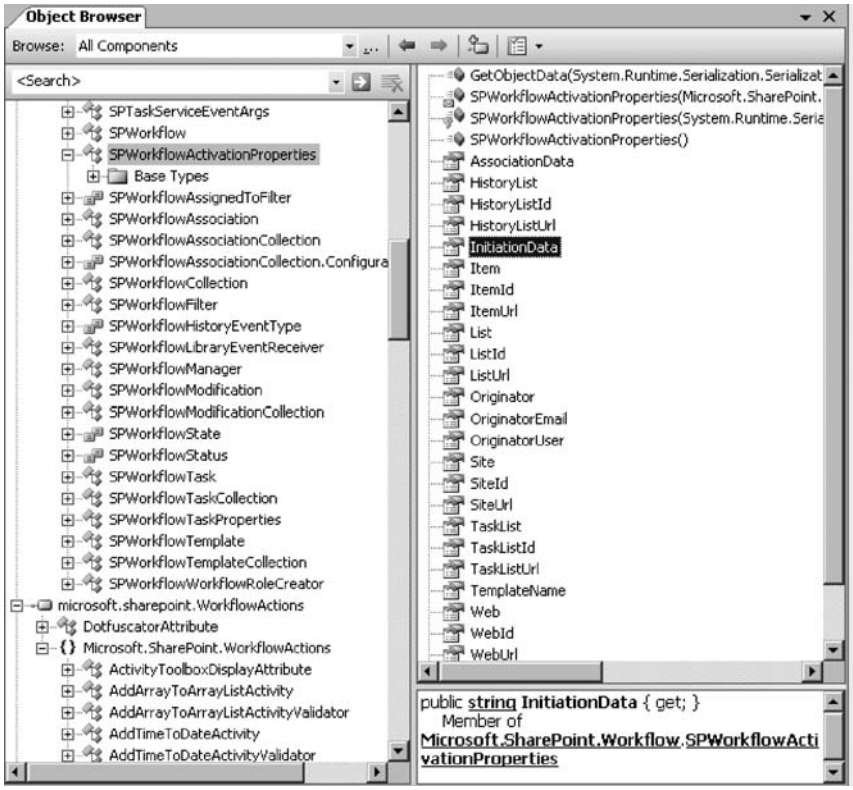
After the workflow starts, the data captured from the Initiation form is stored in the `InitiationData` property of the `SPWorkflowActivatedProperties` object. As mentioned earlier in this chapter, when developing SharePoint workflows the `onWorkflowActivated` activity must be the first activity in the workflow, as it acts as a gateway to SharePoint. The `onWorkflowActivated` activity contains a `WorkflowProperties` property, which returns a `SPWorkflowActivatedProperties` object. The `SPWorkflowActivatedProperties` object provides access to a whole set of useful properties, such as the `Item` object for the you are working with, the `List` object for the list to which the workflow is associated, the `WebId` of the Web site in which the workflow is activated, and even the `TaskList` object for the task list associated with the workflow. Figure 18.70 shows the `SPWorkflowActivatedProperties` object examined using the Visual Studio object browser.

When it comes to actually designing InfoPath forms for use in workflow, there are a few key steps that you need to follow in order for the integration to run smoothly:

Figure 18.69 *AssociationData element in Approval workflow.*

```
<AssociationData>
    <my:myFields xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:my="http
://schemas.microsoft.com/office/infopath/2003/myXSD" xml:lang="en-us">
        <my:Reviewers></my:Reviewers>
        <my:CC></my:CC>
        <my:DueDate xsi:nil="true"></my:DueDate>
        <my:Description></my:Description>
        <my:Title></my:Title>
        <my:DefaultTaskType>1</my:DefaultTaskType>
        <my:CreateTasksInSerial>true</my:CreateTasksInSerial>
        <my:AllowDelegation>true</my:AllowDelegation>
        <my:AllowChangeRequests>true</my:AllowChangeRequests>
        <my:StopOnAnyReject xsi:nil="true"></my:StopOnAnyReject>
        <my:WantedTasks xsi:nil="true"></my:WantedTasks>
        <my:SetMetadataOnSuccess>false</my:SetMetadataOnSuccess>
        <my:MetadataSuccessField></my:MetadataSuccessField>
        <my:MetadataSuccessValue></my:MetadataSuccessValue>
        <my:ApproveWhenComplete>false</my:ApproveWhenComplete>
        <my:TimePerTaskVal xsi:nil="true"></my:TimePerTaskVal>
        <my:TimePerTaskType xsi:nil="true"></my:TimePerTaskType>
        <my:Voting>false</my:Voting>
        <my:MetadataTriggerField></my:MetadataTriggerField>
        <my:MetadataTriggerValue></my:MetadataTriggerValue>
        <my:InitLock>false</my:InitLock>
        <my:MetadataStop>false</my:MetadataStop>
        <my:ItemChangeStop>false</my:ItemChangeStop>
        <my:GroupTasks>false</my:GroupTasks>
    </my:myFields>
</AssociationData>
```

Figure 18.70 *SPWorkflowActivationProperties object.*



1. ***Ensure form is browser-enabled***: From the InfoPath Tools menu, click Form Options and select the Compatibility category. Make sure to check the Design a form template that can be opened in a browser or InfoPath checkbox. To later check the compatibility, enter the URL of the server that is running Forms Services. Click OK.
2. ***Design your form template***: Start designing your InfoPath form as you would a regular InfoPath form. For example, design the form layout, change the color scheme, and insert the field controls onto the form as you would with a regular form.
3. ***Data-bind the controls on your form template***: After you have inserted your field controls on the form, ensure that you give your InfoPath form control the same name as you intend to use in the workflow.
4. ***Add and configure the Submit button***: Use the Submit button to submit the form and its associated data to SharePoint Server 2007. For the Submit button to be configured to actually submit the form, a rule consisting of two actions needs to be added. The first action is configured with a Submit Data Connection. Use the Data Connection Wizard to create a new connection to submit data to the hosting environment. Once the form has been submitted to SharePoint, we need to make sure that the form is closed. This requirement is handled with a second action, which adds a Close the Form action. The rule and its associated actions should look similar to the configuration shown in Figure 18.71.

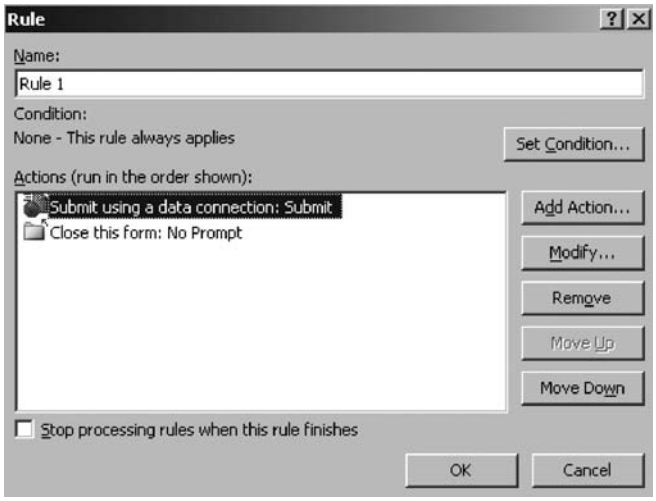


Figure 18.71
Configuring the Submit button actions.

5. ***Configure secondary data source to access task data***: This step is optional and only required if you want to load task data into a task form without writing code. Since tasks are just SharePoint list items, they are susceptible to customization that changes the underlying item schema. For example, if a column is added to the task list, then the item schema for a task in that task list changes. To prevent this from causing issues with your workflow, Microsoft allows you to create a schema file (ItemMetadata.xml) in which you can list the SharePoint columns that you would like your form to access. Figure 18.72 shows an example of the contents of the ItemMetadata.xml file from the HelloWorldSequentialWorkflow sample project in the SDK.

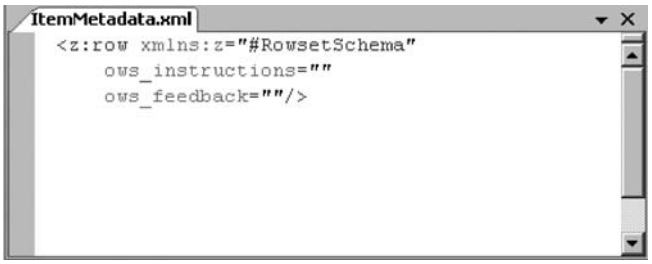


Figure 18.72 *ItemMetaData.XML.*

6. ***Set the form's security level to Domain***: From the InfoPath Tools menu, click Form Options, select the Security and Trusts category, uncheck the Automatically determine security level checkbox, and select the Domain option.
7. ***Check the form for inconsistencies***: Open the Design Checker task pane, set the Verify on server checkbox to checked, and click Refresh. If the Options button is visibl, click it and ensure that browser compatibility messages are being displayed. Correct any inconsistencies that you find before publishing your form template.
8. ***Save the Form***: When saving the form, make sure that you save it to a different location than you intend to publish; otherwise, you will receive conflict errors during the publication cycle. In other words, do not save your form to the workflow project folder.
9. ***Publish the form to workflow project folder***: In this final step, publish the form to the same folder in which the workflow project is contained. Keeping the published form with your workflow project is the best practice, as it keeps all your project resources together and makes the deployment process easier. From the File menu, click Publish, select the option to publish to a network location, click Next, enter the location of your work-

flow project folder suffixed with your form name, enter the form template name, and click Next. The publishing wizard will prompt you to enter an alternate access path to your form. It is important that you leave this setting blank. If you do not leave the setting blank, an error occurs that prevents the form from being published to the server. So clear this setting (Figure 18.73) and click Next. If you receive a warning message complaining about the fact that you don't have an alternate access path specified, don't worry-just click the OK button and your form should be published to the projects folder, ready for deployment.

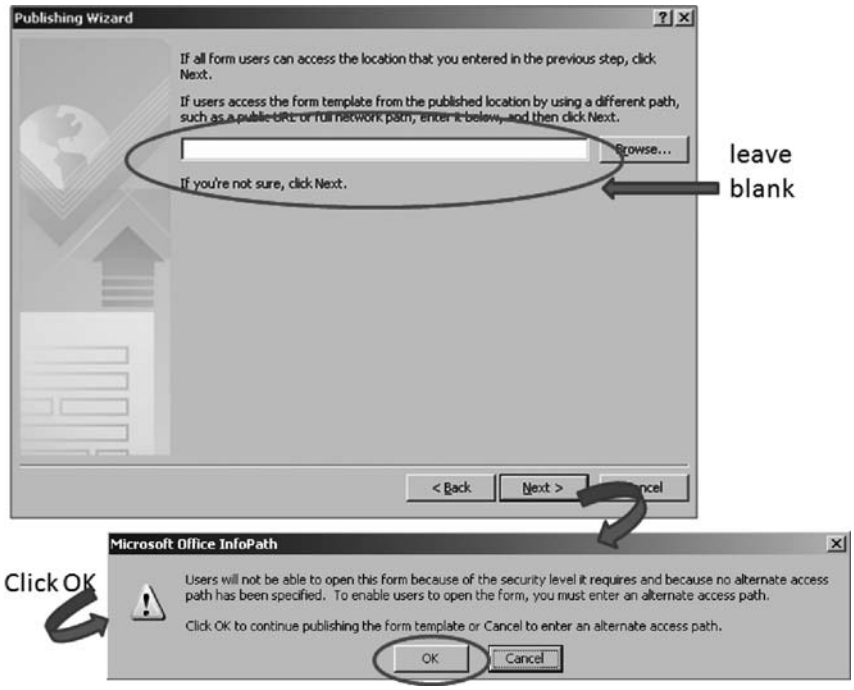


Figure 18.73
Alternate access path.

InfoPath Forms, in conjunction with the workflow capabilities offered out of the box with SharePoint Server 2007, provide developers with a powerful development platform from which to build applications and customize SharePoint to suit specific business needs. Now that you understand the integration aspects of workflow and InfoPath forms, all that remains is for you to start developing your own workflow applications. An excellent starting point is to roll up your sleeves and dissect the sample applications provided with the SharePoint Server 2007 SDK.

18.12 Conclusion

This book has provided you with the basic knowledge and tools to begin designing and customizing your SharePoint Server 2007 implementation. Obviously, your needs will be more specialized and require knowledge beyond what we have been able to provide. Fortunately, there are many more resources at your disposal, such as the SharePoint Server 2007 and Windows SharePoint Services 3.0 SDKs, resource kits, newsgroups, and blogs, not to mention the multitude of community sites devoted to SharePoint.

That said, we hope that you found the content in this book helpful when deploying SharePoint, and we appreciate the fact that you used us as a resource.

Index

3-tier architecture, 13–15
  Application tier, 14
  Database tier, 14
  Web Front End (WFE) tier, 14
2007 Microsoft Office System, 1–9
  desktop programs, 2–3
  servers, 3

Access 2007, 183
Actions, 83
`ActivateFeature` operation, 488
`Activateformtemplate` operation, 600–601
Activation dependencies, 480–82
  cross-scope, 482
  defined, 480
  rules, 480–82
  same-scoped, 482
`ActivationDependencies` element, 480–82
Active Directory (AD)
  connection settings, 146
  forest, 61
  objects, 62, 63
Active monitoring, 231–32
`AdditionalFilterFields` property
  defined, 328
  illustrated, 334
  using, 334
  See also Content query Web Part
`AdditionalGroupAndSortFields` property
  defined, 328
  illustrated, 335
  using, 334
  See also Content query Web Part
Add Web Parts Webpage Dialog, 314
Ad-hoc workflow, 617
Administration Web service, 299
Administrator-deployed form templates, 586, 596–603
  activating, 598–601
  upgrading, 601–3
  uploading to server, 596–98
  See also Form templates
Administrators group, 42
Advanced versioning, 34
Alerts, limit recommendation, 203
Alerts Web service, 299
`AllowPartiallyTrustedCallers` security attribute, 345
`AllowSafeUpdates` method, 296
`AllUsersWebPart` element, 452–53
Anchor text, 119
Application definition files (ADFs), 82–85
  actions, 83
  associations, 84
  components, 82–83
  contents definition, 84
  defined, 82
  entities, 83, 86
  filters, 83
  illustrated, 85
  importing, 85–86
  LOBSystemInstance, 83
  methods, 83
Application.master, 399–400
Application pages, 273
Application templates, 446–47
Approval, 90
ASP.NET 2.0
  handler, 238

Master directive, 391
Membership provider interface, 240
pages, 11, 310
provider models, 240
test.master master page, 392
Web Part Framework architecture, 239
Web Parts, 310
WSS 3.0 support, 12
Aspx pages, 48, 633–34
Assembly file, 278–79
Associations, 84
Asynchronous events, 513, 514
Audiences, 160–61
compiling, 160
defined, 143
for rule-based groups, 160
Auditing, 91–93
configuring, 92
events, setting, 92
levels, 91–92
reports, 92–93
Authentication, 42
embedded SQL, 563
providers, 190–91
Authentication Web service, 299
Authoring workflows, 623–32
in SharePoint Designer 2007, 628–31
in SharePoint Server 2007, 631–32
in WSS 3.0, 624–28
See also Workflows
Authoritative pages, 131
Authorization, 37–42
elements, 37–38
See also Security model
AutoCopy, 34

Backups, 218–19
out-of-the-box options, 218
third-party solutions, 218–19
BDR site definition, 432
Best bets, 131–32
BLANKINTERNETCONTAINER site definition, 432
BLANKINTERNET site definition, 432
Blogs, 51–53, 154
comments, 52, 54
content display, 52–53
defined, 51
on mobile, 57
site illustration, 52
Blog site definition, 432
Branching, 67
Brand Feature, 498–99
`ChangeBranding` class, 500, 502
defined, 498
Feature.xml, 500
files and folders, 499
ProvisionFiles.xml, 500
Branding, 383–427
application.master and, 399–400
applications, 384
browser customizations and, 422–23
color scheme, 407–11
consistency, ensuring, 418–19
defined, 383
enterprise content management, 388–90
entire site, 424–27
exercises, 421–27
with Features, 496–511
Feature set, 506–11
files, keeping together, 420
identification, 417
inconsistent, 416
master pages, 390–99
navigation, 402–7
page layouts, 400–402
pain points, 384–88
publishing page rendering, 402, 403
SharePoint 2007 enhancements, 388–407
site definition exercise, 455–62
site description, 420–21
site icon, 420–21
site title, 420–21
Browser customizations, 422–23
Browser-enabled form templates, 573
controls, 582–84
creating, 578–85
form verification, 584–85
layout, 579–80
secondary data source conversion, 582
secondary data sources, 580–81
user settings, 586
See also Form templates
Business Data Catalog (BDC), 7, 81–88, 242
application definition file, 82–86
back-end data access, 82
Data Source Library, 320

defined, 81
entities in, 86
illustrated, 83
lists and, 87–88
mapping properties to, 150
searching, 137–40
search results, 139
targeted results, 139
Web Parts, 86–87
Business intelligence, 9, 163–73
dashboards, 171–73
Excel Services, 164–69
KPIs, 169–71
Business processes, 8

Call volume, support, 228–29
Cascading style sheets (CSS), 407–9
class identification, 408
defined, 407
sample code, 408
script applied to master page, 409
URL setting, 410–11, 420
Catalogs, 118
CENTRALADMIN site definition, 432
Certificate authority (CA), 612
Check-in/check-out, 90
library support, 36
versioning and, 35
Child controls, Web Parts, 359–61
Closed Web Parts gallery, 317
Code-behind files, 277–78
Coding Web Parts, 354–66
child controls, 359–61
custom, 347–48
editor parts, 362–66
menu, 361–62
properties and methods, 355–59
rendering process, 354–55
tool panes, 362–66
See also Web Parts
Collaboration, 5–6, 45–67
extranet, 191
team lists, 24–25
Collaboration Application Markup Language (CAML) format, 295, 332–33, 455
Collaboration portal, 78–81, 403
document center, 80
home page, 79–80
illustrated, 78
news, 80–81
report center, 81
search, 81
site definition, 447
sites, 81
with Tree View, 407
Colleague tracker Web Part, 156
Color schemes, 407–11
cascading style sheets, 407–9
custom style sheets, 409–11
See also Branding
Columns
defined, 29
indexing, 32
limit recommendation, 201
`CommonViewFields` property
custom field listings in, 331
defined, 328
field listings in, 329
internal column name requirement, 330
See also Content query Web Part
Compliance, 90–91
as core business tenet, 90
resources, 91
strategy, 94, 103
Comprehensive indexing, 120
Configuration database
defined, 20
SiteMap table, 21
`Configurations` element, 450–54
Connecting Web Parts, 366–73
Contacts, 64–66
attributes, 64
in GAL, 65
issues, 64–66
Content
adding to SharePoint, 290–93
approval, 35
database, 20
management, 7–8
pages, 23, 394
removing from SharePoint, 293–95
updating, 290
Content deployment, 111–13
job definition, 112–13
path definition, 112–13
steps, 111–12
Content Management Server (CMS), 106

Content query Web Part, 76, 110, 325–41
AdditionalFilterFields property, 328, 334
AdditionalGroupAndSortFields property, 328, 334
capabilities, 326–27
CommonViewFields property, 328, 329–32
customized, 341
DataColumnRenames property, 328, 332
default fields display, 328
defined, 325
exercise, 338–41
Filter properties, 329, 334–37
HeaderXslLink property, 337
ItemXslLink property, 337
ListsOverride property, 328, 333–34
properties, 328–29
QueryOverride property, 328, 332–33
results, 327
scope, 325
tool pane, 326
ViewFieldsOverride property, 328, 334
WebsOverride property, 328, 333
Content sources, 122–23
configuration, 122–23
defined, 122
limit recommendation, 203
single, 122
Content types, 90
defined, 30
element manifest, 541
enabling, 31
Feature.xml, 540
galleries, 31
inheritance and, 31
list, 31
metadata and, 30
page layouts and, 400
registering event handlers with, 540–42
site columns, 32
user form template deployment to, 592–94
Contributor mode, 243–44
Controls
adding, 582–83
data-blind, 639
mapping to data sources, 583–84
Copy Web service, 300
Core engine, 116–19
Counters, performance, 234–35
Crawled properties, 128
Crawl rules, 123
CreateChildControls method, 360
CreateEditorParts method, 366
Cross-platform migrations, 220–21
Current navigation, 406
Custom actions, 475–79
CustomActions element
defined, 475
ID attribute, 476
Rights attribute, 476
Title attribute, 476
Custom masters
content management lifecycle and, 414
defined, 393
page directives for, 395
page identification, 398
pages referencing, 415
setting, 395
subsites referencing, 413
See also Master pages
Custom site definitions, 385, 441, 442
registration, 442
solution, 456
team, creating, 458
without Feature associations, 443
Custom site templates, 441, 445–46
Custom style sheets, 409–11
Custom templates, 385
Custom themes, 415–18
creation steps, 417–18
template definitions, 417, 418
See also Themes
Custom Web applications, 280–84
Custom Web pages, 267–80
accessibility, 268
assembly file, 278–79
best practices for creating, 270–71
developing in Visual Studio 2005, 270
storing code behind, 277–78
Custom Web Parts, 341–54
adding to Web Part gallery, 351–52
adding to Web Part page, 352–54
AllowPartiallyTrustedCallers security attribute, 345
building assembly, 349
build output path specification, 343–45
code insertion, 349
coding, 347–50

creating, 341–54
creation steps, 342–47
deploying, 351–54
naming assembly and namespace, 347–48
project creation, 343
public key extraction, 349–50
registering, 351
renaming, 348–49
strongly named assembly, 346–47
version number, 346

Dashboards, 171–73
defined, 171
filtering, 172, 173
predefined, 171
Database sizing, 210
`DataColumnRenames` property
defined, 328
illustrated, 332
using, 332
See also Content query Web Part
Data Connection Library (DCL), 574–75, 578
Data connections, 574–78
converting, 577–78
response size, 562
UDC, 575
Data migrations
cross-platform, 220–21
functionality changes, 221–22
planning, 219–22
structure changes, 219–20
Data Source Library, 318–20
Business Data Catalog, 320
Database Connections, 319
defined, 318
Linked Data Sources, 320
multiple data provider definitions, 325
Server-side Scripts, 319
SharePoint Libraries, 319
SharePoint Lists, 319
XML Files, 319
XML Web Services, 319–20
Data sources
configuration, 574
default, 574
joining, 321
linked, 320–21
mapping controls to, 583–84
merging, 321
secondary, 574, 580–83, 640
Data views
creating, 322–25
joined subviews, 324, 325
multiple item, 323
Data view Web Part, 317–18
.ddf files, 260
`DeactivateFeature` operation, 490
Debugging Web Parts, 374
Default.aspx, 282–83
Default master, 396
defined, 393
navigation areas, 403
page directives, 395
See also Master pages
Definitions
defined, 132
extraction, 133
Deploying Web Parts, 351–54
Deployment hardware topologies, 206–7
Deployment sizing, 195–203
application limits, 200–203
performance testing methodology, 196–97
throughput estimation, 197–200
See also SharePoint deployment
Design a Form Template, 573
Design Checker, 571
Development process, 224
Digital signatures, form templates, 611–15
Directory Management Service (DMS), 61, 62–67
configuring, 63
Contacts, 64–66
defined, 62
Distribution Groups, 66–67
functions, 62–63
Discussion lists, 67
Disk subsystem planning, 209–11
Distributed server deployment, 212–13
Distribution Groups, 66–67
Divestitures, 220
DNS architecture
aliases, 215
pointer naming standards, 215
preparing, 215–16
Doc Ave, 218–19
Document Information Panel, 603–7
custom, 607

default Word 2007, 605
defined, 604
replacing, 604
SharePoint site, 605–6
template, creating, 606
Document management, 90–97
auditing, 91–93
compliance, 90–91
information management policies, 93–96
rights management, 96–97
Document Workspace Web service, 300
Duplicate detection, 137, 138

Editor parts, 362–66
creating, 363, 366, 368–69
illustrated, 369
Element manifests, 471–73
approval workflow, 637
content type, 541
defined, 471
file provisioning, 474
illustrated, 468
workflow, 628
Elements.xml, 537, 549, 552
Elevated privileges, 295–96
E-mail
delivery to lists/libraries, 60–62
filing issues, 103
incoming, 58–62
managed folders, 105
records and, 103–5
settings, configuring, 59–60
WSS 3.0 and, 58–67
Enterprise Content Management (ECM), 7–8, 89–113, 388–90
approval stage, 389
defined, 89
document management, 90–97
options in master page gallery, 389
publishing stage, 389
records management, 97–106
tools/technologies, 90
Web Content Management (WCM), 106–13
Enterprise search scoping, 191–92
Entities, 83
Event handler registration, 534–36
with content types, 540–42
document library, 544
exercise, 549–52
with Features, 536–40
with object model, 543–44
removing, 542
removing with object model, 544–46
Event handlers
creating, 533–34, 548–49
deploying, 534, 548–49
logic, 514
testing, 553–56
Event receivers, 515–32
default, Web Part showing, 545
defined, 515
Feature receivers as, 531
`SPEmailEventReceiver` class, 531–32
`SPEventReceiverBase` class, 515–16
`SPItemEventReceiver` class, 520–28
`SPListEventReceiver` class, 528–31
`SPWebEventReceiver` class, 516–20
Events, 513–56
asynchronous, 513, 514
canceling, 517
defined, 513–14
exercise, 546–56
registered, Web Part, 535, 536
synchronous, 514
working with, 533–46
WSS 2.0 support, 532
WSS 3.0 architecture, 514
Excel 2007, 183
Excel Services, 9, 164–69, 242
core parts, 164
defined, 164, 242
Excel Web Service, 242
power of, 167
Excel Web Access, 166
interacting with workbooks via, 168–69
viewing via, 167
Excel workbooks, 164
creating KPIs from, 170
interacting via Excel Web Access, 168–69
publishing, 165–66
viewing in browser, 166–67
`ExecuteUrl` element, 451
Expiration policy, 95
Extensible stylesheet transformations (XSLT), 317
Extranets, 617

Farm Administrators group, 42
Farm Features, 464
Feature element, 466–71
ActivateOnDefault attribute, 467, 470
AlwaysForceInstall attribute, 470
attributes, 467–71
AutoActivateInCentralAdmin attribute, 471
Creator attribute, 469
DefaultResourceFile attribute, 471
defined, 466
Description attribute, 469
Hidden attribute, 470
Id attribute, 469
ImageUrlAltText attribute, 469
ImageUrl attribute, 469
ReceiverAssembly attribute, 470, 482–83
ReceiverClass attribute, 470, 482–83
RequireResources attribute, 471
Scope attribute, 467, 468, 470
SolutionId attribute, 471
Title attribute, 469
Version attribute, 469
Feature receivers, 482–83
creating, 505–6
defined, 482
as event receivers, 531
Features
activated, database entries, 489
activating, 487–90
activation dependencies, 480–82
architecture, 466–82
assemblies, 484
benefits, 464
Brand, 498–99
branding with, 496–511
creating, 484–86
custom actions, 475–79
deactivating, 490–91
defined, 24, 463–64
deploying with solution framework, 493–96
directory, 442
element manifests, 468, 471–73
elements, 466–71
element types, 472–73
Farm, 464
files used in, 494–95
folders, 466
identifying for stapling, 442
installing, 464, 486–87
lifecycle, 483–91
management pages, 487
module elements, 473–75
MOSS, 72–75, 464–66
properties, 479
PublishingLayouts, 464–66
registering event handlers using, 536–40
scope, 467–68
Site, 464
Site Collection, 467–68
three-state workflow representation, 622
uninstalling, 491
use of, 464
Web Application, 464
FeatureSiteTemplateAssociation element, 492, 504
Feature stapling, 441–45, 491–93
defined, 491
Feature identification for, 442
illustrated, 492
Feature.xml file, 443, 537, 552
content type, 540
file provisioning, 475
workflows, 626, 627
Field controls, 108
File element, 452
Filter properties, 334–37
defined, 329
illustrated, 336
property usage, 334–35
values, 336
Filters, 83
duplicate, 129
search engine, 118
Folders, 32–33
Features, 466
managed, 105
themes, 415
turning on/off, 547
Form1_Load method, 306
Forms, 242–43
administrator-deployed, 586
approval workflow association, 634
approval workflow initiation, 635
approval workflow modification, 635
approval workflow task, 636
browser-enabled, 639
data source association, 570–71

inconsistencies, 640
integrating with workflow, 632–41
modification, 633
opening in browser, 591, 592
postbacks and, 563
publishing, 640–41
saving, 566, 640
security level, 640
session state, 566
Submit button, 639
time reporting, 612
updating, 565
upload verification error, 597
user-deployed, 586
verifying, 584–85
Forms Web service, 300
Form template designer, 571–73
Form templates
activating, 598–601
adding code to, 608–16
administrator-deployed, 596–603
browser-enabled, 573, 578–85
building, 615–16
core data source association, 570
data-blind controls, 639
data connections, 574–78
defined, 570–71
deployment, 586–96, 615–16
designing, 563, 567, 639
design once, 572–73
digital signature, 611–15
layout, creating, 579–80
parts, 573–74
preparing for administrator deployment, 595–96
provisioning, 599
published, 561
quiescing, 601–3
security, 611–15
site content type association, 600
unsigned fully trusted, 613
upgrading, 601–3
uploading, 596–98
uploading with ststadm.exe, 598
user deployment prevention, 561
Form Templates library, 598
FrontPage, 243, 317
Fully Qualified Domain Names (FQDN), 215–16
Functionality gaps, 221–22
Functionality matrix, 221–22

Gatherer, 118
`GetListCollection` method, 304, 306
`GetList` method, 304, 307
Ghosted pages, 386
Global Address List (GAL), 64
administrative control and, 66
contacts, 65
Global Assembly Cache (GAC), 343–44, 534
Global deployment, 211–13
distributed/regionalized servers, 212–13
multiple languages, 211–12
Global navigation, 406
GLOBAL site definition, 432, 434, 435, 504, 505
Granularity, 42–43
Groups
Administrators, 42
defined, 38
Farm Administrators, 42
MOSS, 71–72
uses, 39, 41
GUID Generator tool, 260, 261, 443

Hard limits, 200
`HeaderXslLink` property, 337
HelloWorld1.aspx file, 269
`HideCustomAction` element, 477–78
Hit highlighting, 136–37
Holds, 105–6
HTTP handler, 238

IFilters, 192
Imaging Web service, 300
Incoming e-mail, 58–62
Indexed documents, limit recommendation, 202
Indexing pipeline, 118
Indexing service, 192
InfoPath 2007, 184
compatibility checker, 242
defined, 557, 570
Document Information Panel, 603–7
introduction to, 570–74
Publishing Wizard, 587
use of, 570

Visual Studio integration capabilities, 608
.xsn files, 570
See also Forms; Form templates
InfoPath Forms Services, 8, 557–642
active/inactive sessions and, 566
availability, 560
configuration settings, 558–70, 560, 567
configuring, 560–70
defined, 558–59, 560
enabled, 561
Form view, 566, 567
postback requirement, 565
session state, 565–66
Session State Service, 566–67
workflow capabilities and, 559
Information management policies, 93–96
applying, 95–96
creating, 94
defining, 93–94
expiration, 95
rules, 95
uses, 93
Information Rights Management (IRM), 96–97
Inheritance, content types and, 31
`InheritSettings` class, 505
Input Method Editions (IMEs), 211
`InstallFeature` operation, 486–87, 496
Internet Information Services (IIS), 15
installation, 16
Web site support, 16
Investment Areas, 3
ISAPI filter, 237–38
Issue tracking list, 67
`IWebPartField` interface, 371, 372

Kerberos tokens, 186
Key performance indicators (KPIs), 9, 163, 169–71
characteristics, 169
creating, 170–71
displaying, 171
illustrated, 171
uses, 169
Web Parts, 171
Keywords, 131–32
defined, 131–32
searches, 129
at site definition level, 132
Knowledge Network, 151

Layouts branding, 273–75
LDAP queries, 146, 161–62
Libraries
accessing through OWA 2007, 185–87
blog content, 53
check-in/check-out support, 36
Data Source Library, 319
defined, 29
document, 56
enabling e-mail delivery to, 60–62
event exercise, 546–56
event handler registration, 544
Outlook 2007, 179–80
records management, 100
types, 29
user form template deployment to, 587–92
wiki, 47
WSS 2.0, 29
Limit recommendations, 200–203
hard limits, 200
for logical architecture objects, 202
search objects, 202–3
for WSS 3.0 components, 200–202
Line-of-business, 617
Linked data sources, 318, 320–21
Data Source Library, 320
data view creation with, 322–25
joining, 321
merging, 321
options, 320–21
See also Data sources
Links
adding, 292–93
deleting, 295
navigation, creating, 450
published, 301
List content type, 31
List Data Retrieval Web service, 300
`List` element, 451–52
ListFeatures.aspx, 288
ListFeatures.cs, 287
List fields console application, 288–89
List-level permissions, 39
Lists
adding items to, 291
BDC and, 87–88

blog content, 53
columns, 29
Data Source Library, 319
defined, 29
deleting items from, 294
discussion, 67
enabling, 31
enabling e-mail delivery to, 60–62
issue tracking, 67
Outlook 2007, 177–79
project task, 67
retrieving, 304–6
Web services code for adding, 304
WSS 2.0, 29
WSS 3.0 enhancements, 67
ListSites1.aspx, 272–73
ListSites3.aspx, 276–77
ListSites4.aspx, 280
ListSites4.cs, 279
`ListsOverride` property
defined, 328
defining list type, 333
defining specific list, 333
using, 333–34
Lists Web service, 300
adding references to, 303
working with, 304
List templates, 537–38
Look and feel. *See* Branding
`LstLists_SelectedIndexChanged` method, 306

`MakeCAB.exe`, 259–61, 496
Managed folders, 105
Managed properties, 127
Master page gallery
enterprise content management options, 389
master pages in, 393
Master pages, 108, 390–93
changing, 393
characteristics, 390–91
contents, 23
CSS script applied to, 409
custom application, 275
defined, 22–23, 390
elements, 390
inheriting, 275–77
masterpagefile attribute, 391
in master page gallery, 393
placeholder controls, 396
setting, 394
setting with browser user interface, 497
SharePoint, 393–99
test.master, 392
updates, 396
Meetings Web service, 300
Metadata, 29–32
content types and, 30
wiki pages with, 51
Methods
defined, 83
`SPItemEventReceiver` class, 520–21
`SPListEventReceiver` class, 528
`SPWebEventReceiver` class, 516
Web Part, 357–59
Microsoft Application Center Test, 194
Microsoft MSDN Web site, 396
Microsoft Office Forms Server 2007, 559
Microsoft Operations Manager (MOM), 231
Migrations. *See* Data migrations
Minimal master
applied to document center site, 398
HTML, 397
See also Master pages
Mobile access, 55–57
Modification forms, 633
Module elements, 452, 473–75
MOSS, 69–88
administration additions, 70–71
architecture, 239
authoring workflows in, 631–32
branding enhancements, 388–407
centralized records repository, 98
content query Web Part, 325–41
defined, 2
deployment hardware topology, 206–7
enterprise content management, 388–90
enterprise search capabilities, 241
extensions, 70–81
Features, 72–75, 464–66
groups, 71–72
high availability considerations, 207–8
Knowledge Network, 151
network impact, 208–9
object model, 296–98
permissions, 71–72
publishing infrastructure feature, 389–90

SDK, 241, 632
search administration, 121
search data storage, 141
Shared Services, 71
sizing, 203–9
solution areas, 4, 5, 70
SQL Server impact, 205–6
templates, 77–81
themes, 412
Web front-end scalability, 204–5
Web Parts, 75–76, 317–41
Web services, 301–2
workflow and, 616–23
MPS site definition, 432
My SharePoint Sites, 155, 184–85
defined, 155
navigating, 185
My Site, 153–55, 184
defined, 153
Host, 157
My Home, 153
My Links, 155
My Profile, 153
in SharePoint feature planning, 192–93
SSP association, 185

Namespaces, 285
naming, 347–48
`UserProfiles`, 296
`NavBar` element, 449–50
`NavBarPage` element, 453
Navigation, 26–28, 402–7
current, 406
default, 403
forms, 27
global, 406
horizontal, 406
inheritance, 27
links, creating, 450
options, 28
settings, 406
site map, 405
structure, 407
tree view control, 27, 28
vertical, 406
Network
impact, 208–9
tested capacity, 208
Nonautoritative sites, 131
Notepad, 260

Object model
accessing with inline code, 271–73
adding content, 290–93
registering event handlers with, 543–44
removing content, 293–95
removing event handler registrations with, 544–46
SharePoint data retrieval, 286–90
SharePoint Server 2007, 296–98
WSS 3.0, 284–86
Office Data Connection (ODC) file, 575
Official File Web service, 301
Offile site definition, 432
`OnDeleting` method, 357
ONET.XML file, 385, 434, 435
`AllUsersWebPart` element, 452
`Configurations` element, 450–54
dissecting, 448–54
`ExecuteUrl` element, 451
`File` element, 452
`List` element, 451–52
`Module` elements, 452
`NavBar` element, 449–50
`NavBarPage` element, 453
`Project` element, 448–49
`SiteFeatures` element, 454
structure illustration, 448
`View` element, 453–54
Operating System (OS), 225, 229
Operational model
four-tier, 225–26
illustrated, 226
Operating System (OS), 225
planning, 224–26
Operational team
staffing, 229–30
technical level requirements, 230
OSRV site definition, 432
Outlook 2007, 175–80
libraries, 179–80
lists, 177–79
Offline Documents, 180
SharePoint Drafts, 180
synchronization process, 176–77
Outlook Web Access (OWA) 2007

Kerberos tokens, 186
library access through, 185–87
proxy requests, 186

Packaging Web Parts, 375–78
Page layouts, 400–402
content type, 400
creating, 402
defined, 23, 400
page association with, 109
page instantiation from, 401
as template files, 402
Pages
application, 273
authoritative, 131
creating, 109–11
customizing, 475
editing, 111
ghosted, 386
properties, 112
resetting to site definition version, 387
uncustomizing, 475
unghosted, 386
Web Part, 157–58
Passive analyzing, 232
Pathway to production, 230–31
People, 143–62
People picker, 161–62
People Web service, 300
Performance monitoring, 232–35
counters, 234–35
future bottlenecks, 234
strategy, 233–34
Performance testing
methodology, 196–97
Web Parts, 194–95
Permissions
granular, 42–43
individual, 37, 38
list level, 39
MOSS, 71–72
site, 40–41
Permissions Web service, 300
Personalization site links, 159
Personalization site template, 159
Personal portal, 152–55
Pivot searches, 140
Portal, 6
collaboration, 78–81, 403, 407, 447
defined, 6
personal, 152–53
site definitions, 446
Postbacks, 563
Forms Services requirement, 565
settings, 564
PowerPoint 2007, 182
Product Support Services (PSS), 230–31
PROFILES site definition, 432
Programmability, 267–307
Project element, 448–49
Project task lists, 67
`Properties` element, 479
`PropertyConstants` class, 297
Protocol handlers, 117–18
Provision files, 460
PST files, 176–77
Public folders, permissions/roles, 546
Public key token, extracting, 349–50
Published Links Web service, 301
Publishing Excel workbooks, 165–66
Publishing infrastructure feature, 389–980
PublishingLayouts Feature, 464–66
Publishing page rendering, 402, 403
Publishing Portal template, 107
PUBLISHING site definition, 432
Publishing sites, 107–9
illustrated, 108
Pages document library, 109
Publishing Wizard, 587

`QueryOverride` property, 332–33
CAML query format, 332–33
defined, 328
illustrated, 333
`ViewFieldsOverride` property with, 334
See also Content query Web Part
Quiescing, 601–3
defined, 601
entire form, 604
form templates, 603
Quota management, 217

RACI charts, 228
Really Simple Syndication (RSS), 53–55
in blogging environments, 193

defined, 53
reader, 55, 194
in underlying SharePoint functionality, 193
See also RSS feeds
`Receivers` element, 537
Recipient Update Service (RUS), 64
Records management, 97–106
defined, 98
document libraries, 100
e-mail and records, 103–5
holds, 105–6
repository site, 99–101
Recycle bin, 33
Reghosting, 386
Regionalized server deployment, 212–13
Registration
custom site definition, 442
custom Web Parts, 351
event handlers, 534–36
event handlers, removing, 542
event handlers, removing with object model, 544–46
event handlers, to specific lists, 540
event handlers using content types, 540–42
event handlers using Features, 536–40
event handlers using object model, 543–44
site templates, 437–40
`RenderContents` method, 357–58
Rendering
publishing page, 402, 403
sites, 22–28
Web Part, 354–55
WSS 3.0 page, 399
`RenderWebParts` method, 358
ReportCenterCreation feature, 77
Repository
populating, 101–3
records site, 99–101
routing rules and, 102
Requests per second (RPS), 197
calculation elements, 198–99
calculation formula, 198
example calculation, 199
Reusable content, 110
Rights management, 96–97
RSS feeds, 54, 55
in planning SharePoint features, 193–94
subscription, 54
viewing, 55
RSS viewer Web Part, 76
`RunWithElevatedPrivileges` method, 295–96

Scale-out, 197, 205
Scale-up, 197, 205
Scopes, search, 123–236
defining, 124–25
illustrated, 125
rules, 124
site collection, 125, 126
Search, 6–7, 37, 115–42
administration, 121–22
anchor text, 119
best bets, 131–32
click distance, 119
comprehensive indexing, 120
configuring, 120–28
consuming, 128–42
content sources, 122–23
crawl rules, 123
crawl settings, 120
duplicate detection, 137, 138
experience goals, 119–20
features, 141–42
hit highlighting, 136–37
keywords, 131–32
managing, 120–22
mechanisms, 115
modified results, 136
property management, 126–28
query syntax, 129–31
relevance, 119
results page, 132–37
scope rules, 124
scopes, 123–26
site hit frequencies, 123
tabs, 137–39
taxonomies, 115
URL depth, 119
Web Parts, 132–37
Web service, 141
Search Center, 133–34
Search engine, 116–20
catalogs, 118
core, 116–19
filters, 118
gatherer, 118
goals, 119–20

indexing pipeline, 118
protocol handlers, 117–18
word breakers/stemmers, 118
Searches
data storage, 141
keyword, 129
pivot, 140
understanding, 140–41
URL, 130
Search indexes, 218
limit recommendation, 203
sizing, 210–11
Search Scopes, 191–92, 203
Search service, 192
Search Web service, 301, 302
Secondary data sources, 574
adding, 580–81
configuring, 640
converting to UDC files, 582
Securable objects, 38
Security
form level, 640
form templates, 611–15
validation error, 292
Security model, 37–44
authentication, 42
authorization, 37–42
granular permissions, 42–43
user interface, 43–44
See also WSS 3.0
Security-trimmed user interface, 43–44
Security trimming, 90
Server name mappings, 121
Server-side scripts, 319
Service accounts, 213–14
Service Level Agreements (SLAs)
key points, 226
planning, 226–28
priority levels, 227
RACI charts, 228
roles and responsibilities, 227–28
Shadow environments, 230
Shared Services Providers (SSPs), 70, 565
SharePoint
adding content to, 290–93
data retrieval, 286–90
defined, 2
development, 237–65
master pages, 393–99
namespaces, 285
programmability, 267–307
removing content from, 293–95
rights, 97
technology evolution, 9
as two parts, 11
user profile, 144–52
as Web application, 196
Web applications, 16
SharePoint deployment
authentication providers, 190–91
backup and restore, 218–19
database sizing, 210
data migrations, 219–22
disk subsystem planning, 209–11
enterprise search scoping, 191–92
features, 190–95
global, 211–13
MOSS sizing, 203–9
My Sites, 192–93
performance monitoring, 232–35
planning, 189–235
requirements, 189–90
RSS feeds, 193–94
rules, 189–90
search index sizing, 210–11
sizing, 195–203
software, 213–17
support planning, 222–32
user profiles, 192–93
Web Parts impact, 194–95
SharePoint Designer, 243–49
Apply Styles task pane, 247
authoring workflows with, 628–31
contributor mode, 243–44
contributor settings, 245
CSS Properties task pane, 247
data view Web Part, 317–18
defined, 243
environment illustration, 247
environment layout, 246
Manage Styles task pane, 247
task panes, 247–49
ToolBox task pane, 248
tools/components, 243–44
views, 248
workflow editor, 248–49, 629–31
SharePoint Portal Server 2003. *See* SPS 2003
SharePoint Server 2007. *See* MOSS

SharePoint Sites Web Part, 155
SharePoint Solution Generator, 250
`ShowDayHeader` property, 362, 363
`ShowGridLines` property, 362, 363
Simple versioning, 34
Site Collection Features, 467–68
Site collections
  activating form templates in, 600
  administrators, 41
  defined, 17
  illustrated, 17
  namespace, 18–19
  number of decision, 17
  property replication to, 150
  top-level site, 32
  URLs, 19
Site columns, 90
  defined, 32
  illustrated, 33
Site Data Web service, 300
Site definitions, 431–35
  BDR, 432
  BLANKINTERNET, 432
  BLANKINTERNETCONTAINER, 432
  Blog, 432
  branding exercise, 455–62
  CENTRALADMIN, 432
  custom, 385, 441, 442
  default, 431–34
  defined, 23, 430, 431
  GLOBAL, 432, 434, 435, 504, 505
  MPS, 432
  offile, 432
  OSRV, 432
  page versions, resetting to, 387
  populating, 449
  portal, 446
  PROFILES, 432
  PUBLISHING, 432
  solution files, 454–55
  SPS, 433
  SPSCOMMU, 433
  SPSMSITE, 433
  SPSMSITEHOST, 433
  SPSNEWS, 433
  SPSNHOME, 433
  SPSPERS, 433
  SPSREPORTCENTER, 433
  SPSSITES, 433
  SPSTOC, 433
  SPSTOPIC, 434
  SRCHCEN, 434
  SRCHCENTERLITE, 434
  STS, 434
  Wiki, 434
Site Features, 464
`SiteFeatures` element, 454
Site map providers, 404–5
SiteMap table, 20
  in configuration database, 21
  illustrated, 21
Site permissions, 40–41
Sites
  blog, 51–53
  branding, 383–427
  description, 420–21
  framework, 15–19
  hit frequencies, 123
  icon, 420–21
  navigating, 184–85
  provisioned, 461
  publishing, 107–9
  rendering, 22–28
  repository, 99–101
  sample hierarchy, 18
  snapshots, 23
  storage, 19–22
  title, 420–21
  wiki, 46–51
Sites Web service, 300
Site templates, 435–40
  custom, 441
  defined, 435
  registrations, 437–40
Smart client content authoring, 110
SMTP
  domains, 60, 61
  routing infrastructure, 59
  servers, 59, 60
  services, 61
Software deployment
  DNS architecture, 215–16
  preparing for, 213–17
  quota management, 217
  service accounts, 213–14
  SQL server configuration, 216–17
Solution areas, 3–5
  Collaboration, 13

SharePoint 2007, 5, 70
SPS 2003, 4
Solution Generator, 455, 456
Solutions
building with `makecab.exe`, 259–61
building with Visual Studio CAB project template, 261–62
building with Visual Studio extensions for WSS 3.0, 363–65
creating, 254
DDF contents and, 260
deployment process, 255
file composition, 258–59
lifecyle, 254
removing, 257
retracting, 256–57
site definition, 454–55
upgrading, 257
`SPEmailEventReceiver` class, 531–32
`SPEventReceiverBase` class, 515–16
`SPItemEventReceiver` class, 520–28
`AfterProperties` property, 523, 525, 526, 527
`BeforeProperties` property, 523, 525, 526
defined, 520
`ItemAdded` method, 520
`ItemAdding` method, 520, 522, 526
`ItemDeleting` method, 533, 534
`ListId` property, 523, 524
`ListItem` property, 523, 524
methods, 520–21
properties, 523–24
properties sample values, 525
See also Event receivers
`SPListEventReceiver` class, 528–31
defined, 528
`ErrorMessage` property, 530
`FieldAdding` method, 528
`FieldDeleting` method, 528
`Field` property, 529, 530
methods, 528
properties, 530
properties sample values, 531
See also Event receivers
`SPNavigationProvider`, 404
SPS 2003, 1, 11
branding pain points in, 384–88
customization capabilities, 385
limitations, 384
ONET.XML file, 385
pages and list schema, 386
solution areas, 3–4
SPSCOMMU site definition, 433
SPSMSITEHOST site definition, 433
SPSMSITE site definition, 433
SPSNEWS site definition, 433
SPSNHOME site definition, 433
SPSPERS site definition, 433
SPSPORTAL, 77
SPSREPORTCENTER site definition, 433
SPS site definition, 433
SPSSITES site definition, 433
SPSTOC site definition, 433
SPSTOPIC site definition, 434
SPTHEMES.XML file, 417–18
`SPWebEventReceiver` class, 516–20
defined, 516
methods, 516
properties, 518–19
properties sample values, 520
`WebDeleting` method, 516, 517
See also Event receivers
SQL, 129
cluster support, 207–8
connectivity, 231
embedded authentication, 563
in multiple instance mode, 208
Online Books, 232
services, 232
syntax queries, 129–30
SQL Server
configuration, 216–17
impact, 205–6
role, separating, 205
SRCHCEN site definition, 434
SRCHCENTERLITE site definition, 434
Staging process, 223
Stemmers, 118
Stemming, 129
Storage, 29–37
administration, 32–34
metadata, 29–32
search, 37
versioning, 34–36
Storage Area Networks (SANs), 217
.stp files, 445–46
Structured Query Language. *See* SQL
Stsadm utility, 19, 162, 257, 487, 598

STS site definition, 434, 436
Submit button, 639
Support planning, 222–32
  active monitoring, 231–32
  call volume, 228–29
  operational model, 224–26
  operational team, 229–30
  passive analyzing, 232
  pathway to production, 230–31
  SLAs, 226–28
  strategies, 222–24
  See also SharePoint deployment
Synchronization process, 176–77
Synchronous events, 514
  error message display capability, 517
  as processing events, 514
System master, 393

Table of contents Web Part, 76
Tables
  SiteMap, 20, 21
  Webs, 21
Taxonomies, 115
Team collaboration lists, 24
  contents, 24
  defined, 24
  example, 25
  illustrated, 25
  See also Lists
Team-process automation, 617
Templates
  application, 446–47
  collaboration portal, 78–81
  custom, 385
  form, 559, 561
  list, 537–38
  MOSS, 77–81
  personalization site, 159
  publishing site, 107–9
  report site, 171–73
  site, 435–40
  workflow, 620
Test.master master page, 392
Text boxes, 297
Themes, 411–20
  colliding, 416
  creating, 415–18
  folders, 415
  inconsistent branding with, 416
  multiple, using, 420
  setting, 413
  SharePoint Server 2007, 412
  use decision, 418–20
  See also Branding
Three-state workflow, 619–21
  association, 620
  configuration, 621
  defined, 619
  Feature representation, 622
  See also Workflow
Throughput estimation, 197–200
  number of hours per day, 199
  number of operations, 198–99
  number of users, 198
  peak factor, 199
  percent of active users, 198
  for WSS, 206
Tiered operational model, 225–26
Tool panes
  custom property presentation, 363, 364
  illustrated, 364
  interface, 362
Tree View, 27, 28, 407

Unghosted pages, 386
Uniform Resource Locators (URLs), 15–16
  searches, 130
  site collection, 19
  WSS handling, 16
`UninstallFeature` command, 491, 496
Universal Data Connection (UDC) files, 575, 576
  convert data connection to, 577
  use advantages, 575–76
Unstructured workflow, 617
URL depth, 119
User-deployed forms, 586
User form template deployment, 586–96
  to document library, 587–92
  options, 587
  to site content type, 592–94
  successful, 591
  See also Form templates
User Profile Change Web service, 302
`UserProfileManager` class, 297
User profiles, 144–52
  defined, 144

display and edit settings, 149
importing from AD, 144–45
limit recommendation, 202
multilingual considerations, 148
policy settings, 148
populating, 144–47
properties, 147–49
property import mapping, 149
replicable, 148–49
searching, 149–52
search settings, 149
in SharePoint feature planning, 192–93
working with, 297
`UserProfiles` namespace, 296
User Profile Web service, 302
Users
defined, 38
management, 39
per group limit recommendation, 201
Users and Groups Web service, 301

Variations, 110
Veritas Backup Exec, 219
Version histories, 35
Versioning, 34–36, 90
advanced, 34
check-in/check-out and, 35
content approval and, 35
simple, 34
Version number, 346
Versions Web service, 301
`View` element, 453–54
`ViewFieldsOverride` property
defined, 328
with `QueryOverride` property, 334
See also Content query Web Part
Views Web service, 301
Visual Studio 2005, 249–53
Deploy feature, 264, 265
extensions, 251
in SharePoint application development, 250
Tools for Office Second Edition (VSTO SE), 608–10
Workflow Designer, 252, 253, 625, 626
workflow project templates, 253, 625
Visual Studio Tools for Applications (VSTA), 608

Wave 12, 1–2
Web Application Features, 464
Web applications, 20, 280–84
`WebBrowserAttribute,` 365
Web Content Management (WCM), 106–13
content deployment, 111–13
content modification, 109–11
defined, 106
page creation, 109–11
publishing sites, 107–9
WebDAV, 58
Web front-end scalability, 204–5
Web Front End (WFE), 14
defined, 14
servers, 58, 59
Web Part connections, 366–73
interfaces, 370–73
`IWebPartField` interface, 371, 372
`IWebPartRow` interface, 371, 372
menu, 370
types, 370
Web Part gallery, 263
adding Web Parts to, 351–52
Closed, 317
Web Part provisioning in, 379
Web Part pages, 157–58
adding Web Parts to, 352–54
contents, 310
creating, 311–13
gallery, 315
version number use, 346
zone layout, 311
zones, 311, 312
Web Part Pages Web service, 301
Web Part properties, 316–17, 355–59
base, 316–17
content query Web Part, 328–29
custom, creating, 364–66, 367
default settings, 356
defining, 356
viewing with object browser, 357
Web Parts
adding, 313–15
ASP.NET 2.0, 310
BDC, 86–87
child controls, 359–61
coding, 354–66
content query, 76, 325–41
custom, creating, 341–54

data view, 317–18
data view formats, 317
debugging, 374
defined, 309–10
definition, uploading, 315
deploying, 378–81
deployment process, 223, 351–54
impact in SharePoint deployment, 194–95
KPI, 171
menu, 361–62
methods, 355–59
modifying, 315–17
MOSS, 75–76, 317–41
packaging, 375–78
per page limit recommendation, 202
placement, 309
provisioning feature, activating, 379
provisioning in Web Part gallery, 379
registering, 351
removing, 317
rendering process, 354–55
RSS viewer, 76
search, 132–37
solution file creation, 259–61
solution file deployment, 256
table of contents, 76
title, modifying, 316
tool panes, 362–66
.webpart file extraction, 376
in wiki pages, 49–50
zones, 109
See also specific Web Parts
.Webpart files, 261–62
WebReady Document Viewing Capability, 186
Web services
Administration, 299
Alerts, 299
Authentication, 299
Copy, 300
Document Workspace, 300
Forms, 300
Imaging, 300
infrastructure, 299
levels, 297
List Data Retrieval, 300
Lists, 300
Meetings, 300
Official File, 301
People, 300
Permissions, 300
Published Links, 301
Search, 301, 302
SharePoint Server 2007, 301–2
Site Data, 300
Sites, 300
User Profile, 302
User Profile Change, 302
Users and Groups, 301
Versions, 301
Views, 301
Web Part Pages, 301
Webs, 301
Workflow, 302
working with, 302–7
WSS 3.0, 299–301
XML, 319–20
`WebsOverride` property
defined, 328
illustrated, 333
using, 333
See also Content query Web Part
Webs table, 21
Webs Web service, 301
WEBTEMP*.XML files, 435, 441, 455
Wiki pages, 46–51
creating, 49
defined, 46
in document library, 47
with metadata, 51
versioning on, 13
Web Parts in, 49–50
Wiki site definition, 434
Windows Communication Foundation (WCF), 240–41, 618
Windows SharePoint Services 2.0. *See* WSS 2.0
Windows SharePoint Services 3.0. *See* WSS 3.0
Windows Workflow Foundation (WWF), 240–41, 618
`CreateActivity` activity, 618
runtime engine, 618
`SendEmail` activity, 618, 619
`TerminateActivity` activity, 618
Word 2007, 180–82
document information panel, 182
offline editing, 180–81
version comparison, 181
Word breakers, 118
Workflow Designer, 625, 626, 630

Workflow editor, 629–31
Workflows, 616–23
  activation, 621
  ad-hoc, 617
  authoring, 623–32
  automation, 616–17
  in browser, 631
  configuring, 629
  creating, 621–22
  element manifest, 628
  extranets, 617
  Feature.xml, 626, 627
  form integration with, 632–41
  line-of-business, 617
  predefined, 622, 623–24
  project folder, publishing forms to, 640–41
  settings, 25, 26
  in SharePoint Server 2007, 622–23
  team-process automation, 617
  templates, 620
  three-state, 619–21
  unstructured, 617
  WCF, 618
  in WSS 3.0, 618–22
Workflow Web service, 302
WSS 2.0, 1, 11
  design implementation, 237
  events support, 532
  lists and libraries, 29
WSS 3.0, 11–44
  3-tier architecture, 13–15
  architectural ingredients, 13–22
  authoring workflows in, 624–28
  component limit recommendations, 200–202
  defined, 2, 12
  deployment, 15
  e-mail and, 58–67
  event architecture, 514
  MOSS extensions to, 70–81
  navigation, 26–28
  object model, 284–86
  page rendering, 399
  platform services, 12
  required throughput formula, 206
  sample topologies, 15
  SDK, 294, 430
  search administration, 122
  security model, 37–44
  site framework, 15–19
  site rendering, 22–28
  site storage, 19–22
  solution deployment process, 255
  solution framework, 253–65
  storage, 29–37
  three-state workflow, 619–21
  Web services, 299–301
  workflow in, 618–22

XML
  files, 319
  returned by service, 304, 305
  returned properties, 135
  Web services, 319–20
`XMLDocument` elements, 540