Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™



Professional Sitecore Development

John West

www.it-ebooks.info

www.it-ebooks.info

PROFESSIONAL SITECORE® DEVELOPMENT

INTRODUCTION xxix

CHAPTER 1	Introducing the Sitecore ASP.NET CMS	1
CHAPTER 2	Information Architecture	19
CHAPTER 3	The Layout Engine and Data Access APIs	73
CHAPTER 4	Sitecore Security	123
CHAPTER 5	Error Management	161
CHAPTER 6	Optimizing, Scaling, and Troubleshooting	197
CHAPTER 7	Extending and Integrating Sitecore	245
CHAPTER 8	Automated Testing	307
CHAPTER 9	Managing Implementations	353
CHAPTER 10	On Beyond CMS	381
CHAPTER 11	Sitecore Best Practices, Tips, and Tricks	401
APPENDIX A	Resources for Sitecore Developers	443
APPENDIX B	Installing Sitecore	451

INDEX **493**

www.it-ebooks.info

www.it-ebooks.info

PROFESSIONAL

Sitecore® Development

www.it-ebooks.info

www.it-ebooks.info

PROFESSIONAL

Sitecore® Development

John West



John Wiley & Sons, Inc.

www.it-ebooks.info

Professional Sitecore® Development

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2012 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-93901-7

ISBN: 978-1-118-22148-8 (ebk)
ISBN: 978-1-118-23525-6 (ebk)
ISBN: 978-1-118-25997-9 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

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, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at www.wiley.com/go/permissions.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or website is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or website may provide or recommendations it may make. Further, readers should be aware that Internet websites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats and by print-on-demand. Not all content that is available in standard print versions of this book may appear or be packaged in all book formats. If you have purchased a version of this book that did not include media that is referenced by or accompanies a standard print version, you may request this media by visiting http://booksupport.wiley.com. For more information about Wiley products, visit us at www.wiley.com.

Library of Congress Control Number: 2011930308

Trademarks: Wiley, the Wiley logo, Wrox, the Wrox logo, and Wrox Programmer to Programmer are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Sitecore is a registered trademark of Sitecore Corporation. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

www.it-ebooks.info

I dedicate this book to the thousands of internal and external Sitecore developers, project managers, technical writers, sales staff, support engineers, instructors and instructional material managers, and everyone else in the Sitecore community working around the clock and around the world to contribute to the success of Sitecore initiatives on a daily basis.

www.it-ebooks.info

www.it-ebooks.info

ABOUT THE AUTHOR

JOHN WEST works as Chief Technology Officer (CTO) for Sitecore in North America. Between the year 2000 and the time he joined Sitecore in 2004, John consulted on approximately twenty significant web content management system (CMS) implementations including Network Associates (now McAfee), U.S. Bank, John Deere, Stanford Hospitals and Clinics, and Intel. During that time, he taught courses on CMS products in the United States, Europe, and Asia.

While researching the CMS market to replace a legacy CMS, John identified Sitecore as the leading CMS platform architecture available, and joined the company immediately after attending developer training. Initially the only technical employee in North America, John certified hundreds of Sitecore developers while building the foundations of the local technical team including IT (information technology) platform services, presales technical engineering, solution prototypes, documentation, and support, all while working to promote a vibrant Sitecore developer community.

John wrote early drafts for a majority of the current technical documentation about the Sitecore CMS. Including his input on more than six thousand posts on the Sitecore Developer Network (`http://sdn.sitecore.net`), John has interacted with thousands of web developers working on untold CMS projects around the globe. John has a strong interest in programmer productivity and is passionate about helping Sitecore developers learn the product to overcome any challenges.

Although his current responsibilities are primarily those of a consulting product architect on products including the Sitecore ASP.NET Content Management System (CMS), Digital Marketing System (DMS), and Sitecore Azure for deploying Sitecore solutions to the Microsoft Windows Azure cloud, John considers his primary responsibility to be that of Chief Customer Advocate. You can access his blog about Sitecore at `http://sitecorejohn.com` and follow him on Twitter (`http://twitter.com`) as `@sitecorejohn`.

www.it-ebooks.info

www.it-ebooks.info

ABOUT THE TECHNICAL EDITORS

CHRISTIE DENSMORE is a senior software developer who has been integrating Sitecore into .NET websites for several years. Her experience in software architecture, custom development, and Sitecore integrations covers a broad range of applications including global multi-lingual sites and Sitecore's e-commerce software. Christie is based in Atlanta, GA and graduated from Georgia Tech with a Bachelor of Science in Industrial Engineering.

LEWANNA RATHBUN is a Solutions Engineer for Sitecore in North America. In what she describes as her dream job, she works as part of the technical services team that assists Sitecore clients with installation, upgrades, site reviews, custom solutions, and other technical projects. Lewanna began developing with Sitecore CMS 5 in 2006 for a Fortune 100 client. Subsequent experience with other CMS products helped to secure a place for Sitecore in Lewanna's heart. Despite some setbacks during the project, she still claims to be "grateful for the privilege of being involved with this book," and wishes to thank all the amazing Sitecore developers who pioneered the way before her.

www.it-ebooks.info

www.it-ebooks.info

CREDITS

ACQUISITIONS EDITOR
Carol Long

PROJECT EDITOR
Sydney Argenta

TECHNICAL EDITORS
Christie Densmore
Lewanna Rathbun

SENIOR PRODUCTION EDITOR
Debra Banninger

COPY EDITOR
Luann Rouff

EDITORIAL MANAGER
Mary Beth Wakefield

FREELANCER EDITORIAL MANAGER
Rosemarie Graham

ASSOCIATE DIRECTOR OF MARKETING
David Mayhew

MARKETING MANAGER
Ashley Zurcher

BUSINESS MANAGER
Amy Knies

PRODUCTION MANAGER
Tim Tate

VICE PRESIDENT AND EXECUTIVE GROUP PUBLISHER
Richard Swadley

VICE PRESIDENT AND EXECUTIVE PUBLISHER
Neil Edde

ASSOCIATE PUBLISHER
Jim Minatel

PROJECT COORDINATOR, COVER
Katie Crocker

PROOFREADER
Jen Larsen, Word One

INDEXER
Robert Swanson

COVER IMAGE
© iStock / Kemter

COVER DESIGNER
LeAndra Young

www.it-ebooks.info

www.it-ebooks.info

ACKNOWLEDGMENTS

SEVERAL PEOPLE WORKED IN A VARIETY of roles to improve the quality of my work on this project. Before thanking them, I would like to show some appreciation for Bjarne Hansen and Kerry Bellerose of Sitecore for allocating my time to this project (honestly, Kerry, the publisher introduced almost all of the passive voice in this book). I also have Derek Roberti of Sitecore to thank for helping me maintain sanity through some of the more difficult phases of this project. Adam Conn, Suzy McKee, and Darren Guarnaccia of Sitecore provided most of the content for the chapter about Sitecore products other than CMS (Sitecore's web Content Management System). Of course, I can never thank Ivan Sharamock and Alex Shyba of Sitecore enough for their invaluable assistance with this and countless other Sitecore initiatives.

John Mueller (`http://blog.johnmuellerbooks.com`) rewrote almost every chapter, and provided more advice than I could possibly find time to apply during the timeline allowed by this project. I also enjoyed working with Paul Reese, Sydney Jones Argenta, and Carol Long at Wiley.

In writing this book, I solicited input from a number of seasoned Sitecore developers working for established Sitecore partners as described in the following paragraphs.

Klaus Petersen is a partner at the Danish software consultancy and Sitecore partner Alpha Solutions. Since beginning professional work with Sitecore in 2003, Klaus has been part of numerous large and extremely large Sitecore installations, building some of the most significant content management solutions in Sitecore CMS versions 4, 5, and 6.

Klaus has been a Sitecore MVP (Most Valued Professional) since 2008. Since 2005, Klaus has worked to support internal development efforts at Sitecore as part of the Sitecore Expert Team that consists of consultants and solutions partners who help the company to define how the platform can best support emerging customer objectives. The core engineering team at Sitecore continues to spar with Klaus about various aspects of the Sitecore platform.

Alpha Solutions offers project-oriented information technology development with an emphasis on providing complete solutions for its customers. Alpha Solutions focuses on the process-related elements of such projects, with competence that spans project leadership, strategic initiatives, architecture, modeling, development, and testing. Alpha Solutions is certain that the most effective strategy involves sharp focus, and maintains that goal with its 25 employees intent upon being amongst the best in their particular fields. Customers often select Alpha Solutions for the largest and most visible Sitecore projects in the world, especially in relation to search-driven solutions.

Alpha Solutions has integrated Sitecore with a number of large-scale search platforms including Microsoft FAST, Exalead, Google, Apache Solr, and a number of smaller Nordic search vendors. The results include solutions such as `http://krak.dk`, `http://edbpriser.dk`, `http://boligportalen.dk`, `http://finanstilsynet.dk`, `http://ug.dk`, `http://br.dk`, `http://toysrus.dk`, `www.jobs24.co.uk`, and `www.jobschampion.com`, as well as more than 50 additional large-scale Sitecore implementations.

www.it-ebooks.info

Alpha Solutions works with Sitecore customers in both the private and public sectors. The implementation timeline for typical Sitecore projects with Alpha Solutions ranges between 600 and 10,000 hours. Through its experience with Sitecore, the organization has created processes and developed knowledge regarding the control of extremely scalable and technologically complex projects.

Hedgehog Development, LLC is a recognized leader in Sitecore web content management implementations, custom application development, and integration services. The privately held company headquartered in Holbrook NY, serves clients around the world, ranging from global giants, to Fortune 500, to small, local companies.

Hedgehog's specialties include .NET development and Visual Studio Team System (VSTS), SQL server, cloud services development, and SharePoint development and implementations. The company's developers also have considerable experience in building large-scale applications that aid customers in making their deployment process run quickly and easily.

Built on the philosophy "If you can dream it, we can make it happen," Hedgehog has a long record of accomplishment in undertaking the toughest online challenges from clients and providing the most efficient customized solutions. Hedgehog's industry experience, insights, and attention to detail have earned the organization a solid reputation as the go-to company for developers working in Sitecore Content Management System and Microsoft .NET, two of the world's most important and robust development platforms.

Founders Daniel Galvez and Peter C. Madsen, who have worked together professionally for 15 years, serve as Hedgehog's core management team. The extended team of highly experienced, senior-level developers includes Charles Turano, who is also a partner, and Sean Kearney, who in 2009 became a Sitecore MVP. The company's well-known and highly successful Team Development for Sitecore product (TDS — `http://bit.ly/xIx33R`) provides a best-practice solution for the development of Sitecore-based web sites. TDS works as a plug-in for the Microsoft Visual Studio programming environment that simplifies the process of adding Sitecore items to a source control system and streamlines the way individuals and teams work together to develop Sitecore solutions. In 2011, the TDS product earned Hedgehog the coveted position of Technology Partner in the Sitecore Partner program, making Hedgehog one of only three worldwide companies partnered as both a Sitecore Implementation and Sitecore Technology partner.

In 2010, Hedgehog's client The Knot received the North American Sitecore Site of the Year award and the North American Sitecore Media Site of the Year award. For more information about Hedgehog Development, visit `www.hhogdev.com`.

Alistair Deneys is a Sitecore MVP who has worked with Sitecore for six years. Alistair works as a Solution Architect for Next Digital in Melbourne, Australia, where he designs and delivers solutions built using Sitecore. Next Digital is the longest-standing Sitecore partner in Australia, having introduced Sitecore as one of its offerings in 2004.

Prior to immersing himself in everything Sitecore, Alistair had a fascination with XSLT (eXtensible Stylesheet Language Transformations) and wrote a simple CMS using XML, XSLT, and the Apache ANT build engine (`http://ant.apache.org`) to output static HTML pages. He used this basic CMS to manage several smaller websites.

www.it-ebooks.info

Alistair cut his Sitecore teeth on the first .NET version of the CMS, Sitecore 4. Since then, he has worked on many different Sitecore projects of varying size and complexity.

During the initial stages of Alistair's use of Sitecore, he engaged in as many training courses as he could schedule, and eventually ended up running the official Sitecore training program for the Australian region under the direction of the newly established Sitecore Australia office.

Alistair has blogged about Sitecore for over three years, with posts covering a wide range of topics such as extending the CMS, best practices, and testing techniques. He actively contributes to numerous Sitecore Shared Source projects, most importantly the WeBlog blogging module (previously EviBlog; see `http://bit.ly/lIIuGb`). He wrote the first batch-processing module for Sitecore, which he named Revolver (`http://bit.ly/7GgU5Z`). Revolver provides a command prompt within the Sitecore desktop to allow developers and administrators to script repetitive tasks with ease.

Alistair did not follow a traditional computer science background, but instead received a Bachelor of Engineering degree from Swinburne University of Technology in robotics and mechatronics. He began programming computers during primary school, starting with GW Basic and progressing through Pascal, C, C++, Java, PHP, SQL, JavaScript, and C#.

Nick Laidlaw is Chief Technology Officer at Agency Oasis with direct experience in a variety of medium and large Sitecore implementations. Since 1997, Nick has worked extensively with content management systems that range from custom builds to enterprise-scale solutions. Oasis clients can utilize Nick's expertise when looking for content management system recommendations, analyzing deployment requirements for new solutions, and converting existing systems. In 2010, Nick earned the Sitecore MVP award. Along with his other responsibilities, Nick continues his activity within the Sitecore development community.

Oasis is a full service digital marketing agency that specializes in interactive marketing campaigns, website design and development, and the integration of digital marketing technology platforms such as Sitecore CMS. Founded in 2001, Oasis maintains its headquarters in Boston with offices in New York and San Francisco. The firm works with a robust roster of global brands to support enterprise-level websites and complex digital marketing campaigns that require international reach.

Oasis was one of the first Sitecore integration partners in North America and continues to be among the top Sitecore partners in terms of completed implementations in that region. With two Sitecore MVPs on staff and a deep bench of Sitecore Certified developers, Sitecore customers often call upon Oasis to implement sophisticated and complex Sitecore assignments. The firm's experience with global brands has lent itself to work on enterprise-level deployments of over 100 websites controlled from a single centralized Sitecore platform. Additionally, Oasis has extensive experience with international deployments that include web properties in as many as 15 different languages.

Oasis has received multiple technology and creative awards including recognition from the Horizon Interactive Awards, the Massachusetts Innovation & Technology Exchange (MITX) Awards, and Hatch Awards. Additionally, *Boston Business Journal* has consistently recognized Oasis as one of Boston's "Best Places to Work."

www.it-ebooks.info

www.it-ebooks.info

CONTENTS

INTRODUCTION *xxix*

CHAPTER 1: INTRODUCING THE SITECORE ASP.NET CMS **1**

Understanding Web Content Management Systems **2**
Introducing Sitecore **3**
Logging In 4
Embedded Applications 10
Working with Sitecore 12
Sitecore Company History 13
Sitecore ASP.NET CMS Version History 13
Sitecore Components 14
Sitecore Databases 14
Hosting IIS Websites 15
IIS Application Pool 16
Document Root Subdirectory 16
Data Subdirectory 16
Sitecore Visual Studio Project 17
Sitecore Rocks 17
Take Home Points **17**

CHAPTER 2: INFORMATION ARCHITECTURE **19**

Introducing Information Architecture **20**
Sitecore Items **22**
Item Languages, Versions, and Revisions 23
Item Properties 26
Item Structure 28
Definition Items 29
Insert Options 30
Sorting Items 31
Managing ItemURLs 33
How Sitecore Generates URLs 34
How Sitecore Resolves URLs 35
Removing /sitecore/content from URLs 36
Clone Items 36
Alias Items 38
Wildcard Items 39

www.it-ebooks.info

Sitecore Data Templates **39**
Data Template Sections 41
Data Template Fields 42
Data Template Field Types and Categories 42
Data Template Field Properties 46
Standard Values 49
Data Template Inheritance 53
The Standard Template 53
Data Validation **55**
Managing Multilingual Solutions **56**
Language and Culture 57
Determining the Context Language 58
Falling Back to an Alternate Language 59
Managing Media **60**
Sitecore Media Folders 61
Media URLs 61
Configuring the Media Library 64
Media Upload Watcher 65
WebDAV 66
Media Types 66
Configuring MIME Types 68
Media Path Provider 68
Caching Media 69
Sharing Content **69**
Importing Content **70**
Take Home Points **71**

CHAPTER 3: THE LAYOUT ENGINE AND DATA ACCESS APIS **73**

The ASP.NET Page Lifecycle **74**
The Sitecore Context **77**
Layout Details and Layout Deltas **78**
Presentation Components **78**
Devices 78
Layouts 79
Placeholders and Placeholder Settings 80
Presentation Controls (Renderings) 81
Sublayouts 81
XSL Renderings 82
Method Renderings 82

www.it-ebooks.info

URL Renderings 82
Web Controls 83
Rendering Parameters and Rendering Parameters Templates 84
Presentation Control Definition Item Properties 88
Choosing Presentation Technology 90
Presentation Component Data Sources 90
Conditional Rendering and Personalization 91
The FieldRenderer Web Control and the renderField Pipeline 98
The Page Mode 98
Fundamental Sitecore APIs 99
Sitecore.Configuration.Factory 99
Sitecore.Data.Database 100
Sitecore.Data.Items.Item 101
Sitecore.Data.Fields.Field and Related Classes 103
Sitecore.Data.ID 104
Sitecore.Data.Items.EditContext, BeginEdit(), and EndEdit() 104
Sitecore.Data.Items.CustomItem 105
Sitecore ID Constants 106
Sitecore Data Access Techniques 107
Direct Item API Access 107
Sitecore Query Access 108
Sitecore Fast Query Access 108
Lucene Search Index Access 109
Internal Links Database Access 113
Recursion 116
Sitecore Web Services 118
Syndicate Content with Sitecore 118
Public RSS Feeds 118
Client RSS Feeds 120
Layout Engine Best Practices 120
Take Home Points 121

CHAPTER 4: SITECORE SECURITY 123

Access Rights 124
Inheriting Access Rights 127
Evaluating Access Rights 127
Setting Access Rights 130
Interpreting Access Rights with the Access Viewer 131

www.it-ebooks.info

Securing Data Template Fields 133
Applying Access Rights with APIs 137
Security Accounts and Domains 140
Users 140
User Profiles 143
Anonymous Users 145
Context User 146
Roles 146
Everyone Roles 150
Nested Roles 151
Sitecore Client Roles 152
Security Domains 152
Preventing Sitecore from Applying Security 153
Impersonating a Sitecore User 153
Disabling Sitecore Security 154
Requiring Authentication for a Managed Website 154
Integrating and Extending Sitecore Security 155
Configuring ASP.NET Membership, Profile, and Role Providers 156
Building Virtual Users 156
Employing Federated Authentication 157
Switching Providers 157
Take Home Points 159

CHAPTER 5: ERROR MANAGEMENT 161

Exception Levels 163
Designing an Error Helper Class 163
Implementing Your Own Exception Classes 167
Trapping Exceptions with try...catch...finally Blocks 168
Handing Errors in Presentation Controls 169
Handing Syntax and Runtime Errors in XSL Renderings 169
Handling Exceptions in Sublayouts 174
Handling Exceptions in Web Controls 180
Hiding Presentation Control Errors 184
Handling Exceptions at the Page Level 185
Handling Exceptions at the Application Level 186
Error Management Pages 187
ASP.NET Error Management Pages 187
IIS Error Management Pages 189
HTTP 404 Not Found 189
System Outages 196
Take Home Points 196

www.it-ebooks.info

CHAPTER 6: OPTIMIZING, SCALING, AND TROUBLESHOOTING 197

Optimizing Sitecore Performance 198
Leveraging Sitecore Caching 198
Utility Caches 199
Database Caches 199
Site Media Caches 204
Site Output Caches 205
Tuning Cache Sizes 211
Disabling Cache Size Limits 214
Bringing the Most Bits Together 214
Scaling Your Solution 216
Scaling Infrastructure 217
Load Balancing 218
Balancing Content Management 218
Balancing Content Delivery 220
Scaling Publishing 220
Scaling the Database Layer 221
Virtualizing Sitecore 222
Edge Caching and Content Delivery Networks 222
Architecting High Availability 223
Planning Hardware Capacity 224
Troubleshooting Your Solution 225
The Sitecore Log Files 225
Rendering Statistics 227
Tracking Performance Thresholds 229
Debugging in a Browser 229
Debugging with Microsoft Visual Studio 233
Spelunking Sitecore 234
The Web.Config File 234
Web.Config Include Files 235
The /configuration/sitecore Section 236
Reflecting on Sitecore 238
Reviewing Stack Traces 242
Browsing Sitecore Databases 242
Take Home Points 243

CHAPTER 7: EXTENDING AND INTEGRATING SITECORE 245

Determining Types with the Configuration Factory 247
Extending Sitecore Classes with Extension Methods 249

www.it-ebooks.info

Leveraging the Sitecore User Interface Framework **250**
Introducing Sitecore Commands 250
Extending Ribbons 251
Adding Entries to Item Context Menus 259
Using Content Editor Warnings 260
Overriding Sitecore User Interface Applications 262
Engaging the Rules Engine **266**
Validating Data **270**
Validating Fields 271
Validating Items 274
Scheduling Sitecore Processes **277**
Defining Agents in the Web.config File 278
Registering Tasks in a Sitecore Database 280
Integrating from the Back End **282**
Initializing with Hooks 282
Handling Events 282
Processing Pipelines 287
Handing the httpRequestBegin Pipeline 289
Processing the renderField Pipeline 293
Intercepting the renderLayout Pipeline 296
DMS Pipelines 298
Hook, Pipeline Processor, Event Handler, or Rule? 299
Extending the Sitecore Page Editor **299**
Take Home Points **306**

CHAPTER 8: AUTOMATED TESTING **307**

Understanding Automated Testing **308**
Complexities of Testing Against Sitecore **309**
Testing Techniques for Sitecore **310**
The Test Project **311**
Creating the Test Project 311
Running the Tests 315
Testing with the HTTP Protocol **317**
Implementing Tests over HTTP 318
Limitations of Testing over HTTP 322
Testing Using a Web Browser Driver **322**
Implementing Testing with a Web Browser Driver 323
Limitations of the Web Browser Driver Technique 325
Testing with an Embedded Test Runner **325**
Implementing an Embedded Test Runner 326

www.it-ebooks.info

Deploying the Embedded Test Runner 333
Instantiating Controls to Test 334
Implementing Tests that Instantiate Controls 334
Limitations of Testing by Instantiating Controls 335
Invoking the Sitecore API Directly 336
Using Sitecore APIs without an HTTP Context 338
Considering Calling Sitecore APIs Without an HTTP Context 338
Limitations of using Sitecore API outside an HttpContext 342
Working with Test Data 342
Location of Test Data 343
Creating Test Data through HTTP 344
Creating Test Data through the API 347
Creating Test Data from XML 348
Take Home Points 351

CHAPTER 9: MANAGING IMPLEMENTATIONS 353

Approaching Sitecore Projects 354
Prototyping with Sitecore and RAD 354
Prototyping with Traditional RAD 355
Prototyping with Sitecore RAD 355
Facilitating Object-Oriented Coding Techniques 358
Integrating Front-End Markup 359
Integrating Custom Business Logic/Objects 359
Publishing with Sitecore 360
Publishing Modes 362
Republishing 362
Incremental Publishing 362
Smart Publishing 362
Publishing Restrictions 363
Publishing Targets 364
Publishing to Preproduction Sitecore Environments 365
Publishing Media 366
Scheduling Publication 366
The publishItem Pipeline 367
Workflow 367
The Sitecore Workbox 367
Locking and Workflow 368
Workflows 369
Workflow States 369

www.it-ebooks.info

Initial Workflow State 369
Final Workflow States 370
Workflow Commands 370
Workflow Actions 370
Managing Sitecore Deployments 372
Sitecore Item Serialization 372
Creating a Deployment 373
Deploying to the QA Environment 373
Deploying Additional Changes 374
Deploying to the Production Environments 374
Upgrading Sitecore 374
Team Development for Sitecore 375
Managing Multiple Sites with Sitecore 375
Take Home Points 379

CHAPTER 10: ON BEYOND CMS 381

The Digital Marketing System 382
Engagement Analytics 385
Engagement Automation 387
Testing and Optimization 387
Visitor Profiling 388
Real-Time Personalization 389
Universal Profile Management 390
Campaign Management 390
Dynamic Segmentation 391
Web Forms for Marketers 392
Email Campaign Manager 392
Standalone Sitecore Products 393
Sitecore Intranet Portal 393
Sitecore Foundry 394
Sitecore App Center 394
Optional Modules 395
Active Directory 395
Adaptive Print Studio 395
Calendar 396
dtSearch 396
E-Commerce Modules 397
SharePoint Integration Framework 397

www.it-ebooks.info

Sitecore Search Engine Optimization Toolkit 397
Sitecore Azure 398
Take Home Points 399

CHAPTER 11: SITECORE BEST PRACTICES, TIPS, AND TRICKS 401

Sitecore Best Practices 402
Managing Sitecore Implementations 403
Estimating Hardware Requirements 404
Administering Sitecore Solutions 405
Architecting Data Infrastructure 406
Designing Data Templates 408
Applying Standard Values 411
Configuring Insert Options 411
Managing Multiple Sites 411
Working with Multiple Languages 411
Storing Relational Data 413
Coding Presentation Infrastructure 414
Automating Publishing Workflow 417
Securing Sitecore Solutions 418
Using the Media Library 420
Maximizing Solution Performance 423
Sitecore Tips and Tricks 424
Optimizing Sitecore Usability 424
Logging In to a Sitecore User Interface 425
Limiting User Interfaces 425
Optimizing the Sitecore Desktop Experience 427
Ctrl+Right-Click 429
Optimizing the Content Editor 430
Maximizing Content Editor Performance 433
Content Editor Keyboard Shortcuts 433
Investigating and Copying Raw Field Values 434
Copying and Moving Items with the Clipboard 435
Sitecore Keyboard Shortcuts 435
Optimizing the Rich Text Editor 437
Rich Text Editor Profiles 438
Rich Text Editor Keyboard Shortcuts 438
Ad Hoc Reports 439
Overlay Icons 439

www.it-ebooks.info

Showing the Developer Tab 441
Working with the Sitecore Debugger 441
Take Home Points 441

APPENDIX A: RESOURCES FOR SITECORE DEVELOPERS 443

Accessing Sitecore Resources 443
Sitecore Training 444
Sitecore Partners 444
Becoming a Sitecore Partner 444
Benefiting from Your Sitecore Partnership 444
Sitecore Portal 445
Registering for a Sitecore Account 445
Signing Up for E-Mail Alerts 445
Resetting Your Sitecore Password 445
Sitecore Developer Network 446
Sitecore Documentation 446
Using the Sitecore Developer Network Forums 447
Accessing the Sitecore Shared Source Library 447
Sitecore Blogs 447
Sitecore Demo Sites 447
Virtual and Local Sitecore User Groups 448
The Sitecore Support Portal 448
Sitecore Symposium 449
Sitecore Success Services 449

APPENDIX B: INSTALLING SITECORE 451

Preparing to Install Sitecore 451
Choosing an Installation Strategy 452
Installing Microsoft SQL Server 454
Configuring an Existing SQL Server Instance 458
Installing Sitecore with the Setup Program 459
Performing a Complete Installation 461
Performing a Database Only Installation 464
Performing a Client Only Installation 465
Installing Sitecore from a Zip or Other Archive 466
Archiving a Sitecore Solution 467
Creating and Configuring a New Installation 469
Creating a Visual Studio Project 484
Take Home Points 491

INDEX 493

www.it-ebooks.info

INTRODUCTION

RECENT YEARS HAVE SEEN DRAMATIC EXPANSION in the use of mobile computing and social media on the Internet. Organizations show increasing interest in web analytics, and will soon comprehend the importance of engagement and experience management and monitoring. With the increasing value of Search Engine Optimization (SEO) and the nature of the web as a crucial channel for customer service, sales, and other essential interactive business functions, and especially with the advent of the Sitecore Digital Marketing System (DMS) and Customer Engagement Platform (CEP), these continue to be exciting and lucrative times with tremendous opportunities for Sitecore developers. If you already understand web development, ASP.NET, and the purposes of Content Management Systems (CMS), Sitecore is probably the tool for which you have been looking. If you are not familiar with these underlying technologies, examining Sitecore is a great way to learn best practices in web development.

This book provides material to increase the capabilities of developers working with the Sitecore ASP.NET web CMS. To help developers new to the Sitecore CMS achieve maximum productivity and satisfaction with minimal effort, the primary focus of this text includes information architecture, presentation, and extension of the Sitecore CMS. It also includes a chapter about tips, tricks, and best practices for working with the platform.

I realize that this book is an anachronism: a *book* about a *web* Content Management System. At the same time, a single, comprehensive (though far from complete) resource in book format could benefit developers new to the Sitecore CMS. Most of the content comes from my own experience and research over 15 years of web development. During that time, I spent over seven years working for Sitecore, before which I spent four years leading projects on other CMS platforms.

This book is a 1.0, meaning I hope to write at least a second edition (probably around Sitecore 7) with more useful information. I compiled most of the information in this book from resources that have been available for some time. Experienced Sitecore developers may recognize that some of the ideas and even content in this book has previously appeared in Sitecore training and documentation materials, my blog, and other writings. Even so, I hope that you will find useful the consolidation of these ideas presented previously along with new content under a single cover. My objective is to sift and condense what I consider some of the most important resources into a single, structured resource in the form of a book.

WHO THIS BOOK IS FOR

This book is for developers who have recently attended Sitecore .NET Developer Training (SND — `http://bit.ly/ru1jen`). While it may contain useful information for users, administrators, project managers, or anyone not familiar with the Sitecore ASP.NET CMS, I did not intend this book for those audiences. Though experienced Sitecore developers may use this book as a form of reference, and jump from section to section without reading the entire volume, I encourage all readers to review all of the material in the order presented within these pages.

www.it-ebooks.info

Experience with other CMS software can be a benefit or a hindrance to developers learning Sitecore. Terms such as *template* can have different meanings on different platforms. Worse, experience with poor CMS platform architectures can engrain development practices that can be hard to break. Especially when working with Sitecore staff and other Sitecore developers, try to think, write, and talk using Sitecore terminology. Do not be ashamed to use the Sitecore Support Portal (`http://support.sitecore.net`) or the Sitecore Developer Network (`http://sdn.sitecore.net`) forums (`http://bit.ly/vbOyf8`) to ask experienced Sitecore developers for specific pointers before proceeding down an inefficient path.

Because of its architecture, and its adherence to and support for common principles, standards, and typical requirements in ASP.NET, CMS, and web development problem spaces, Sitecore can be an excellent training platform for developers learning these and related technologies.



Because I believe eXtensible Stylesheet Language (XSL) has significant disadvantages for developers with competency in .NET, this book touches upon, but does not focus on, XSL.

WHAT THIS BOOK COVERS

This book describes the Sitecore ASP.NET web Content Management System. While it touches upon underlying technologies and tools including Microsoft Windows, Internet Information Server (IIS), the ASP.NET web application server, SQL Server, and Visual Studio, its primary focus is development with the Sitecore ASP.NET web CMS.

Although much of the information in this book is available through Sitecore training, in Sitecore documentation, and on the web at large, I sincerely hope that this attempt at a somewhat comprehensive resource can help new Sitecore developers orient themselves to the system. At the same time, a single book cannot provide a completely comprehensive reference for the entire Sitecore CMS product. Considering the components and capabilities of the entire Sitecore Customer Engagement Platform (CEP), which consists of the Sitecore CMS plus the Sitecore Digital Marketing System (DMS) that provides web analytics, enhanced personalization, marketing automation, and potentially additional functionality such as email campaign and web forms management, Sitecore is a very broad and very deep framework.

This book is not a step-by-step guide to building solutions with Sitecore. Instead, it contains a very large number of very small facts intended to improve your capabilities on a wide range of topics. The reader should have a working knowledge of software development methodologies, ASP.NET, and the Sitecore ASP.NET CMS user interfaces and fundamental concepts before reading this book. While reinforcing and adding detail around existing knowledge, this book attempts to provide the next few rungs on the ladder towards Sitecore mastery, with pointers to additional rungs where possible.

www.it-ebooks.info

I developed this book using Sitecore CMS 6.5.0 rev. 110818 (Update-1), but most of the information it contains pertains to any 6.x version of Sitecore. This book includes minor details about other versions of the Sitecore ASP.NET CMS.

HOW THIS BOOK IS STRUCTURED

This book introduces subjects in a logical order that builds on itself. The sequence of chapters also provides some correspondence to order of importance, with material early in the book having more significance than subsequent content. Chapter 1 introduces the ASP.NET web Content Management System (CMS).

All web content management solutions involve two major components: a mechanism for representing data and a system of rules to present that data as part of one or more managed websites. The data is fundamental to the solution: without data, the presentation layer cannot do anything. The data of a website, also known as its information architecture, tends to outlive the visual presentation of that data; data is permanent while code is fleeting. Therefore, Chapter 2 of this book begins by describing data infrastructure aspects of the Sitecore CMS, and Chapter 3 proceeds to explain how you can present that data on your websites.

After you have data and code, the next step often involves restricting access to those resources. Therefore, Chapter 4 covers the Sitecore security model. Because all web solutions experience errors, which attackers can use to compromise the security of the system, Chapter 5 describes error management techniques that you can use with Sitecore.

After you have developed your information architecture and the components required to present that data, secured that data, and addressed error conditions, the next common concerns include performance and scalability. Chapter 6 provides guidance to help you maximize the throughput and capacity of your Sitecore solutions.

With this background, you are ready to begin customizing and extending Sitecore in various ways as described in Chapter 7, which focuses on integration. After you have implemented the majority of your solution, you can begin testing it using the information provided in Chapter 8.

Chapter 9 focuses on how to manage Sitecore implementations, both from the perspective of project management and implementation, but also considering various options for release management to get updates from your development environment to your production content delivery environments.

After you have mastered core CMS concepts, you can read Chapter 10, which includes information about Sitecore products beyond its core CMS. These products include the Digital Marketing System (DMS) used for web analytics, experience management, engagement automation, and monitoring.

Chapter 11 concludes this book with tips, tricks, and best practices for working efficiently with the Sitecore CMS. This chapter repeats some of the most important information provided in the preceding chapters, but also provides details not included anywhere else.

www.it-ebooks.info

Appendix A lists a wide array of resources available for Sitecore developers including online materials and in-person events. Appendix B provides guidance for installing the Sitecore ASP.NET CMS.

WHAT YOU NEED TO USE THIS BOOK

To use the code sample code provided in this book, you will need:

- A Microsoft Windows computer meeting Minimum Sitecore Development Host Requirements listed in the Sitecore Installation Guide (`http://bit.ly/pv7X3B`), with IIS installed and ASP.NET enabled
- A licensed copy of the Sitecore ASP.NET web CMS
- Microsoft SQL Server (any edition including Express)
- Microsoft Visual Studio 2010 (any edition other than Express)

I highly recommend that you complete Sitecore .NET Developer Training (SND — see `http://bit.ly/wOc6Xf`) before you read this book.

CONVENTIONS

To help you get the most from the text and keep track of what is happening, we used a number of conventions throughout the book.

- We *highlight* new terms and important words when we introduce them.
- We show keyboard strokes like this: Ctrl+A.
- We show file names and code within the text like so: `persistence.properties`
- We present code in two different ways:

```
We use a monofont type with no highlighting for most code examples.
We use bold to emphasize code that is particularly important in the present context
or to show changes from a previous code snippet
```



As described further in this book, features including the Sitecore configuration factory support Web.config include files that allow you to patch the actual `/web.config` *file. This book describes the Web.config file in relation to features that support include files, but includes a leading slash and applies a style* (`/web.config`) *when referring to the actual* `/web.config` *file itself. Web.config include files can apply only to elements within the* `/configuration/sitecore` *section.*

www.it-ebooks.info

SOURCE CODE

As you work through the examples in this book, you may choose either to type in all the code manually, or to use the source code files that accompany the book. All the source code used in this book is available for download at `www.wrox.com`. When at the site, simply locate the book's title (use the Search box or one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book. The following icon highlights code that is included on the website:



Listings include the filename in the title. If it is just a code snippet, you'll find the filename in a code note such as this:

code snippet filename



Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-0-470-93901-7.

After you download the code, just decompress it with your favorite compression tool. Alternately, you can go to the main Wrox code download page at `www.wrox.com/dynamic/books/download.aspx` to see the code available for this book and all other Wrox books.

ERRATA

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata, you may save another reader hours of frustration, and at the same time, you will be helping us provide even higher quality information.

To find the errata page for this book, go to `www.wrox.com` and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page, you can view all errata that has been submitted for this book and posted by Wrox editors. A complete book list, including links to each book's errata, is also available at `www.wrox.com/misc-pages/booklist.shtml`.

If you don't spot "your" error on the Book Errata page, go to `www.wrox.com/contact/techsupport.shtml` and complete the form there to send us the error you have found. We'll check

www.it-ebooks.info

the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

P2P.WROX.COM

For author and peer discussion, join the P2P forums at `p2p.wrox.com`. The forums are a Web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At `http://p2p.wrox.com`, you will find a number of different forums that will help you, not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to `p2p.wrox.com` and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join, as well as any optional information you wish to provide, and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.



You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works, as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

www.it-ebooks.info

1

Introducing the Sitecore ASP.NET CMS

WHAT'S IN THIS CHAPTER?

- Reviewing web content management systems
- Separating content management environments
- Sitecore product and company history
- Identifying Sitecore solution components

This chapter explains the purpose of a web content management system (CMS), describes the separation of CMS environments, such as test and production, and provides an overview of Sitecore — both the company and its namesake ASP.NET (Application Server Pages with the .NET Framework) CMS, as well as the history of that product. Finally, this chapter describes the components that comprise a Sitecore instance, which is an installation of the Sitecore ASP .NET CMS product.

Web content management software enables nontechnical CMS users to maintain the content driving a web solution without the need to understand its technical implementation. Sitecore uses the ASP.NET platform provided by the Internet Information Server (IIS) web server on the Microsoft Windows operating system to provide one of the most advanced and comprehensive web content management solutions and development frameworks available today.

No web content management system delivers the solution you need without some level of input, whether in the form of entering content and choosing predefined presentation components or building an entire solution. While predefined presentation components are available for the platform, Sitecore is on the latter end of this continuum, focusing on CMS developers as much as users, providing both a finished product and a development framework.

www.it-ebooks.info

UNDERSTANDING WEB CONTENT MANAGEMENT SYSTEMS

Web content management systems are typically web applications that provide browser-based user interfaces for users within an organization (CMS users) to maintain the content and often some aspects of the presentation of one or more published websites. Web content management systems typically depend upon a web server and an application server or the Common Gateway Interface (CGI — see `http://bit.ly/tvN7Vg`). Individual content management systems provide additional specific features such as authentication, authorization, locking, versioning, workflow, audit, internationalization, and localization of the content, as well as separation of work-in-progress data from the published websites.

Web content management systems typically store structured, unstructured, and binary content. Presentation components of the system format that content as markup, such as HTML, for clients such as web browsers. That markup typically contains embedded links to media and other resources in the CMS. One primary aim is to enable nontechnical subject matter experts to maintain the content of websites without having to learn technologies such as HTML or install client-side software. Additional goals typically include consistency in the presentation of the published websites, as well as reuse of content among pages and multiple managed sites, as well as formatting that content differently for various devices (different markup for browsers, mobile devices, printers, and so forth), or content reuse for any other purpose.

There are two primary types of web content management systems. In the early days when web servers and web application servers were not tightly integrated (think CGI), the first CMS tools merged content with code in some kind of generation process run in a content management environment and then published the result to a content delivery environment as files. That result could be static markup to serve without processing (which generally required a separate architecture to manage web applications as opposed to static pages) or code to process further at runtime using a web application server. More advanced solutions stored information in database records in addition to or instead of files, often using query string parameters or other URL components to identify those records.

As application servers became more common, and especially as ASP.NET merged the application server with the web server, content management systems evolved to generate output dynamically for each HTTP request. These modern systems merge content with presentation at runtime to generate each page. While publishing static files can have some benefits, the advantages of generating output at runtime become more clear as the Internet evolves further into a dynamic, integrated, social experience personalized for each visitor.

Due in part to the incredible pace of change on the web, no CMS software can possibly generate your site automatically, or even meet all of your current expectations, let alone your potential future requirements. In fact, requirements typically change between platform selection and the time the website becomes available to public visitors. A website may have a shelf life of just a few years before it begins to appear stale, and organizations frequently acquire other organizations or go through other rebranding exercises. The solution architecture must be flexible enough to support new features and visual redesign. Consider a CMS as a development platform or framework rather than a canned solution. Think of CMS implementation as an ongoing program, not a project with a fixed duration.

www.it-ebooks.info

Content management solutions typically involve a number of technical environments. Developers often install Sitecore on their workstations, though some organizations employ a shared development environment for multiple developers. From there, code changes should go through at least an integration test environment and then additional environments such as user acceptance testing (UAT). When complete, changes arrive in the content management production environment, where CMS users update the site; and from content management production to content delivery, where visitors access the published website. Some organizations deploy changes to a *staging* environment before they reach content management production and content delivery.



As opposed to code changes initiated by developers, content changes often go through a separate publishing workflow process initiated by CMS users in the production content management environment.

Therefore, there are actually two production environments: production content management and production content delivery. Content delivery is the most critical environment; if that environment is down, the published website is down. If the production content management environment is down, CMS users cannot update the site.

INTRODUCING SITECORE

Sitecore is a complete ASP.NET development platform and web content management system (CMS). Sitecore is the leading ASP.NET web CMS available worldwide today. Its unique open architecture, diverse capabilities, and extensibility make it the most advanced CMS on the market. Sitecore developers generally agree that they can deliver more value on their projects in less time using the Sitecore product suite than using any other system.

Sitecore is a high-performance, scalable, extensible ASP.NET web CMS that generates all output dynamically and uses the same components and system architecture for both content and application pages. Sitecore does not limit the markup you can generate or the technologies that you can use, such as XHTML (eXtensible HyperText Markup Language), HTML5, CSS Cascading Style Sheets (CSS), and Asynchronous JavaScript and XML (AJAX).

The Sitecore CMS provides at least the following array of facilities for managing complex and dynamic websites:

- Flexible, hierarchical data storage
- APIs and points of configuration and extension, including hooks into the user interface, pipelines, event handlers, and much more
- A layout engine that dynamically assembles and caches ASP.NET control trees, making it easier to implement a consistent solution by reusing code and content than an inconsistent solution by cutting and pasting
- The concept of devices, which enable different presentations of individual content items under varying conditions

www.it-ebooks.info

- Security, including authentication, authorization, role, and profile management based on ASP.NET membership providers
- Workflow and separation of unpublished content from published content
- Visual comparison of differences between versions of field values
- A media library for managing binary assets
- A rules engine for browser-based configuration of dynamic logic
- Optional modules for additional functionality, most importantly the Digital Marketing System (DMS) described in Chapter 10
- Developer tools such as the free Sitecore Rocks extension for Microsoft Visual Studio

Sitecore keeps content and presentation separate until runtime, when it merges the two based on the context of the user (their personal profile and access rights), the request (the content item identified from the requested URL), and potentially other criteria such as the type of device that requested the page. Based on this information, the layout engine assembles a hierarchy of presentation controls, and then determines whether to execute each or retrieve output cached previously under equivalent processing conditions.

You can think of Sitecore as providing an extension to ASP.NET itself, including browser-based user interfaces, abstract data storage facilities, complete application programming interfaces (APIs), and opportunities for configuration, extension, and customization.

Logging In

To log in to Sitecore, access the URL `/sitecore` on a Sitecore instance in one of the browsers supported by Sitecore (generally, a current version of Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, or Apple Safari (see the Sitecore Installation Guide at `http://bit.ly/pv7X3B` for information about supported browsers in your version of Sitecore). For example, the URL of the login page for the instance of Sitecore that I used while writing this book is `http://sitecorebook/sitecore`.



If you previously logged in with the Remember Me option, accessing `/sitecore` *may not take you to the Sitecore login page, instead taking you directly into a Sitecore user interface. In this case, you can access the login page at* `/sitecore/login`.

To select one of the Sitecore user interfaces, click Options. Figure 1-1 shows the Sitecore login page.

This book assumes that you are familiar with all three CMS user interfaces available from the Sitecore login page:

- The Page Editor (see Figure 1-2), for nontechnical content contributors.
- The Content Editor (see Figure 1-3), for more advanced content managers.

www.it-ebooks.info

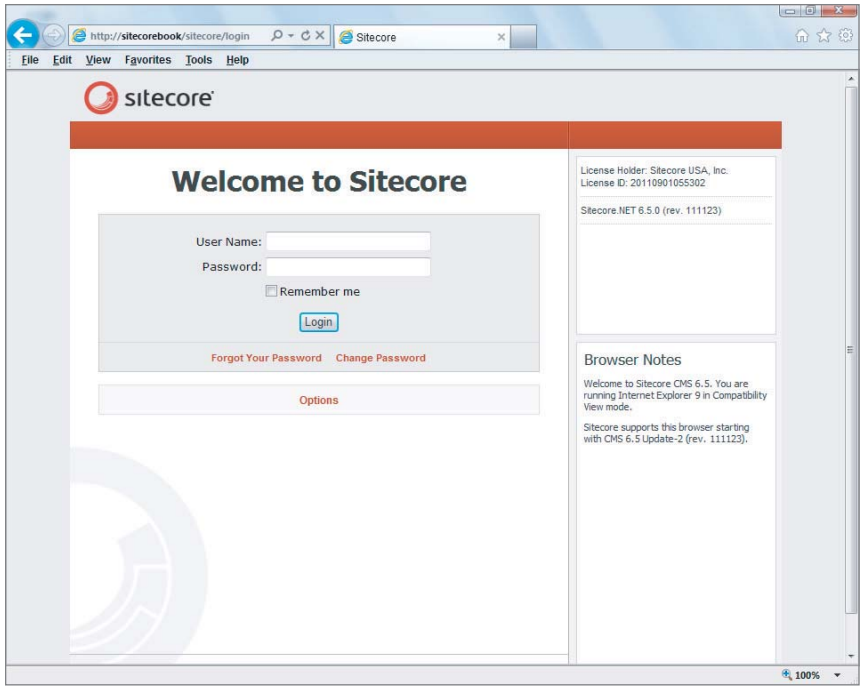


FIGURE 1-1

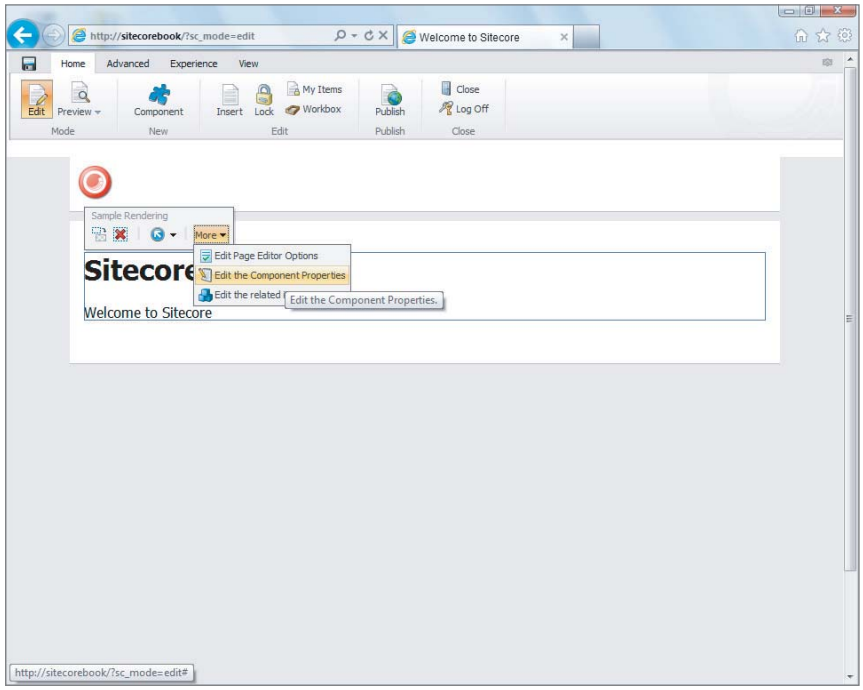


FIGURE 1-2

www.it-ebooks.info

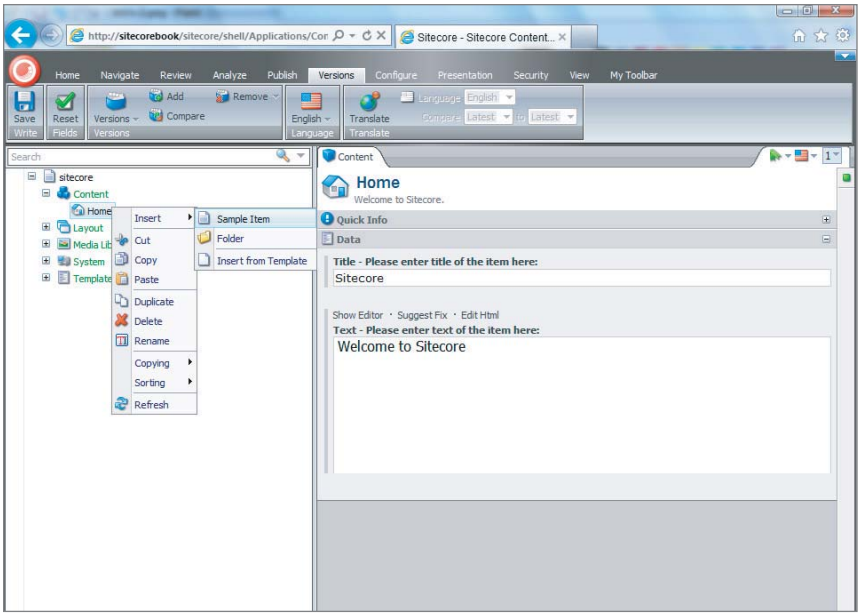


FIGURE 1-3

- The Sitecore desktop (see Figure 1-4), for administrators, advanced content managers, and developers. The image shown in Figure 1-4 appeared after clicking the Sitecore button ➪ Development Tools ➪ Keyboard Map, which provides some context by opening the Keyboard Map application.



The Sitecore desktop functions like a windowing operating system, but runs in a browser. In this book, the term desktop *generally refers to the Sitecore browser-based desktop unless otherwise indicated, such with the phrase* Windows desktop.

Many of the instructions in this book require that you use the Content Editor. To launch the Content Editor, use a browser to access the Sitecore login screen at `/sitecore`. For example, if the URL of the Sitecore instance is `http://sitecorebook`, access `http://sitecorebook/sitecore`. Figure 1-1 shows the Sitecore login screen.

You can use the Content Editor as a standalone application, or you can use the Content Editor within the Sitecore desktop. To choose between these approaches, click Options on the Sitecore login screen. Login options appear on the login screen as shown in Figure 1-5.

www.it-ebooks.info

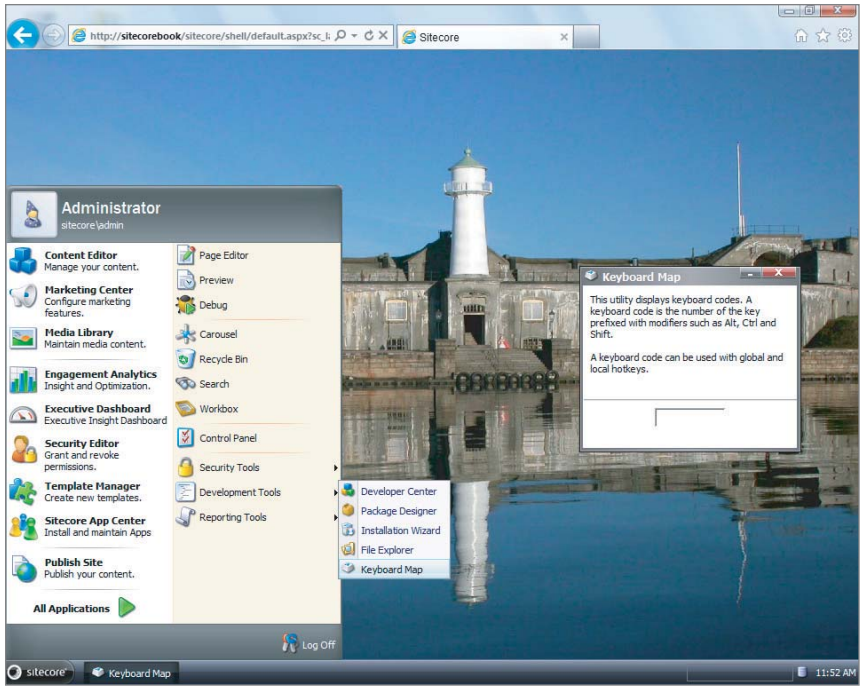


FIGURE 1-4

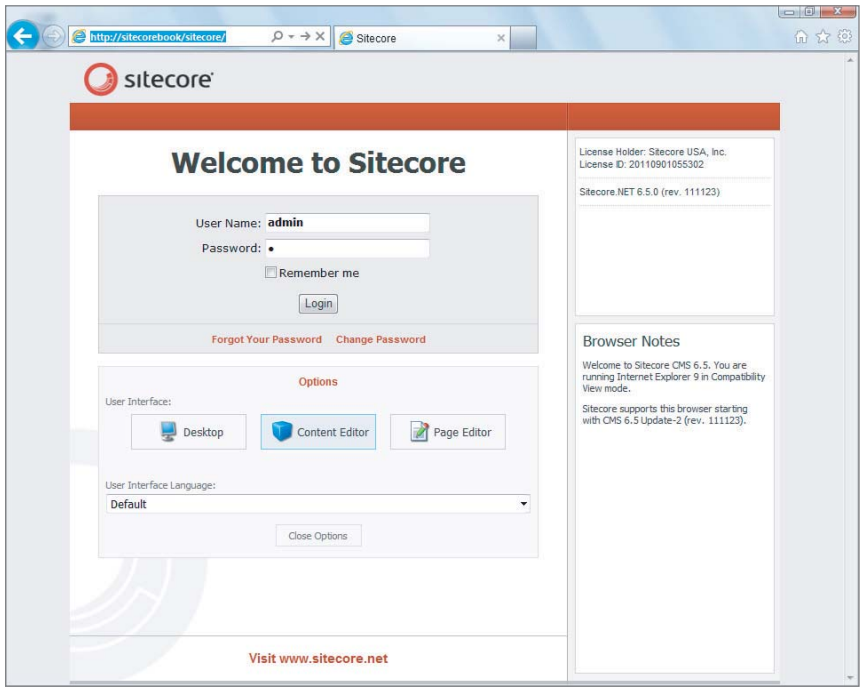


FIGURE 1-5

www.it-ebooks.info

Enter your authentication credentials. To use the Content Editor as a standalone application, double-click Content Editor. The Content Editor appears as shown in Figure 1-6.

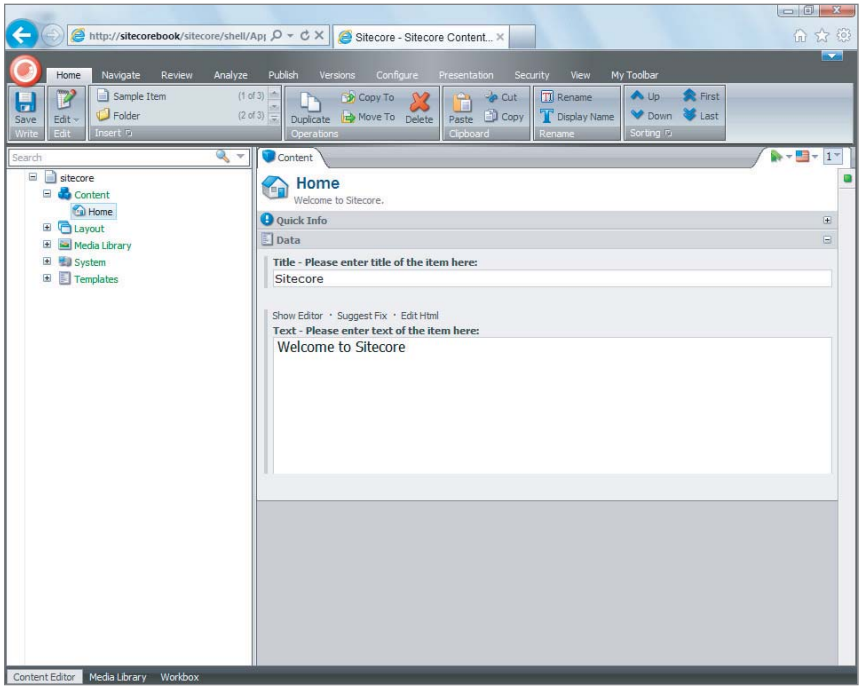


FIGURE 1-6

Alternatively, to use the Sitecore desktop, enter your authentication credentials and double-click Desktop on the Sitecore login screen. The Sitecore desktop appears as shown in Figure 1-7.



The photograph shown in Figure 1-7 is of a lighthouse at the Danish Trekroner Fort (see `http://bit.ly/ABmTge`*). Sitecore initially developed and continues to engineer much of the CMS in Denmark. Sitecore desktop background images distributed with the CMS are similar to the desktop background images provided with the Microsoft Windows operating system, but are intended for use within the Sitecore browser-based desktop.*

Click the Sitecore button in the lower-left corner of the Sitecore desktop. The Sitecore menu appears as shown in Figure 1-8.

www.it-ebooks.info

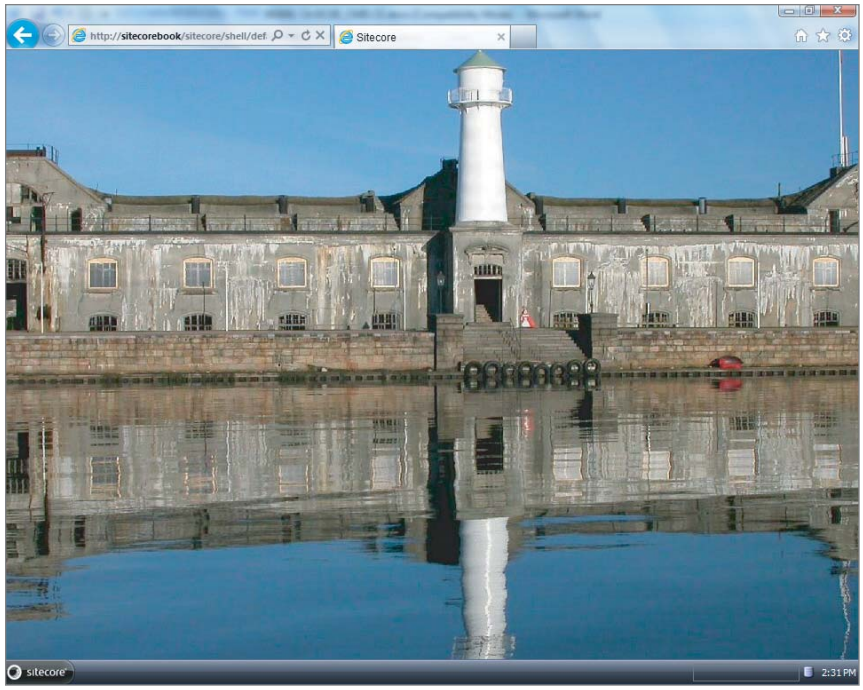


FIGURE 1-7

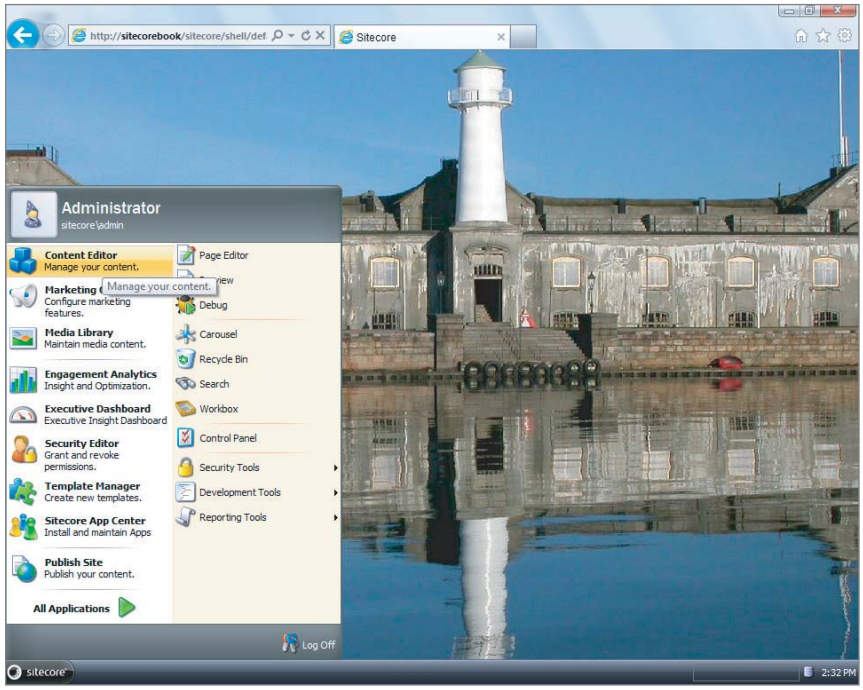


FIGURE 1-8

www.it-ebooks.info

Click Content Editor. The Content Editor appears within the Sitecore desktop as shown in Figure 1-9.

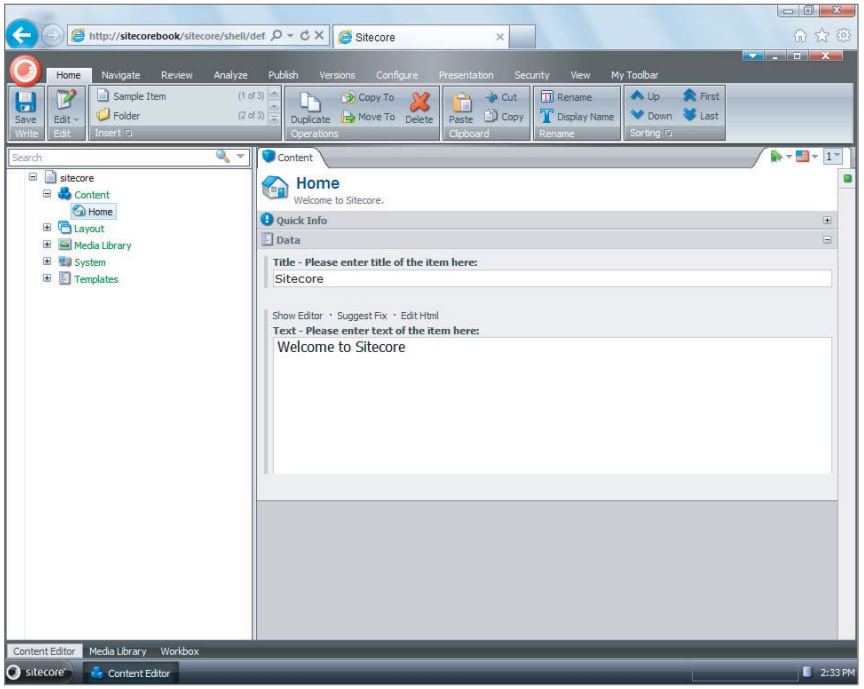


FIGURE 1-9

Embedded Applications

From within each of the Sitecore user interfaces, and especially from within the Sitecore desktop, you can access additional Sitecore applications. These applications include the following:

- **Marketing Center** — Configure marketing programs and tracking elements to help understand who is visiting the site, what brought them there, and which programs and content are most engaging.
- **Media Library** — The Media Library is just the Content Editor with the Media Library tab selected, exposing a repository for binary assets such as images and video.



I am not aware of any task that you can perform in the Media Library that you cannot accomplish by navigating to the `/sitecore/media library` *branch in the Content Editor. Due to (arguably) better usability, some users may prefer to access the Media Library to work with media items rather than the Content Editor.*

www.it-ebooks.info

- **Engagement Analytics** — Measure the effectiveness and efficiency of your website and marketing programs (requires the Digital Marketing Suite, or DMS).
- **Executive Dashboard** — Assess from a high level how marketing channels, traffic sources, and campaigns perform (requires the Digital Marketing Suite, or DMS).
- **Security Editor** — Apply access rights to items for users and roles.
- **Template Manager** — The Template Manager is simply the Content Editor rooted at the `/sitecore/templates` item.



I am not aware of any task that you can perform in the Template Manager that you cannot achieve by navigating to the `/sitecore/templates` *branch in the Content Editor. Due to (arguably) better usability, some developers may prefer to access the Template Manager to work with data templates rather than the Content Editor.*

- **Sitecore App Center** — Configure and manage email, social media monitoring, Windows Azure hosting, search engine optimization (SEO), translation, standards compliance, and other cloud-based services.
- **Preview** — View the site as its content may have appeared at a point in the past or as it may appear at a point in the future.
- **Debugger** — Activate the Sitecore browser-based debugger to analyze your pages and their performance.
- **Carousel** — Visually analyze internal links between items.
- **Recycle Bin** — Review and restore deleted items.
- **Search** — Search the content database.
- **Workbox** — Manage active workflow processes.
- **Control Panel** — Manage system and personal configuration options.
- **Access Viewer** — Review effective security access rights.
- **Domain Manager** — Manage security domains.
- **Role Manager** — Manage security roles.
- **User Manager** — Manage CMS users and identifiable visitors of the published sites.
- **Developer Center** — Use a browser-based integrated development environment (IDE) to manage layouts, sublayouts, XSL renderings, and other development resources.



When possible, use Microsoft Visual Studio, especially with Sitecore Rocks, in favor of the Developer Center.

www.it-ebooks.info

- **Package Designer** — Create packages for migrating components from one Sitecore environment to another.
- **Installation Wizard** — Install packages created with the Package Designer.
- **File Explorer** — Navigate the document root filesystem of the IIS website hosting the Sitecore solution.
- **Keyboard Map** — Collect codes for use in keyboard shortcuts.
- **Log Viewer** — View Sitecore log files.
- **Broken Links** — Manage broken internal links.

Additionally, specific Sitecore user interfaces often contain further embedded interfaces. For example, when you select a definition item for a data template in the Content Editor or the Template Manager, Sitecore displays the Template Builder, which is a custom editor for items based on that data template. For more information about data templates, see Chapter 2.

This book assumes that you have some familiarity with each of these user interfaces, or can quickly learn to use them with minimal instruction and no screen shots. By following the Microsoft Windows and Office UI conventions wherever possible, Sitecore user interfaces support usability through familiarity and discoverability, and provide for application configuration at both the role and user levels.

Working with Sitecore

Sitecore is an ASP.NET application that uses the same architecture and APIs that you use to develop solutions with Sitecore: the Sitecore user interfaces are in fact a website managed with Sitecore. It can take some time to adjust to this concept, so to say it another way, Sitecore uses the same technologies and its own APIs to build the CMS user interfaces that you use to build solutions with Sitecore. This benefits Sitecore developers who need to customize the Sitecore user interfaces by lowering the amount of API knowledge required to achieve such objectives.

With Sitecore, URLs on the published websites often do not correspond to files under the document root or in a virtual directory on the web server, but to items (records) in a relational database that indicate what presentation components the layout engine should use to process HTTP requests that specify those items. While mapping URLs to files can have some benefits, mapping URLs to items in a data store has tremendous value in terms of content and presentation component reusability.

Because Sitecore is a development framework, you can implement an information architecture and presentation components to generate output in any way you want. Sitecore does not provide data structures or presentation components to capture and present your solution; you use Sitecore to implement these components to deliver the exact solutions needed. The platform is incredibly flexible, providing comprehensive validation and other features that you can configure and extend to meet almost any foreseeable requirements. Sitecore's robust combination of architecture, flexibility, efficiency, and, most important, usability, reduces the risk that a web solution might not reach production or achieve user adoption, and increases the project's potential return on investment (ROI).

www.it-ebooks.info

Sitecore provides additional software such as the Digital Marketing Suite (DMS) for advanced experience management, personalization, and web analytics, as well as additional software products and services that work with its web content management system. These include an Intranet management product, the Foundry product for dynamically syndicating numerous similar websites, e-commerce products, a framework for integrating Sitecore with Microsoft SharePoint, and numerous others as described in Chapter 10.

Sitecore Company History

After working together extensively on numerous technical and other projects in and out of school, six Copenhagen University graduates founded Sitecore in 2001. Sitecore focused on its ASP.NET web content management system as its core competency, building a partner channel to deliver consulting and entire solutions to its customers. Sitecore quickly capitalized on the rise of ASP.NET from the ashes of classic ASP (Application Server Pages without .NET) and the foibles of Java and missteps of several Java-based CMS platforms, outcompeting large vendors in established markets and expanding worldwide.

Even in difficult global economic conditions, Sitecore continues to announce record business performance, including an impressive number of new customers every month. Sitecore now has more than 390 employees around the world, primarily in research, development, and innovation, and more than 750 partners that function as an extension of the organization. Thousands of organizations use Sitecore to manage tens of thousands of websites, and approximately 15,000 users can access the Sitecore Developer Network (SDN — `http://sdn.sitecore.net`).

Sitecore ASP.NET CMS Version History

When I joined Sitecore in the summer of 2004, the current distribution of Sitecore CMS was version 4.3.2.x. I would not have joined Sitecore if my research had not lead me to believe it was the best CMS available at the time, and I think Sitecore 4.x would probably still be a viable CMS for many projects.

Sitecore CMS 5.0 through 5.3 replaced the Sitecore 4 concept of layout groups with the concept of devices, replaced a simple version approval checkbox with advanced workflow capabilities, added the extranet, security, core, archive, and Recycle Bin databases, and introduced the Sitecore desktop, built with new Sitecore UI technology. Possibly most important for developers, Sitecore CMS 5 provided a consolidated API for both the CMS and the published websites, whereas Sitecore 4 had separate APIs for each. With Sitecore CMS 5, Sitecore introduced the current Sitecore Developer Network.

Sitecore CMS 6.0 introduced standard values, branch templates and command templates (all of which eliminated the need for the concept of *masters* used to instantiate new items in earlier Sitecore versions). Sitecore 6 improved usability and performance in the Content Editor, replaced WebEdit (with the circular content markers) with the Page Editor, and replaced a proprietary security implementation with ASP.NET membership providers. Sitecore 6 eliminated the Archive and Recycle Bin databases by implementing tables for these features in the content databases, and eliminated the Security and Extranet databases by moving security information to the core database.

Sitecore CMS 6.1 introduced the rules engine, added the Field Editor and Edit Frames, provided a foundation for the Online Marketing Suite (now the Digital Marketing System, or DMS), and introduced rendering parameters templates.

www.it-ebooks.info

Sitecore CMS 6.2 added support for WebDAV (Web-based Distributed Authoring and Versioning) and the File Drop Area field type, introduced the Word Document field type, and provided support for public and Sitecore Client RSS feeds.

Sitecore CMS 6.3 improved Sitecore's architecture for horizontal scalability by introducing remote events, eliminating the staging module, supporting load balancing in the content management environment, and introducing support for Sitecore Azure to deploy solutions to the Microsoft Windows Azure Internet cloud.

Sitecore CMS 6.4 improved the Page Editor and its extensibility, updated the Telerik Rich Text Editor included with the CMS, introduced compatibility with and support for .NET 4.0 and ASP .NET MVC, provided clones, the notification engine, and layout deltas, and was the first version of Sitecore to support additional browsers in the Sitecore Desktop.

Sitecore 6.5 provided the foundation for the Digital Marketing System (DMS) and hence Customer Engagement Platform (CEP), which replaces Sitecore's Online Marketing Suite (OMS), its first platform supporting integrated web analytics. In addition to simple traffic measurement and other common web statistics, CEP supports efficient marketing and engagement automation, automatic visitor origin classification, as well as analysis of the value of each visit and the efficiency of paths through the website. In comparison to OMS, DMS provides improved scalability, greater performance, enhanced report readability, improved capabilities for real-time personalization, and high-level dashboards.

Sitecore Components

Sitecore is an ASP.NET application that provides a layer between the developer and the ASP.NET framework, including components such as databases.



Follow Microsoft best practices for Internet Information Services (IIS) and ASP .NET security, performance, and scalability, load testing, hardware requirements planning, and other administrative considerations.

A Sitecore installation consists of five basic components: some number of relational databases (three by default), a Microsoft Windows Internet Information Services (IIS) website, an application pool, a corresponding filesystem for the document root of that website, and a separate subdirectory for storing data files (technically, that subdirectory can exist within the document root). Most Sitecore solutions include a Visual Studio project, and I recommend that all Sitecore developers use the free Sitecore Rocks extension for Visual Studio. For instructions to install Sitecore, see Appendix B.

Sitecore Databases

Each Sitecore instance depends on some number of Microsoft SQL Server or Oracle relational databases. You can install the database server on the same machine as the web server. For production, you should use one or more separate, dedicated database servers to host the Sitecore databases.

www.it-ebooks.info



You can use the free Express versions of Microsoft SQL Server and Oracle in development, test, and potentially additional non-production environments. Because you write code against a Sitecore API layer that abstracts the underlying storage mechanism, you can even use different database vendors in different environments, unless you write code that interacts directly with the database.

The three default Sitecore databases serve the following purposes:

- **Master** — The master database contains all versions of all content, including changes not yet published to the live website. Excluding the browser-based Sitecore debugger that accesses the published website by default, CMS user interfaces access the master database by default.
- **Web** — The web database contains the current published version of each item in each language. The web database is the default publishing target database: by default, Sitecore publishes changes from the master database to the web database. You can add additional publishing target databases as described in Chapter 9.
- **Core** — The core database defines the Sitecore user interfaces, and controls which users can access which features of the various Sitecore applications. By default, the core database also contains the tables used by the ASP.NET security provider implementations, as well as system tables such as that used for internal link management. The managed website that provides the Sitecore user interfaces accesses the core database and allows the user to view and edit content in the master database by default.

Each of these databases contains very similar database schemas, but the core database typically contains tables for its additional default function. You can configure Sitecore to use different databases for each function.

You can sometimes add and eliminate databases, such as by adding one or more additional publishing target databases in the content management environment or by removing the master database from a content delivery environment. If you configure multiple Sitecore instances to write to a single database, to ensure appropriate cache clearing, you must enable remote event management on both instances. For more information about eliminating databases and sharing databases between instances, see The Sitecore Scaling Guide (`http://bit.ly/oKvYuN`).

Optional modules, including the Digital Marketing Suite (DMS) and Web Forms for Marketers (WFFM), require additional relational databases. Apply enterprise backup and optimization techniques to the Sitecore databases, including rebuilding database indexes.

Hosting IIS Websites

Sitecore requires an IIS (Internet Information Services) website with a corresponding application pool and document root to host the ASP.NET application. For more information about IIS and ASP.NET, see `http://iis.net` and `http://asp.net`, respectively.

You can use IIS to host any number of websites. IIS bindings determine which website services a request based on the hostname, port, and protocol (HTTP or HTTPS).

www.it-ebooks.info



Technically, the root subdirectory of the website (its document root) and each subdirectory can expose an ASP.NET application, and each application can use a separate application pool. Sitecore does not support applications in subdirectories.

Sitecore can use a single IIS website to manage multiple logical websites. While IIS can manage multiple websites, including multiple instances of Sitecore with separate filesystems, each Sitecore instance can manage multiple logical websites with a common filesystem and databases.

IIS Application Pool

Each IIS website can expose an ASP.NET application, which depends on an application pool. The application pool hosts an ASP.NET application, and Sitecore is an ASP.NET application. The Sitecore setup executable creates an IIS application pool for the instance. If you install Sitecore without using the setup executable, you should manually create an application pool for each new instance.

Windows uses a process to host the application pool, typically `w3wp.exe` but sometimes `aspnet_wp.exe`. A Windows user such as Network Service or an application pool identity owns the application pool process. For information about application pool identities, configuring filesystem access rights, and other aspects of the application pool, see Appendix B.

Document Root Subdirectory

The IIS website references a document root subdirectory. By default, this subdirectory contains the Sitecore application and supporting files. Your solution files will appear in this subdirectory. The document root subdirectory contains the `/web.config` file that configures the ASP.NET application at that location.



As described further in this book, features including the Sitecore configuration factory support Web.config include files. Web.config include files allow you to patch the actual `/web.config` *file. This book describes the Web.config file in relation to features that support include files, but includes a leading slash and applies a style (*`/web.config`*) when referring to the actual* `/web.config` *file itself. Web.config include files can only apply to elements within the* `/configuration/sitecore` *section.*

Data Subdirectory

Sitecore uses the subdirectory specified by the `dataFolder` variable in the Web.config file to store data files such as its license, logs, and Lucene search indexes.

www.it-ebooks.info



Lucene (`http://lucene.apache.org`*) is an open-source search engine maintained by the Apache group (*`http://apache.org`*). For more information about Sitecore's use of Lucene, see* `http://bit.ly/rxaImG`.

Sitecore Visual Studio Project

Most Sitecore solutions involve at least one Microsoft Visual Studio project. Install Visual Studio on development workstations, not on production servers. Most Sitecore developers use IIS rather than the web server built into Visual Studio. For instructions on how to create a Visual Studio project for your Sitecore solution, see Appendix B.

Sitecore Rocks

Developers working with Visual Studio and the Sitecore ASP.NET CMS should use the Sitecore Rocks (`http://sitecorerocks.net`) extension for Visual Studio. Most Sitecore developers use Visual Studio and can benefit greatly from this free tool from Sitecore.



Install Sitecore Rocks on the development workstations on which you install Visual Studio.

For more information about Sitecore Rocks, see my blog post at `http://bit.ly/oZTaZI`. For additional useful tools for Sitecore developers, see my blog post at `http://bit.ly/nQHKfH`.

TAKE HOME POINTS

This chapter introduced web content management systems (CMS), CMS environments, the Sitecore ASP.NET CMS product, the Sitecore company, and the main components of a Sitecore installation. Sitecore developers using Visual Studio can implement advanced, scalable, consistent Sitecore solutions on the Microsoft Windows, IIS, and ASP.NET infrastructure to enable nontechnical CMS users to maintain structured content to populate a website using only a browser. Sitecore is a pure ASP.NET application, and tightly conforms to the core architectural principle of the framework.

The Sitecore solution infrastructure maximizes code and content reuse, and minimizes development and maintenance costs. It also provides a comprehensive, extensible, and flexible platform for your projects and a pleasant working environment for developers. Sitecore provides a variety of browser-based interfaces and applications for various types of users and tasks, as well as tools for developers, including the Sitecore Rocks plug-in for Visual Studio and optional software modules that integrate with the Sitecore CMS.

www.it-ebooks.info

www.it-ebooks.info

2

Information Architecture

WHAT'S IN THIS CHAPTER?

- Creating items, including item properties, definition items, and item URLs
- Using data templates, including base templates, the standard template, standard values, template sections, template fields, field properties, and field types
- Considering data validation, including validators and validation options
- Implementing multilingual solutions, including languages, regions, and multiple managed sites
- Working with binary media, including media library settings and media item URLs
- Understanding clones, aliases, and wildcards
- Using insert options, including branch templates, command templates, and the uiGetMasters pipeline

This chapter introduces fundamental high-level concepts and low-level constructs and techniques used to implement and enforce information architecture with the Sitecore ASP.NET web content management system (CMS). Data templates define the structure of each type of item that you can store in a Sitecore database. Some types of items represent pages, such as the home item for each managed site. Sitecore contains logic to determine the URL for each such item, and to determine the item indicated by the URL in an incoming HTTP request. Sitecore uses items to manage numerous additional types of information, including configuration data, system components, and binary media.

www.it-ebooks.info

Each data template defines zero or more sections that contain fields to define the structure and properties of a type of items. A data template can inherit sections and fields from any number of base templates. By default, data templates inherit from the standard template, which contains fields that define system properties available to all items. You can translate each item into any number of languages, and create any number of versions within each language. You can specify that Sitecore should not maintain versions or allow translation for the values of specific fields. Standard values define defaults ("standards") for new items based on each data template. You can use validation to enforce data entry requirements. You can import content by invoking Sitecore Application Programming Interfaces (APIs) to create items and update field values.

Insert options control the types of items that users can create beneath existing items to define a hierarchical data structure. You can create item structures to represent websites, lookup lists, metadata taxonomies, and any other kind of data that you can structure in a hierarchy. Sitecore provides a number of facilities that enable you to share items and field values among components. You can create aliases to provide alternate URLs for items. You can create clones to make items appear in multiple locations in the content tree with the ability to override field values in the cloned items without actually duplicating those items. You can use wildcards to map URLs to resources in external repositories such as relational databases rather than items in the Sitecore database.

For more information about the topics described in this chapter, including instructions to implement the various components described, see The Sitecore Data Definition Reference (`http://bit.ly/nmGuiB`), The Sitecore Data Definition Cookbook (`http://bit.ly/oc8F9K`), and The Sitecore Data Definition API Cookbook (`http://bit.ly/ohVWlq`).

INTRODUCING INFORMATION ARCHITECTURE

All web content management systems include two major components:

- Facilities to define data structures that allow CMS users to enter site content
- Facilities to present that data in those structures as a website

This chapter focuses on the former component; Chapter 3, which describes the Sitecore layout engine and data access APIs, focuses on the latter.

You can think of information architecture as a model for the structure of the data behind a web solution. Information architecture includes both high-level structures — defining the relationships among managed sites, their home pages, sections, applications, and pages, as well as other features of the solution — and low-level details, defining individual page types and the properties of individual data elements and other features within those pages. The information architecture can describe security, workflow, archiving, validation, and related requirements, and can even specify the presentation components required to render specific types, sections, pieces, and other aspects and groupings of content.

Information architecture may be the most critical component of a web solution. You cannot implement an optimal solution without solid information architecture, let alone enhance that solution over time. The user interface of a website begins to look stale as time passes, but the information architecture of that solution can have a longer life if you can apply a new presentation layer to the existing data.

www.it-ebooks.info

Because definitions for some Sitecore terms result in circular references (for example, items depend on data templates, which depend on items, which depend on data templates), the following list provides a quick overview of key terminology explained in subsequent sections of this chapter:

- **Item** — An item is a data record, similar to an instance of a class in object-oriented programming (defined using the `class` keyword in C#).
- **Data template** — A data template defines the structure of a type of items, much like a class or structure in object-oriented programming.
- **Data template field** — A data template consists of some number of data template fields, which are similar to properties of a class or members of a structure in object-oriented programming.
- **Standard value** — Each data template can specify standard values for each field defined in the template and any of its base templates, much as constructors and other initialization code can set property values in object-oriented programming.
- **Base template** — Each data template can inherit from any number of base data templates, much as a class can inherit from a base class and implement interfaces in object-oriented programming.
- **Standard template** — Most data templates inherit from this standard data template, as all classes eventually inherit from the `System.Object` class in .NET.

Sitecore facilitates hierarchical information architectures. Most developers are familiar with relational databases, but most websites are hierarchical — a home page contains sections, sections contain subsections and pages, and subsections contain nested subsections and pages. Hierarchical information architectures can assist in securing sections and subsections of the site; you can use security inheritance to deny anonymous users read access, or to grant write access for a CMS role to an entire section or subsection.

Items in the information architecture have implicit relationships with other items, such as the parent/child relationship and the preceding-sibling and following-sibling relationships. Items also have explicit relationships with other items, such as when an image field in a content item contains a reference to a media item.

Sitecore uses the information architecture to determine the URLs of content items. URLs include the name of the item preceded by the names of its ancestors. For example, the default URL of the `/sitecore/content/home/section/subsection/page` item is `/section/subsection/page.aspx` relative to the `/sitecore/content/home` item that represents the home page for the default managed site.

Most Sitecore solutions use the information architecture to drive navigation for the site. In such cases the information architecture typically matches the visual structure of the website exactly. For example, the children of the home item might represent sections that appear in a top navigation component. When the user clicks an element in the top navigation, taking the browser to that section, the items representing subsections and pages within the clicked section could appear in the left navigation. Other solutions use techniques such as faceted search, which presents a default navigational structure based on content classification characteristics along multiple dimensions, and

www.it-ebooks.info

allows visitors to navigate in a variety of ways rather than using a single taxonomy, often through the application of filters that eliminate elements in that navigational structure that the visitor deems irrelevant.

SITECORE ITEMS

Sitecore items represent individual resources within the CMS. Items can represent any type of data, such as a page, section, paragraph, or metadata attribute. Each item contains a number of properties common to all languages and versions of the item, such as its name and ID. In addition to content elements, Sitecore uses item to manage system and configuration components.



Make item names unique among siblings. Sitecore does not require this, but provides a validator that you can use to enforce this rule.

Each item can contain any number of field values that can vary by language and by version within each language. Each item exists within a hierarchy of items in a Sitecore database. The path of an item identifies its location within the hierarchy. Sitecore assigns a globally unique identifier (GUID, or just ID) to each item.

Sitecore does not differentiate content from metadata or control data. Each item can contain fields for content, metadata, system data, and potentially other types of data. You can use items to implement metadata and other taxonomies or to manage any data that you can represent in a hierarchy.

Like files, items contain data, but like subdirectories, items can contain other items. This means that you do not have to think about whether something is a subdirectory or a file when you create it; in Sitecore, you create an item regardless of whether you need to store data or contain other items. That item might not have any children today (making it like a file), but it could have children in the future (making it like a subdirectory). Either way, each item can contain field values and other items. If you need to change the fields that appear in an item, you can add fields to the data template associated with the item, or update the item to use a different data template.

You can use the `Sitecode.Data.Items.CustomItem` abstract class as a base class for your classes that represent different types of items. For more information about this approach, see Chapter 3. Many solutions benefit from the implementation of .NET classes to represent various types of Sitecore items. You can use the `Sitecore.Data.Items.CustomItemBase` class as a base class for your classes that represent different types of items. Sitecore Rocks has features that you can use to generate classes from data templates, or you can use either the CustomItemGenerator (`http://bit.ly/xzjzP3`) Sitecore Shared Source project or the CodeGen (`http://bit.ly/w8ppkd`) Sitecore Shared Source project for this purpose.

For more information about many of the concepts described in the remainder of this section, see The Sitecore Guide to Reusing and Sharing Data (`http://bit.ly/pjNlqG`).

www.it-ebooks.info

Item Languages, Versions, and Revisions

You can translate each item into any number of languages. You can register languages before you use them, in which case they are available to all items and you can configure their properties in advance, such as to specify a flag, spellcheck dictionary, and security to control which CMS users can edit content in that language. Alternatively, you can add a version of an item in any language, in which case Sitecore applies default language properties.



Sitecore does not store different values for each language in fields that you mark as shared.

When you register a language, Sitecore creates a definition item under the `/sitecore/system/Languages` item to contain information about that language, such as the flag to display and the spellchecking dictionary to use.



I recommend that you register languages before you use them. That way, you can iterate the language definition items even if no versions exists in those languages in your content items, and you can retrieve metadata about languages consistently from those definition items.

To register a language in the Sitecore desktop, follow these steps:

1. Click Sitecore, and then click Control Panel. The Control Panel appears.
2. Click Globalization in the Control Panel. The Globalization Control Panel appears.
3. Click Add a New Language in the Globalization Control Panel. The Add Language wizard appears. If you have not already disabled wizard welcome pages, a welcome page appears; click Next. The Language Codes page appears as shown in Figure 2-1.
4. In the drop-down list at the top, select the appropriate language and region, and then click Next. The Code page and Encoding page appears.
5. Click Next. The Checker Dictionary page appears.
6. For the Spellchecker file path, enter the name of a spellchecker dictionary file for the language from the `/sitecore/shell/Controls/Rich Text Editor/Dictionaries` subdirectory of the document root. If no file corresponding to the new language exists, leave this field blank. Then click Next. The completion page appears.
7. Click Finish. The Add Language Wizard closes. You can now apply security or update other properties of the new language definition item under the `/sitecore/system/Languages` item in the Content Editor.

www.it-ebooks.info

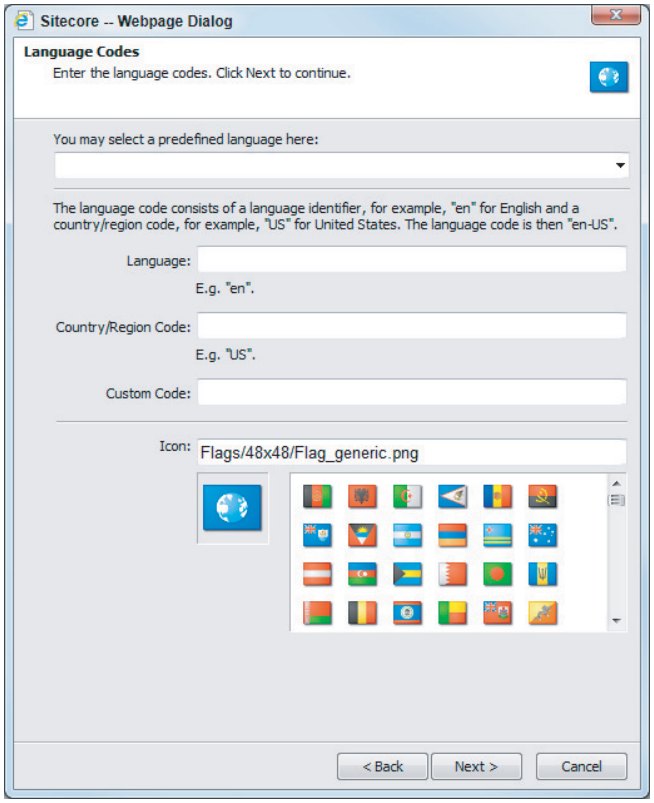


FIGURE 2-1

To create a version of an item in a language that you have not registered:

1. Select the item for which you want to add a version in an unregistered language in the Content Editor.
2. Click the flag at the top-right corner of the editing pane, and then click More Languages. The Select Language dialog appears as shown in Figure 2-2.
3. Select the language, and then click OK. The Select Language dialog closes.
4. Click Add a new version in the warning that appears at the top of the editing pane in the Content Editor. Sitecore creates the first version of the item in the language that you selected. This step applies only for languages registered in advance; no versions exist for a language until you create them.

For an example of a presentation component that links to each registered language for which a version exists for an item, see my blog post at `http://bit.ly/omG5Fd`.

You can create any number of versions of an item in each language. By default Sitecore uses the latest available version of each item in each language. Click the number at the top right corner above

www.it-ebooks.info

the editing pane in the Content Editor to select an alternate version of an item. Click the Versions tab to perform other operations on versions, including deletion and translation.

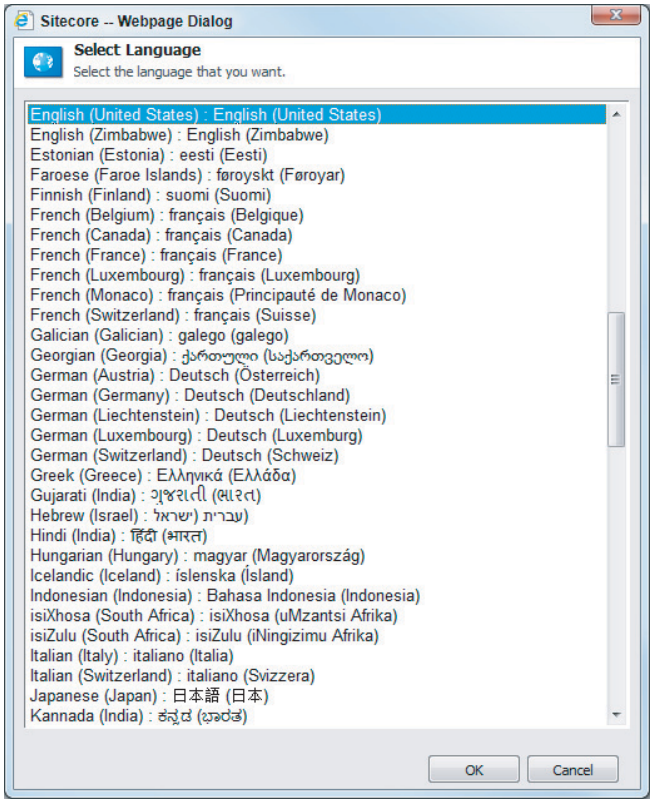


FIGURE 2-2



Having many versions in any number of languages for an item can reduce performance, especially when you work with items that contain a large number of fields, or fields that contain a large number of links. For information about a solution that uses a scheduled process to remove old versions of items, see my blog post at `http://bit.ly/kq4Jmp`.

You may not find the API that removes a version where you expect it. The following code removes the oldest version of the context item:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
item.Versions[item.Versions.GetVersionNumbers()[0]].Versions.RemoveVersion();
```

www.it-ebooks.info



Many examples in this book use the context item exposed by the `Sitecore.Context.Item` *static property. I used this shortcut to avoid repeatedly explaining and reproducing code to retrieve a database and an item within that database. Using the context item should work, but does not always provide the most logical example. You can apply any technique demonstrated for the context item to any item (any instance of the* `Sitecore.Data.Items.Item` *class).*

In addition to a version number, which controls workflow and other features, Sitecore assigns a unique revision ID to each item every time you make a change to the item. Sitecore stores the revision ID in the field named `__Revision` (which you can access through the `Sitecore.FieldIDs.Revision` property that functions like a constant containing the ID of this field) in the Statistics section of the standard template. The `__Revision` field is unrelated to the version number for a language of an item, which Sitecore does not store as a field value, but as a property of that version.



Instead of hard-coding field names or IDs, which can increase maintenance effort, create static classes like the `Sitecore.FieldIDs` *class with properties to expose the IDs of your data template fields. For the same reason, you can create classes like the* `Sitecore.ItemIDs` *class to store the IDs of specific items, and like the* `Sitecore.TemplateIDs` *class to store the IDs of your data templates.*

Item Properties

To control the data type of an item, how that item appears in the user interface, how you can refer to that item in code, and the item's relationships with other items in the database, each item defines all of the properties in the following list:

- **Name** — The name of the item, which is not necessarily unique, even among its siblings.
- **Key** — The lowercase of the item name, which is not necessarily unique, event among siblings.
- **Path** — The names of the item and its ancestors in document order, separated by the slash (`/`) character. (Sitecore does not store the path in the item, but constructs the path dynamically based on its location in the content tree, from the top of the hierarchy down.)
- **Data Template** — The data template, or structure definition, associated with the item. (A property of the item contains the ID of its data template.)
- **ID** — The unique identifier for the item. (An item has the same ID in the Master database and all publishing target databases, and may also exist in the Core database with the same ID.)

In addition, most items can define values for numerous additional fields defined by the standard data template from which almost all other data templates inherit.

www.it-ebooks.info

Sitecore item names have two fundamental purposes: to identify Sitecore items within the item hierarchy, and to provide default URLs for content items. These purposes are sometimes at odds: you might not want space characters in URLs, but whitespace helps users identify items in the CMS. One solution to this dilemma is to use the `__Display Name` (`Sitecore.FieldIDs.DisplayName`) field defined in the Appearance section of the standard template. If that field has a value, Sitecore user interfaces show the display name of the item; otherwise the name of the item appears. Display names also allow different values for different languages.

To set the display name of an item, do the following:

1. Select the item in the Content Editor.
2. Click the Home tab, and then click Display Name in the Rename group. A prompt appears.
3. Enter the new display name for the item, and then click OK. The prompt disappears and you return to the Content Editor.

Sitecore provides for item name validation using the following settings in the Web.config file:



Remember that by the conventions used in this book, Web.config can refer to Web.config include files, where `/web.config` *refers to the actual* `/web.config` *file. For more information about Web.config include files, see my blog post at* `http://bit.ly/j5Ms7C`.

- `InvalidItemNameChars` — Sitecore does not allow these characters in any item names.
- `ItemNameValidation` — Item names must match this regular expression.
- `MaxItemNameLength` — Item name lengths cannot exceed this limit.



Before changing either `InvalidItemNameChars` *or* `ItemNameValidation` *setting in the Web.config file to allow additional characters in item names, I recommend that you check with Sitecore support to confirm that the CMS does not exclude the character(s) under consideration for any technical reason. If you must use special characters in an item name, consider setting the display name of the item instead.*

These settings provide useful safeguards, but do not allow much flexibility. For example, you cannot apply different naming restrictions to items in different branches, items based on different data templates, items with layout details or containing items with layout details, or according to any other logic that you might want to define. For a solution that uses the rules engine to control item names dynamically, see my blog post at `http://bit.ly/qn0w1l`. For more information about the rules engine, see Chapter 7.

www.it-ebooks.info

Item Structure

Sitecore can represent items in a number of formats, including:

- Browser-based user interfaces such as the Content Editor
- Native database storage
- Objects in memory (the `Sitecore.Data.Items.Item` class and its relatives)
- XML containing all field values in all versions in all languages, with or without descendant items
- XML containing only the structure of items, including field definitions but not including language information, versions, or field values, as used by XSL (eXtensible Style Sheet) renderings
- Serialization files used for source control of items in databases as well as exchange with other Sitecore instances
- Package entries used to exchange items in databases with other Sitecore instances

You can use some of these formats to export data. For example, to access the XML representation of an item from .NET, you can use the `GetOuterXml()` method of the `Sitecore.Data.Items.Item` class that represents an item. The first parameter to this method specifies whether to include descendants of the item in that XML. To access the XML representation of items available to XSL renderings, pass a `Sitecore.Data.Items.Item.Item` object to the `Sitecore.Configuration.Factory.CreateItemNavigator()` static method and retrieve the `OuterXml` property of the `Sitecore.Xml.XPath.ItemNavigator` object that it returns. From an XSL rendering you can use an `<xsl:copy-of>` element to write the XML representation of an item and its descendants to the output stream, and then view the source of the rendered page in a browser. For example, to render the XML representation of the context item visible to XSL, you can use the following code in an XSL rendering:

```
<xsl:copy-of select="$sc_currentitem" />
```



Remember that Sitecore APIs automatically apply access rights for the context user. Items to which the context user does not have read access do not appear in either object or XML representations. If code or a requested URL attempts to access an item to which the context user does not have read access, Sitecore returns an error message or behaves as if that item does not exist.

For more information about XSL renderings and the layout engine, see Chapter 3. For an example of using a layout to access the XML representation of items using a browser, see my blog post at `http://bit.ly/p0q9Ge`.



If you render XML over HTTP, remember to set the `System.Web.HttpContext.Current.Response.ContentType` *property to* `text/xml` *so that you can easily use a browser to view that XML. Of the techniques that you can use to format items, handlers as defined within the* `/configuration/sitecore/customHandlers` *element of the Web.config file may be most appropriate for this purpose. Handlers are more efficient than Web Forms.*

www.it-ebooks.info

Definition Items

Definition items specify the properties of system components such as devices renderings used by the layout engine, and even components such as data templates, sections, and fields. Definition items use the same technical infrastructure as content items, which the layout engine renders as pages of managed sites, but definition items do not have URLs and do not contain layout details to inform the layout engine how to render those items. For example, when you register a new language, Sitecore creates a language definition item to store properties of that language. A definition item represents a logical construct in the system, and its fields define properties of that construct.

Because of their simplicity and versatility, Sitecore uses items to represent every type of information it can. Sitecore uses items to represent or define:

- Folders and content, including Really Simple Syndication (RSS) feeds
- Media folders and media items
- Presentation components, including devices, layouts, renderings, and placeholder settings
- URL alias definitions
- Term translations
- Publishing target database definitions
- Language definitions
- Configuration settings, including layout presets, security presets, and various other types of information not specifically mentioned in this list
- Rules engine configuration, including conditional rendering rules
- Data validation features
- Child item sorting rule definitions
- Scheduled task details
- Data template and standard values definitions
- Interface for editing user profiles
- Sitecore user interface components, including the data template field types
- Workflow processes definitions



Instead of using items to store information about items in workflows, Sitecore uses items only to define workflow processes, and stores information about items in those workflows directly to a relational database. Similarly, while the CMS stores security access rules in items, Sitecore uses security providers to manage most attributes of users and roles, and the default providers store that information directly to a relational database.

Items in the Core database configure numerous aspects of the Sitecore user interfaces. In fact, the Sitecore user interfaces rely on a managed site built using much of the same technology that you

www.it-ebooks.info

use to build sites with the CMS, but based on items in the Core database. Optional modules for the Sitecore CMS, including the Digital Marketing System (DMS), also use definition items extensively for a wide range of purposes.

Insert Options

Insert options control the types of items that users can insert beneath existing items. Insert options can include data templates, branch templates, and command templates. You can use insert options to restrict the information architecture and to control what types of items users can create beneath existing items, but more importantly, you can apply insert options in standard values to define insert options for all items of a specific type.



Whenever possible, instead of defining insert options in individual items, define insert options in the standard values of your data templates.

Branch templates allow users to insert predefined structures consisting of some number of items. When you use a branch template to create an item, Sitecore copies all descendants of the branch template definition item, and then expands tokens such as `$name` not just in the standard values for the data templates behind those items, but in the names of those items as well. For more information about tokens such as `$name`, see the section of this chapter about standard values.



Along with standard values, branch templates replace masters, which Sitecore used in versions prior to CMS to 6.0. Earlier versions used masters to constructor new items and hierarchies of items and to define insert options. Branch templates support features beyond those previously supported by masters. For example, a branch template definition item can have more than one child, meaning that you can implement a branch template to insert multiple sibling items simultaneously rather than inserting a single root item as required by masters.

Command templates allow CMS users to insert items by applying logic that you implement. For example, an insert option for the `/sitecore/system/Languages` item enables use of a command template to insert a language definition item under that existing item. That command template contains code that invokes the Add Language wizard described previously, which lets the system use the same user interface and logic that you can invoke from the Globalization Control Panel to register a language.

Sitecore uses the `uiGetMasters` pipeline defined in the Web.config file to determine effective insert options at runtime. The `uiGetMasters` pipeline includes processors that retrieve insert items defined in the selected item, invoke insert rules defined in that item, invoke global insert rules, and then apply security to the result. For more information about pipelines, see Chapter 7.

www.it-ebooks.info



As mentioned previously, Sitecore CMS 5 and earlier versions included masters, which functioned as constructors for new items. Sitecore 6 introduced standard values and branch templates, which eliminated the need for masters. To avoid disruptive changes, master terminology remains in some places in the Sitecore product, including the name of the `uiGetMasters` *pipeline mentioned here.*

To control access to the Insert from Template option that appears with insert options in Sitecore user interfaces, configure access rights for the `/sitecore/content/Applications/Content Editor/Menues/New/Insert from Template` [sic] item in the Core database. You cannot use the `uiGetMasters` pipeline to control access to the Insert from Template option.

Insert options rules use the rules engine to determine insert options dynamically. In the context of insert options, one benefit of insert options rules is that you can define insert options dynamically through a browser-based user interface instead of assigning insert options declaratively or determining insert options programmatically with code in `uiGetMasters` pipeline processors. You can define global insert options rules under the `/sitecore/system/Settings/Rules/Insert Options/Rules` item using the `System/Rules/Insert Options Rule` data template. A processor in the `uiGetMasters` pipeline invokes insert options rules while determining effective insert options for an item.

Insert rules (not to be confused with insert options rules described in the previous paragraph) also determine insert options dynamically. Unlike insert options rules, which apply when determining effective insert options for all items, insert rules apply only to the individual items in which you select them. Insert rules depend on definition items under the `/sitecore/system/Settings/Insert Rules` item that use the based `System/Branches/Insert Rule` data template. Configure insert rules where you declare insert options (click the Configure tab in the Content Editor, and then click Assign in the Insert Options group). For more information about insert options, see my blog post at `http://bit.ly/Hc4hta`.

Sorting Items

In addition to sorting items manually, you can select a rule for sorting the children of an item, sometimes called the subitem sorting rule (although sometimes the term *subitems* includes all descendants).

For each item, Sitecore stores a numeric sort order value in the `__Sortorder` field (`Sitecore.FieldIDs.Sortorder`) defined in the Appearance section of the standard template. Sitecore user interfaces, developer APIs, and XML representations order lists of items by their sort order values. When you invoke commands in the user interface to sort items, Sitecore calculates and applies new sort order values for the affected items (and potentially their siblings). In some cases, such as to control the visual order of sections defined in a data template and the base templates in its inheritance chain, you may need to view standard fields and set the value of the sort order field manually.

If you do not manually sort the children of an item, Sitecore automatically applies the sorting rule specified in the `__Subitems sorting` field. The default sorting rule sorts items by name, but you can select alternative child sorting rules, and even implement your own child sorting rules for each item.

www.it-ebooks.info

To select a child sorting rule for an item, do the following:

1. Click the Home tab in the Content Editor. Click the square icon next to the group name Sorting in the Sorting group. The Set the Sorting for the Subitems dialog appears as shown in Figure 2-3.

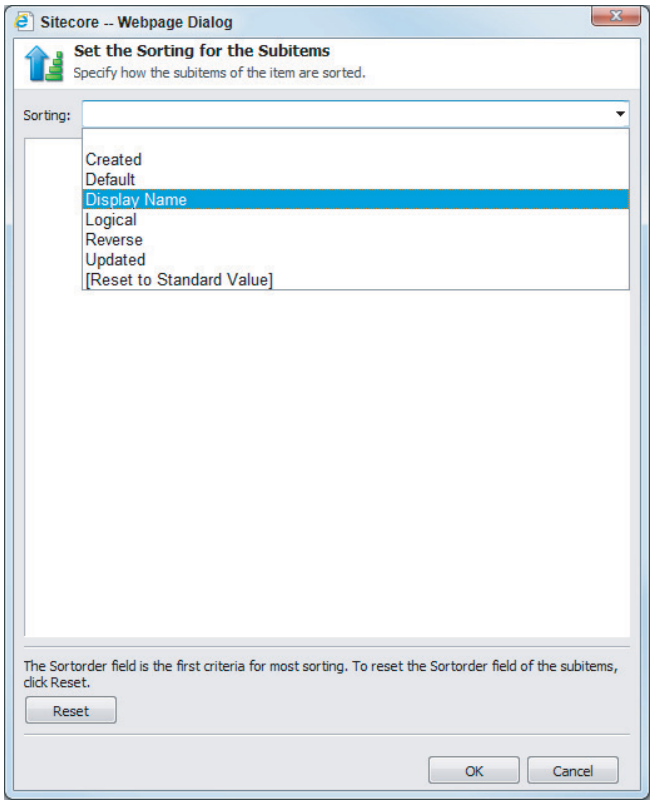


FIGURE 2-3

2. Select the desired child sorting rule in the Sorting field in the Set the Sorting for the Subitems dialog, and then click OK. The Set the Sorting for the Subitems dialog disappears, and you return to the Content Editor.



Remember to publish children, siblings, and parents after operations that change the sort order values of items, or when the order of items in the content delivery environment erroneously differs from the order of items in the content management environment.

For an example that implements the `System.Collections.Generic.IComparer<Sitecore.Data.Items.Item>` interface to sort items by a field value, and registers that comparer as a child sorting rule, see the FieldValueComparer (`http://bit.ly/yuboXs`) Sitecore Shared Source project.

www.it-ebooks.info

If you use .NET 3.5 or later, you can LINQ (Language-Integrated Query, http://bit.ly/woQJ2a) to sort items. In your code, add a using directive for the System.Linq namespace:

```
using System.Linq;
```

Then you can add .OrderBy() and .OrderByDescending() clauses to various types of lists. For example, you can sort by a property of the Sitecore.Data.Items.Item class, such as the display names of the children of the context item:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;

foreach(Sitecore.Data.Items.Item child in
 item.Children.OrderBy(x => x.DisplayName))
{
 // process child
}
```

You can also sort items by the value of a field:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;

foreach(Sitecore.Data.Items.Item child in
 item.Children.OrderByDescending(x => x["FieldName"]))
{
 // process child
}
```

For more information about sorting items, including how to sort with XSL and an XSL extension library for sorting, see my blog posts at http://bit.ly/ndc9no and http://bit.ly/oJTFXf. Read the comments on those blog posts for some warnings about sorting in multilingual Sitecore solutions.

Managing ItemURLs

While Sitecore provides default logic to determine and resolve URLs for items, you may need to implement custom URLs for your solution, such as to add query string parameters or resolve URLs with custom logic. There are two major considerations for managing URLs with Sitecore:

- Configuring the solution to generate URLs that meet your requirements
- Configuring the solution to correctly map those URLs to the corresponding items in the database

The first component typically involves the link provider specified by the type attribute of the /configuration/sitecore/linkManager/providers/add element named sitecore in the Web.config file. To override the link provider, create a class that inherits from the class specified by the type attribute of this <add> element, and update that attribute to reference your class. The second consideration typically involves a processor in the httpRequestBegin pipeline defined in the Web.config file that, among other things, parses URLs to determine the context item. You can add processors to and override existing processors in this pipeline to set the Sitecore.Context.Item static property as required for your solution.

www.it-ebooks.info

For more information about managing links with Sitecore, see The Sitecore Guide to Dynamic Links (`http://bit.ly/ri6Oww`).

How Sitecore Generates URLs

The default Sitecore link provider specified by the `type` attribute of the `/configuration/sitecore/linkManager/providers/add` element named `sitecore` in the Web.config file automatically generates URLs that assist in search engine optimization (SEO) efforts for each item based on its name and its path relative to the home item of the context site. If the `Rendering.SiteResolving` setting in the Web.config file is `true` and the item is not the home item of the context site or one of its descendants, Sitecore constructs the URL relative to the home item of the managed site that Sitecore determines to be an ancestor of the item. To construct the URL of an item, Sitecore removes the path to the start item of the managed site associated with the item. For example, under the default configuration, the URL of the `/sitecore/content/home/section/page` item would be `/section/page.aspx`.

Internal links in raw field values, such as in the source view of Rich Text fields, contain item IDs, which Sitecore refers to as *dynamic links*. Presentation components use constructs that invoke the `renderField` pipeline defined in the Web.config file to transform those references into friendly URLs. To convert IDs in field values to friendly URLs based on item paths explicitly, you can do any of the following:

- Use the `Sitecore.Web.UI.WebControls.FieldRenderer` web control.
- Invoke the `Sitecore.Web.UI.WebControls.FieldRenderer.Render()` static method.
- Call the `renderField` pipeline.
- Call the `Sitecore.Links.LinkManager.ExpandDynamicLinks()` static method (which does not add Page Editor, debugging, or other features, as the `renderField` pipeline does).

The default link provider specified by the `type` attribute of the `/configuration/sitecore/linkManager/providers/add` element named `sitecore` supports the following attributes:

- `addAspxExtension` — Determines whether to include the `.aspx` extension in the URLs of content items
- `alwaysIncludeServerUrl` — Determines whether to include the protocol, such as `http`, and domain, such as `www.domain.tld`, in URLs
- `encodeNames` — Determines whether to encode characters in item names according to the `/configuration/sitecore/encodeNameReplacements/replace` elements in the Web.config file
- `languageEmbedding` — Determines whether to include a language code in URLs
- `languageLocation` — Determines whether to use the first step in the URL path or the `sc_lang` query string parameter to specify the language code
- `shortenUrls` — Reserved for future use by Sitecore
- `useDisplayName` — Determines whether Sitecore uses the display names of items to construct URLs rather than using the names of those items

www.it-ebooks.info



Attributes of the link provider do not affect the URLs of Sitecore media items.

For an example that overrides the default link provider with the following features, see the LinkProvider (`http://bit.ly/AeRE0O`) Sitecore Shared Source project:

- Where one exists, use the alias for an item instead of its path.
- Include the trailing slash (`/`) character in URLs when possible.
- Convert all characters in the path part of the URL to lowercase.
- Include the language in URLs for some sites, but not others.

To determine the friendly URL of a content item, call the `Sitecore.Links.LinkManager.GetItemUrl()` static method. For example, to access the URL of the context item:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
Sitecore.Links.UrlOptions urlOptions =
 (Sitecore.Links.UrlOptions) Sitecore.Links.UrlOptions.DefaultOptions.Clone();
urlOptions.SiteResolving = Sitecore.Configuration.Settings.Rendering.SiteResolving;
string url = Sitecore.Links.LinkManager.GetItemUrl(item, urlOptions);
```



The `Sitecore.Links.LinkManger.GetItemUrl()` static method might not automatically apply the `Rendering.SiteResolving` setting defined in the Web.config file. If you set the `Rendering.SiteResolving` setting to `true`, follow the approach outlined in the preceding code. If you do not set the `Rendering.SiteResolving` setting to `true`, you do not need to create the `Sitecore.Links.UrlOptions` object and you can call the implementation of the `Sitecore.Links.LinkManager.GetItemUrl()` method that requires only one parameter (the item).

How Sitecore Resolves URLs

Processors in the `httpRequestBegin` pipeline defined in the Web.config file parse the requested URL to determine the context item. They do so in this order:

- `QueryStringResolver` resolves the `sc_itemid` query string parameter used to specify a context item when you invoke CMS user interfaces such as the Page Editor.
- `DynamicLinkResolver` resolves dynamic links that contain item IDs in case you forget to use the `renderField` pipeline.
- `AliasResolver` resolves aliases as described in a subsequent section of this chapter.
- `DefaultResolver` resolves requests for the home item of a managed site.
- `ItemResolver` resolves item paths.

www.it-ebooks.info



Sitecore data providers apply wildcards internally. A subsequent section of this chapter describes wildcards.

Removing /sitecore/content from URLs

If an item is not the home item of the context site or one of its descendants, or if the `Rendering.SiteResolving` setting in the Web.config file is `true` and the item is not the home item of any managed site or the descendant of such a home item, Sitecore generates its URL using the full path to the item. For example, under the default configuration, the URL of the `/sitecore/content/external_data/page` item is `/sitecore/content/external_data/page.aspx`, where you might prefer `/external_data/page.aspx` or simply `/page.aspx`.

The best way to address this issue is to ensure that you create all items for which you generate URLs as descendants of the home item of one of the managed sites, and if you manage multiple sites on a single instance, set the `Rendering.SiteResolving` setting in the Web.config file to `true`. For more information about managing multiple sites with individual Sitecore instances, see Chapter 9. If you must store items in a location not under the home item of any managed site or use some items in multiple sites, do either of the following:

- Create items for which you want URLs under the home items of managed sites, where those items contain fields to let CMS users select items not under the home item of any managed site. Configure presentation components for the items under the home items of the managed sites to retrieve data from the items selected in those fields.
- Implement a custom link provider to generate URLs according to your requirements. Add a custom processor after the default `ItemResolver` processor in the `httpRequestBegin` pipeline to set the `Sitecore.Context.Item` static property to the required item when that property is `null` and the requested URL corresponds to one of those items.

Clone Items

A clone functions like a virtual copy of an item; it can override field values in the item of which it is a clone. Clones use the same data templates as the cloned items. When you access a field value in a clone, Sitecore retrieves the value from the clone if it contains a value for that field, or from the cloned item if the clone does not contain a value for that field and the cloned item does, or from the standard values for the data template associated with the cloned item if neither the clone nor the cloned item contains a value for that field. You can even clone clones and inherit field values from the original cloned item. When you clone an item, Sitecore creates of each of its descendants as well.



You cannot clone a data template or data template section.

When you select a clone in the Content Editor, Sitecore displays a warning to inform you whether changes have occurred to the cloned item since the creation of the clone. The warning tells you of

www.it-ebooks.info

field changes as well as the addition, deletion, and renaming of descendant items under the cloned item. Using actions available in the warning, you can then accept the changes and apply them to the clone or reject the changes and keep the values and structure previously cloned. When you delete a cloned item, Sitecore converts its clones to actual items and copies field values from the deleted clone to the items themselves, which are then no longer clones.



Publishing expands clones in the Master database to create actual items in publishing target databases. The `IsClone()` *method of the* `Sitecore.Data.Items.Item` *class, which you can use to determine whether an item in the Master database is a clone, does not return* `true` *for items in publishing target databases.*

For clones, the `__Source (Sitecore.FieldIDs.Source)` field defined in the Advanced section of the standard template stores information about the cloned item. To prevent infinite recursion, clones do not inherit the value of the `__Source` field from the cloned item. Neither do standard values apply for the `__Source` field.

To create a clone, Sitecore uses the `CloneTo()` method of the `Sitecore.Data.Items.Item` class. This method resets the values of all fields in the clone to the values defined in the cloned item, except for those returned by the `GetNonInheritedFieldIDs()` method of the `Sitecore.Data.Items.CloneItem` class. In other words, clones effectively do not apply values for the fields defined by the standard template shown in Table 2-1.

TABLE 2-1: Fields Defined in the Standard Template

FIELD NAME	CONSTANT	SECTION
`__Updated`	`Sitecore.FieldIDs.Updated`	Statistics
`__Updated By`	`Sitecore.FieldIds.UpdatedBy`	Statistics
`__Revision`	`Sitecore.FieldIDs.Revision`	Statistics
`__Created`	`Sitecore.FieldIDs.Created`	Statistics
`__Created By`	`Sitecore.FieldIDs.CreatedBy`	Statistics
`__Workflow`	`Sitecore.FieldIDs.Workflow`	Workflow
`__Workflow State`	`Sitecore.FieldIDs.WorkflowState`	Workflow
`__Lock`	`Sitecore.FieldIDs.Lock`	Workflow

For more information about clones, see my blog post at `http://bit.ly/mRujMC`.

www.it-ebooks.info

Alias Items

You can use aliases to provide additional URLs for content items. An alias URL is typically shorter than the default friendly URL for an item, and often used for print campaigns or other forms of marketing. To create or manage aliases for an item in the Content Editor, click the Presentation tab, and then click Aliases in the URL group.

Aliases involve definition items under the `/sitecore/system/Aliases` item that use the `System/Alias` data template. If a requested URL matches the path to an alias definition item, the `AliasResolver` processor in the `httpRequestBegin` pipeline sets the context item to the item specified by the Linked item field in the Data section of that alias definition item.

Aliases support nesting. You can create the `/sitecore/system/Aliases/Section` item using the `Common/Folder` data template, and the `/sitecore/system/Aliases/Section/Page` item using the `System/Alias` data template to define the alias URL `/Section/Page.aspx` for an item elsewhere in the content tree.

CONSIDERATIONS FOR WORKING WITH ALIAS ITEMS

The following list describes some issues you need to address when working with aliases.

- Aliases can result in multiple URLs for an item, which can affect search engine ranking.
- Try to avoid aliases by putting the content in a location and naming it to generate the URLs you want in the first place. Partly for search engine optimization (SEO), I don't like the potential for a page to have multiple URLs. I have even stronger objections to creating multiple aliases for a single item unless it's necessary.
- When generating links to items, Sitecore does not apply aliases automatically. The LinkProvider (`http://bit.ly/AeRE0O`) Sitecore Shared Source project includes a link provider that applies aliases where they exist, but this can use inordinate system resources in solutions that contain large numbers of aliases.
- As with all items that you create in the Master database, you must publish alias definition items to the publishing target database(s) supporting the content delivery environment.
- Without a custom solution, all aliases apply to all managed sites, and each alias name can reference only a single item. If you create an alias pointing to an item under the home item of a specific managed site, that alias works for all the managed sites. Regardless of the context site, Sitecore sets the context item to the item specified by the alias, but with a different processing context for each managed site.
- When Sitecore processes an HTTP request that contains a URL that maps to an alias, Sitecore sets the context item to the item specified by the alias, not to the alias definition item itself.

www.it-ebooks.info

If you do not use aliases, you can disable aliases by setting the `AliasesActive` setting in the Web.config file to `false`, or by removing the `AliasResolver` processor from the `httpRequestBegin` pipeline.

Wildcard Items

Sitecore wildcard items match all HTTP requests that do not match the names of any siblings of the wildcard definition item. A wildcard is an item with a single asterisk (`*`) as the only character in its name. Otherwise, wildcard items are just like any other item. You can apply security, layout details, workflow, and other features to wildcard items. You can use them for various purposes, most commonly to integrate data from external systems, in which case the part of the URL matched by the wildcard typically corresponds to an identifier in the external system.

For example, if the `/products` item contains a wildcard item (`/products/*`) and no other children, Sitecore sets the context item to `/products/*` for any URL that would otherwise correspond to a child of the `/products` item, such as `/products/ProductID.aspx`. Presentation components specified for the wildcard item parse the requested URL (in the `Sitecore.Context.RawUrl` static property) to determine the requested `ProductID`, and render information about that product from the external system. If needed, wildcards support nesting, such as `/Products/*/*` to match URLs such as `/Products/Category/ProductID.aspx`. If multiple presentation components in a page depend on the `ProductID`, the first presentation control to parse the URL, or preferably a custom processor in the `httpRequestBegin` pipeline, can store the product identifier in the visitor's ASP.NET session, in the `Sitecore.Context.Items` static collection, or elsewhere for subsequent presentation components to access.



If you use the `Sitecore.Context.Items` *static collection, be sure to use a key that Sitecore will never use, and use programming facilities such as constants to avoid hard-coding that key in multiple components.*

Sitecore automatically sets the context item when the requested URL matches only the wildcard definition item, but your presentation components must generate links using appropriate URLs, such as to contain the product ID instead of the asterisk (`*`) character. In other words, you have to generate URLs that trigger the wildcards even though no items by those names exist at those paths in the Sitecore database.

For more information about wildcards, see my blog post at `http://bit.ly/p1C8RD`.

SITECORE DATA TEMPLATES

Data templates (or just templates) define the structure of types of items as a collection of named field values. For example, the data template for news article items could include fields for the title, byline, author, date, content body, and other data elements common to all news articles. Each data template field has numerous configuration properties, such as ID and data type. Each item based on a data template can contain a value for each field that template defines. Data templates can provide

www.it-ebooks.info

default ("standard") values for each field in the data template. Using base templates, data templates support multiple and sequential inheritance from other data templates. Most data templates eventually inherit from Sitecore's standard template, which defines sections and fields that can apply to any item.



Many CMS products use the term template to describe presentation components. Some CMS products use the term template for both data capture and presentation components. Sitecore uses it only for data capture components, and uses terms such as layout, layout details, and renderings to describe presentation components.

A data template consists of a definition item that defines properties of the data template. Each definition item for a data template can be the parent of any number of definition items for data template sections, which contain properties of the sections in that data template as described in the next section. Each section definition item can be the parent of any number of definition items for data template fields, which define properties of the fields in that section as described in the section subsequent to that which describes data template sections.

Each definition item for a data template can contain a standard values definition item, which defines values for the fields in that data template that apply to all items that do not specify values for those fields. The Template Builder application that you see when you select a data template in the Content Editor or the Template Manager abstracts the underlying implementation of sections, fields, and standard values, allowing you to specify basic properties of the template, its sections, and the fields within those sections through a simplified user interface. Figure 2-4 shows the Template Builder open in the Content Editor with the `Sample/Sample Item` data template selected in the content tree.

In the image shown in Figure 2-4, you can see the definition item for the standard values of the data template and a single section definition item containing two field definition items under the template definition item selected in the content tree on the left. The editing pane on the right shows the Template Builder, which simplifies access to those items. The Builder tab on the Ribbon, which appears when you navigate to a data template definition item, lets you perform operations on the object selected in the Template Builder, such as to remove or sort sections and fields, to set base templates for the selected data template, or to configure standard values for that template.



To edit the data template associated with an item, select the item in the Content Editor, click the Configure tab, and then click Edit in the Template group.

www.it-ebooks.info

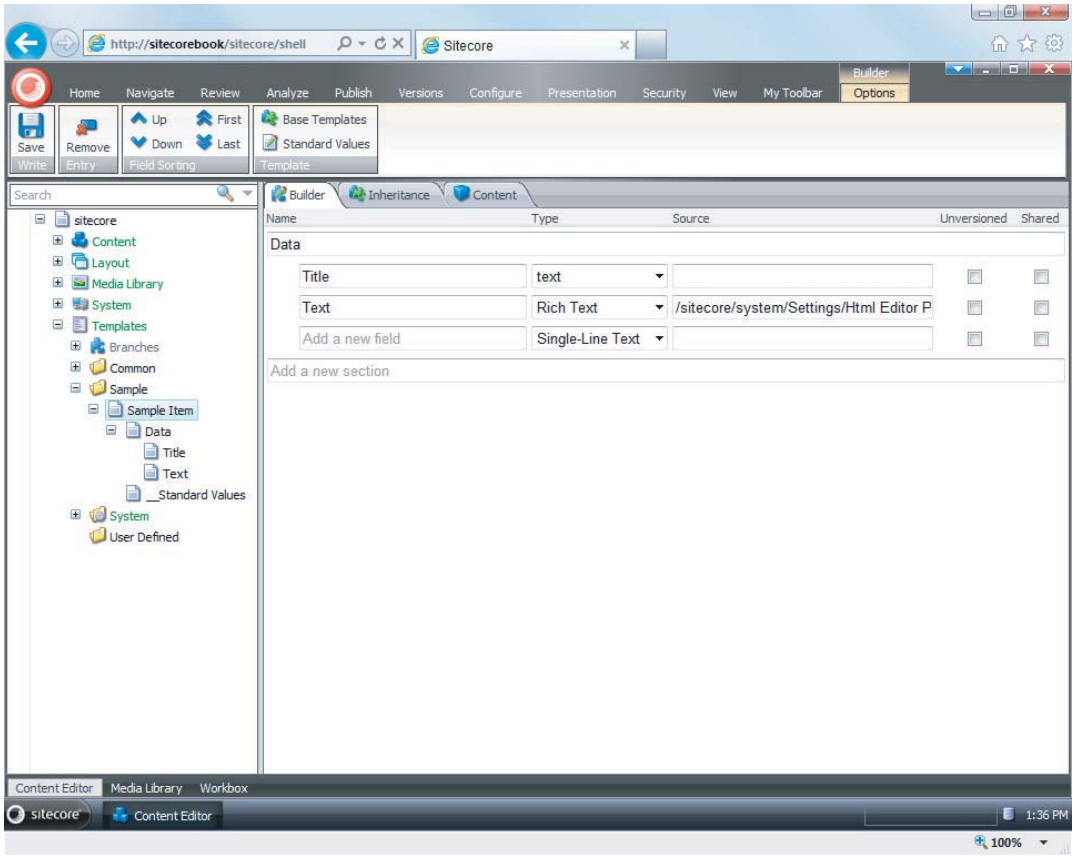


FIGURE 2-4

Data Template Sections

A data template consists of zero or more data template sections, which group the fields defined by the data template. A definition item for a data template can contain any number of children that represent data template sections, each section contains any number of fields. Data template sections simply collect and order fields for the following purposes:

- To group fields logically for data entry
- To make it easier for users to locate fields in items
- To avoid presenting a monolithic data entry form for CMS users
- To group fields for reuse by other data templates, such as base template with no items based directly on that base template

www.it-ebooks.info



Sort the definition items for fields that CMS users access most frequently to the top of each section, and sort the definition items for sections that users access most frequently to the top of each data template.

When a data template and one or more of its base templates contain a section by the same name, Sitecore user interfaces show a single section containing all of the fields defined by the data template and in all sections of its base templates that define sections by that name.



Base templates are simply data templates inherited by other data templates.

Sitecore developers write code that accesses fields directly by name or ID, and refers to data template sections only in very specific circumstances. For example, sections are relevant to developers ordering fields defined in the same section of multiple base templates, using the Sitecore APIs instead of the Sitecore user interfaces to define data templates. Sections are irrelevant when retrieving field values from an item.

Data Template Fields

Each data template section can contain any number of data template fields, which define the CMS user interface component for editing values and the storage formats for those values. You can apply a number of properties to each field to control its features, such as to support translation and versioning of the field value.

Name data template fields to help users determine the purposes of your fields, and supply additional properties such as help text to provide additional contextual information. If the name of the field is not friendly to users, set the Title property of the field.



To avoid confusion, do not use the same field name twice in a data template, including its base templates, even if that field name would appear in separate sections.

In some cases, the names of data template fields appear in code. You may want to apply a naming convention for data template fields, such as to exclude whitespace characters. In such cases, apply titles to data template fields for greater usability.

Data Template Field Types and Categories

The data type of each data template field, referred to as its field type, controls the user interface component that Sitecore displays for the field in CMS user interfaces. For example, the Single-Line Text field type presents a user interface that allows the CMS user to enter a single line of text, while

www.it-ebooks.info

the Rich Text field type presents a WYSIWYG (What You See Is What You Get) HTML editor. Field types also control the format of the field value represented by Sitecore, as well as how Sitecore manages internal links contained in fields of that type. Excluding the Attachment field type used for the binary component stored in the database for media items, all field values contain text. For some field types, that text is in a specific format, such as XML or XML-escaped HTML.

Sitecore defines the field types available to data templates using items based on the `System/Templates/Template field type` data template under the `/sitecore/system/Field types` item in the Core database. Each definition for a field type can have children that define commands visible for the field in the Content Editor and the Page Editor. Definition items for field types can use the Assembly and Class fields in the Data section to designate the class that implements the user interface for the field type. Alternatively, if a definition item for a field type contains a value in the Control field in the Data section, the prefix before the colon (`:`) character matches the `prefix` attribute of one of the `/configuration/sitecore/controlSources/source` elements in the Web.config file, where the `namespace` attribute of that `<source>` element identifies the .NET namespace that contains the class that implements the user interface for the field type, and the token after the colon indicates the name of that class.

The `/App_Config/FieldTypes.config` file maps the keys (lowercase names) of the data template field types to the classes that implement internal link management for those field types. If you define your own field types that support internal links, you should consider adding entries to this file.

Sitecore provides numerous field types, organized into the following categories. For an example data template that demonstrates many of these field types, use the Installation Wizard tool on the Development Tools submenu of the Sitecore menu in the browser-based desktop to install the `User Defined/Field Types` data template. For instructions to install a package, see The Sitecore Package Designer Administrator's Guide (`http://bit.ly/zDvwnv`). To examine the fields it contains, you do not need to create an item based on this data template. Instead, navigate to the definition item for its standard values (`/sitecore/templates/User Defined/Field Types/__Standard Values`) in the Content Editor or the Template Manager. The package (`.zip` file) that you installed contains this item. You will read more about standard values later in this chapter.



The definition item for the standard values of data templates is simply an item based on that data template stored as a child of the definition item for the data template, where the name of that child is `__Standard Values`.

Analytics

The field types in the Analytics category support features of the Customer Engagement Platform (CEP) provided by the Digital Marketing System (DMS). This book does not describe these field types because you should not use them in your own data templates.

Simple Types

The field types in the Simple Types category implement typical data entry components such as Checkbox, Date, Datetime, Image, and File, Integer, Multi-Line Text, Rich Text (HTML), and Single-Line Text.

www.it-ebooks.info



The Image field type lets you select an image from the media library; the File field type lets you select any file, such as a `.pdf`.

Sitecore stores values for most of these field types as simple text, using the ISO (International Organization for Standardization) format `yyyyMMddTHHmmss` for dates, and XML elements for images and files. For an explanation of the characters in date format strings, see `http://bit.ly/Ajmy6f`.



Depending on how you access field values, they may contain XML entities in place of special characters, such as `&` *in place of an ampersand (&) character.*

List Types

Most of the field types in the List Types category let the user select one or more items, while the Name Lookup Value List and Name Value List types allow the user to enter key/value pairs. With a Name Value List, the CMS user enters both keys and values. With a Name Lookup Value List, the CMS user enters keys and selects an item from a drop-down list to associate with each of those keys.

The Droplist and Grouped Droplist field types allow the user to select a single item and store the name of that item. Droplist allows the user to select a child of the item specified by the Source property of the field; Grouped Droplist allows the user to select a grandchild of the Source item, where the children of that source item represent categories containing those items.



To avoid weak references, use field types that store IDs rather than names. For example, use the Droplink field type, which stores the ID of the selected item, rather than the Droplist field type, which stores only its name. Whenever you present data to a user, include item paths instead of or in addition to IDs.

The Grouped Droplink field type in the List Types category, as well as the Droplink and Droptree field types in the Link Types category that follows, allow the user to select an item. Sitecore stores the ID of the selected item as the value of the field.

The Checklist, Multilist, Treelist, and TreelistEx field types allow the user to select zero or more items. Sitecore stores a list containing the IDs of those items separated by pipe (|) characters.



Of the field types that allow the user to select more than one item, the TreelistEx field type performs best and has as wide a range of capabilities as any other field type.

www.it-ebooks.info

Link Types

The Link Types category of contains field types that can store a reference to a single Sitecore item, a single external URL, or another type of link such as an anchor within the page or an email address.

The Droplink field type allows the user to select a single item from a drop-down list, and stores the ID of that item.

The Droptree field type allows the user to select a single item from a tree structure, and stores the ID of that item.

The General Link field type allows the user to specify an internal, external, JavaScript, anchor, or email address, and stores that value and other properties of that link as an XML element.

Sitecore clones use the Version Link field type, which stores a reference to a database, item ID, language, and version number in a proprietary format.

Developer Types

The Developer Types category contains field types intended for Sitecore developers.

The Icon field type allows the user to select an image from Sitecore's icon library, and stores that value as a relative path.



Sitecore provides an extensive icon library in the `/sitecore/shell/Themes/Standard` *subdirectory under the document root of the IIS website hosting the instance. You can use these images royalty-free on the sites that you manage.*

The IFrame field type enables you to embed custom applications as fields in data templates. You define the storage format for the IFrame field type. For more information about the IFrame field type, see The Sitecore Client Configuration Cookbook (`http://bit.ly/qS8Dc1`).

The Tristate field type allows the user to specify that a value is Yes (true), No (false), or Default (undefined).

System Types

Other than the File Drop Area and possibly Rules field types, the field types in the System Types category are primarily for the Sitecore application itself, and not intended for your data templates. You can use these field types in your own data templates, but their details are beyond the scope of this book.

The File Drop Area field type uses Web-Based Distributed Authoring and Versioning (WebDAV — see `http://www.webdav.org`) to associate media with content items.

The Rules field type lets you use the rules engine in custom processing contexts.

Deprecated Types

Sitecore includes the field types in the Deprecated category for internal use, but does not provide technical support for their use in your solutions.

www.it-ebooks.info



Do not use the Deprecated data template field types.

Custom Data Template Field Types

In addition to overriding the default field types, you can implement custom field types. Overriding a field type is not a trivial task and is beyond the scope of this book. For more information about custom data template field types, see Creating a Composite Custom Field (`http://bit.ly/oS8PXq`) on the Sitecore Developer Network (SDN). For an example of a custom data template field, see my blog post at `http://bit.ly/ntte5Z`, as well as the following Sitecore Shared Source projects:

- **VideoEmbed** — Lets CMS users specify the URL of a video (`http://bit.ly/fDPhQA`)
- **ColorPicker** — Lets CMS users select HTML color codes from a visual palette (`http://bit.ly/zbnHj8`)
- **FieldTypes** — Includes Visual List, Carousel, Text List, Limited Single-Line Text, Slider, and Filtered Multilist custom field types (`http://bit.ly/wClS70`)
- **AccountSelectorField** — Lets CMS users select users and roles (`http://bit.ly/4UzRih`)
- **InlineItemListField** — Creates items without leaving a Multilist field (`http://bit.ly/yCQkKr`)



The IFrame field type can often provide the same functionality as a custom field type, and presents less of a learning curve for CMS developers.

Data Template Field Properties

You can specify a number of field properties to control how users and code interact with each field and its values.

Sitecore uses definition items to describe the fields in each data template. Field definition items are like any other items, and therefore have names, IDs, paths, and other universal properties. Your code can refer to fields by name, but to follow best practice, reference fields by ID, such as by creating a class similar to the `Sitecore.FieldIDs` class that exposes the IDs of fields provided by the CMS.

If you specify the Title property of a field, CMS user interfaces display the value of that property as the label for the field. Otherwise, CMS user interfaces display the name of the field.

The Source property works differently for different types of fields. The Source property does not apply to some field types, such as simple text fields. For list fields, the Source property specifies the items to appear in the selection list. For fields of type Image and File, the Source property specifies the root of the tree for item selection dialogs. For Rich Text fields, the Source property can specify a Rich Text Editor (RTE) profile definition item, which controls the features visible in the editor.

www.it-ebooks.info

If the Source property of a field of type Image or File begins with a tilde character (~, probably located below the Esc key on your keyboard), as in (for example) `~/sitecore/media library/images`, then the selection tree shows the specified location by default, but allows the user to navigate to the root of the tree. For an example of setting the Source property dynamically at runtime, see my blog post at `http://bit.ly/p8q52j`. The Source field of these and other field types allow a relative path, which specifies an location relative to the item selected by a user. For example, if the value of the Source property is an XPath shortcut for the parent element, and a user edits the `/sitecore/content/home/page` item, that source path references the `/sitecore/content/home` item; when editing a different item that is not a sibling of the `/sitecore/content/home/page` item, the Source property of the same field could evaluate to a different item. These three approaches can increase data template reusability and inheritance options, such as when you use the same data template for multiple managed sites. For more information about managing multiple websites with a single Sitecore instance, see Chapter 9.

The Source property of a selection field can specify a path, multiple paths separated by pipe characters, a query beginning with the `query:` prefix, a fast query beginning with the `query:fast:` prefix, or multiple parameters encoded much like a URL query string as described later in this section). For more information about Sitecore query and fast query, see Chapter 3. Sitecore invokes the `getLookupSourceItems` pipeline defined in the Web.config file to determine the items to include in the drop-down list or selection tree for fields that show such features. My blog post at `http://bit.ly/q7oGx8` provides an example processor that let you use the `getLookupSourceItems` pipeline that lets you use tokens such as `$now` and `$id` in the Source property of fields to construct queries that specify the item selected by the user and the current system date.

Where possible, all selection field types, including Checklist, Droplist, Grouped Droplink, Grouped Droplist, Multilist, Name Lookup Value List, Treelist, and TreelistEx, support multiple parameters in the Source property of the data template field, encoded and separated as query string parameters. These parameters include:

- `AllowMultipleSelection` — Allows the user to select an item more than once (allowed values are `yes` and `no`)
- `DatabaseName` — Names the database containing the items to include in the list or display in the selection tree
- `DataSource` — Specifies a path that would have been the Source property for the field if you had no other parameters to pass



Under the default implementation of the `getLookupSourceItems` *pipeline, the* `DataSource` *parameter must specify an item or a pipe-separated list of items, not a query or a fast query.*

- `ExcludeItemsForDisplay` — A comma-separated list of IDs; the specified items do not appear in the drop-down list or selection tree
- `ExcludeTemplatesForDisplay` — A comma-separated list of template names; items based on these data templates do not appear in the drop-down list or selection tree

www.it-ebooks.info

- `ExcludeTemplatesForSelection` — A comma-separated list of template names; the user cannot select items based on the specified data specified
- `IncludeItemsForDisplay` — A comma-separated list of template IDs; the items they specify appear in the drop-down list or selection tree
- `IncludeTemplatesForDisplay` — A comma-separated list of template names; items based on these data templates appear in the drop-down list or selection tree
- `IncludeTemplatesForSelection` — A comma-separated list of template names; the user must select an item based on one of the data templates specified



When a drop-down list or selection tree does not display an item, it does not display its descendants either.

The `__Masters` (`Sitecore.FieldIDs.Branches`) field defined in the Insert Options section of the standard template provides a complex example of Sitecore's internal use of these parameters. While you should normally assign insert options using the Assign command in the Insert Options group on the Configure tab, you can also assign insert options using this field of type TreelistEx. The Source field of the `/sitecore/templates/System/Templates/Sections/Insert Options/Insert Options/__Masters` item that defines this data template field uses the `DataSource` parameter to root the tree at the `/sitecore/templates` item, the `IncludeTemplatesForSelection` parameter to allow selection of definition items for data templates, branch templates, and command templates, the `IncludeTemplatesForDisplay` parameter to navigate a variety of types of folders, and the `ExcludeItemsForDisplay` parameter to prevent certain items from appearing in the tree.

To access the `__Masters` field as a standard TreelistEx, click the View tab in the Content Editor, and then select the Standard Fields checkbox in the View group. Then, click the Edit command above the field titled Insert Options (the Title property of the `__Masters` field) in the Insert Options section. The Select Items dialog appears, allowing you to select definition items for branch templates, command templates, and data templates. This selection tree provided by the Select Items dialog allows you to expand folders to navigate the tree, but not to add those folders to your selection. Additionally, the Source property of this field excludes the data templates for section definition items from the selection tree, which prevents you from expanding data templates to view the sections they contain or selecting those section definition items. It also specifies the types of items that you can select, which prevents you from selecting the standard values definition items beneath the definition items for those data templates. Remember to clear the Standard Fields checkbox in the View group on the View tab before proceeding.

For shared fields those for which you check the Shared checkbox in the definition item for the field, Sitecore maintains a single value for all versions of each item in all languages. For fields that you choose not to version by checking the Unversioned checkbox in the definition item for the field, Sitecore maintains a single value for all versions of each item in a language, but different values for different languages. Sitecore does not version field values if you select either the Shared or the Unversioned checkbox in the definition item for the field.

www.it-ebooks.info



Publishing operations that include a version of an item include all values in shared fields and fields of the published language that you do not version, regardless of the workflow state or publishing restrictions associated with other versions of that item.

The Reset Blank property in each data template field controls whether Sitecore resets that field to its standard value when a user enters an empty string in that field. If you do not check the Reset Blank checkbox (as by default), Sitecore stores an empty string in the field when the user clears the value of that field. If the Rest Blank property is checked, Sitecore instead resets the field to its standard value.

Standard Values

Standard values define field values that apply to all items based on a data template that do not override those values. All items based on a data template automatically contain its standard values, but each item can override standard values for individual fields.

You can create a standard values item for each data template. The standard values item for a data template is just an item based on that data template, stored as a child of the data template definition item, with the name `__Standard Values`.

To create or edit the standard values item for a data template, do the following:

1. Select the data template in the Content Editor or Template Manager.
2. Click the Options tab, and then click Standard Values in the Template group.

Standard values items inherit standard values defined in base templates, recursively. For example, if template B inherits from template A, and template A defines a standard value for a field, items based on template B share that standard value for that field, unless those items or standard values for B override the standard value defined for that field by Template A. If A inherits from another template, the standard values for that template apply to A and hence B and any items associated with B.

Just as clones do not inherit the value of the `__Source` (`Sitecore.FieldIDs.VersionLink`) field defined in the Advanced section of the standard template from the cloned item, items do not inherit the value of the `__Source` field from the standard values of the data template associated with those items. Standard values apply to all other fields, but item creation and save operations often override standard values for the same fields that Sitecore resets for clones (previously listed in Table 2-1).

Always define at least layout details (see Chapter 3), insert options, and initial workflow (see Chapter 9) in the standard values for each data template. If you need to override standard values for a large number of items, you can create a data template that inherits from the existing template, apply alternate standard values for that inheriting data template, and associate those items with that inheriting template.



Set the icon for a data template in the data template itself, not in its standard values.

www.it-ebooks.info

When you create an item, Sitecore expands the following tokens in the standard values for the data template, resulting in field values in the item that override the standard values for those fields:

- `$name` — The response from the user to the prompt to name the new item
- `$id` — The ID of the new item
- `$parentid` — The ID of the parent of the new item
- `$parentname` — The name of the parent of the new item
- `$date` — The system date in `yyyyMMdd` format
- `$time` — The system time in `HHmmss` format
- `$now` — The system date and time in `yyyyMMddTHHmmss` format

You can always determine the values for these properties of the item at runtime, but you may want to use these tokens to store the values at creation time, in case the user moves or otherwise changes the data. For example, the `__Created` (`Sitecore.FieldIDs.Created`) field defined in the Statistics section of the standard template stores the creation date and time for each item. Your presentation code can use this field, for example as the release date of a news article, but the user cannot easily change its value. If you create a Datetime field in your data template, and set its standard value to `$now`, Sitecore will set that field as it sets the `__Created` field defined in the standard template. Unlike the `__Created` field that most users cannot update, CMS users can subsequently change the value of your Datetime field like any other field. Regarding the `$name` token, you may want to store the name originally entered by the user, perhaps using the display name property of the item, in case the user or the system subsequently changes the name of that item.

To add your own expanding tokens for use in standard values, do the following:

1. Create a class that inherits from the class specified by the `MasterVariablesReplacer` setting in the Web.config file in your Visual Studio project.
2. Override the `Replace()` and `ReplaceField()` methods to call the corresponding methods in the base class and then perform your replacements.
3. Update the `MasterVariablesReplacer` setting in the Web.config file to specify the signature of your class.

If the value for a field in a version of an item is null, Sitecore retrieves the standard value for that field. In this context, Sitecore differentiates between null (equivalent to a missing row in the database) and an empty string. An empty string is a value like any other. A null value specifically indicates that a field should contain its standard value.

To reset a field to its standard value, first select the item containing the field in the Content Editor.



For clones, these operations reset the field to the current value in the cloned item.

www.it-ebooks.info

To reset the `__Renderings` (`Sitecore.FieldIDs.LayoutField`) field defined in the Layout section of the standard template, which specifies layout details, to the standard value defined for the item's data template or by the cloned item, follow these steps:

1. Select the Presentation tab, and then click Reset in the Details group. A confirmation dialog appears.
2. Click OK. The confirmation dialog disappears and you return to the Content Editor.

To reset the `__Masters` (`Sitecore.FieldIDs.Branches`) field defined in the Insert Options section of the standard template, which defines insert options, to the standard value defined for the item's data template or by the cloned item, follow these steps:

1. Select the Configure tab, and then click Reset in the Insert Options group. A confirmation dialog appears.
2. Click OK. The confirmation dialog disappears and you return to the Content Editor.

To reset the `__Subitems Sorting` (`Sitecore.FieldIDs.SubitemsSorting`) field defined in the Appearance section of the standard template, which defines the child sorting rule, to the standard value defined for the item's data template or by the cloned item, follow these steps:

1. Select the Home tab, and then click the square icon next to the group named Sorting on the Ribbon. The Set the Sorting For The Subitems dialog appears as shown previously in Figure 2-3.
2. Select Reset to Standard Value in the Sorting drop-down list and then click OK. The Set the Sorting For The Subitems dialog disappears and you see the Content Editor.



In the Set the Sorting For The Subitems dialog, the Reset button resets the sort order values of the children of the selected item according to the sorting rule currently defined in the selected item.

To reset any other field to the standard value defined for the item's data template or by the cloned item, follow these steps:

1. If the standard template defines the field, select the View tab in the Content Editor, and then select the Standard Fields option in the View group.



If you show standard fields, remember to hide them afterward to improve performance and usability, as well as to make an inadvertent error in one of those fields less likely.

2. Click the Versions tab and then click Reset in the Fields group. The Reset Fields dialog appears as shown in Figure 2-5. The fields left side of this dialog shows the current values of

www.it-ebooks.info

fields in the item; the right side shows the standard values for those fields. If you elected to show the fields defined in the standard template, they appear in the Reset Fields dialog.

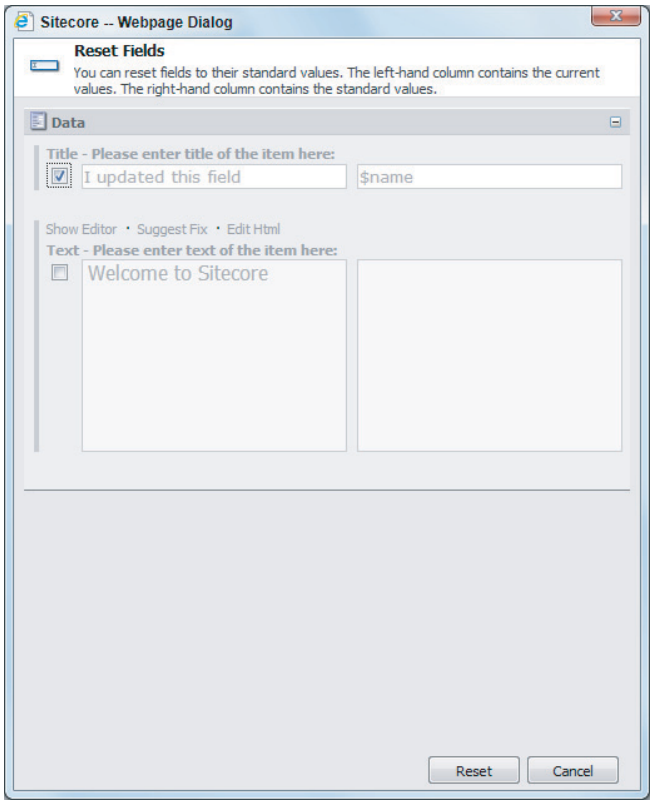


FIGURE 2-5



If you reset a field to its standard value, and that standard value contains a token such as `$name`*, the value of the field will contain that token rather than the value expanded when you originally created the item*

3. Check the checkboxes for the fields to reset in the Reset Fields dialog, and then click Reset. The Reset Fields dialog disappears and you return to the Content Editor.
4. If the standard template defines the field, select the View tab, and then clear the Standard Fields checkbox in the View group.



To override the logic that Sitecore uses to determine standard values, override the class specified by the `type` *attribute of the* `/configuration/sitecore/standardValues/provider/add` *element named* `sitecore` *in the Web.config file.*

www.it-ebooks.info

The `ContainsStandardValue` property of the `Sitecore.Data.Fields.Field` class that represents a field value indicates whether a field contains its standard value. For example, to determine whether the context item contains the standard value for the field named Title, do the following:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
Sitecore.Data.Fields.Field title = item.Fields["Title"];

if (title.ContainsStandardValue)
{
 // the context item contains the standard value for the field named Title.
}
```

Data Template Inheritance

Each data template can inherit sections, fields, standard values, and potentially other properties from any number of other data templates, called *base templates*. If you directly or indirectly inherit from the same data template more than once, Sitecore behaves as if you had inherited that data template only once. If you do not specify any base templates for a data template, that template inherits directly from the standard template by default.



Avoid circular template inheritance. A data template should never inherit directly or indirectly from itself.

If a data template inherits from two base templates that define different standard values for a field inherited from a common base template, the standard value for that field in the template that inherits from those two base templates may contain either value.

The Standard Template

The standard template defines sections and fields common to all items. If a template does not explicitly inherit from no template (the data template with the null ID, `{00000000-0000-0000-0000-000000000000}`), one of the base templates of the standard template, or a data template that does not inherit from the standard template, that template inherits from the standard template by default. The standard template and the data templates for definition items for data templates, data template sections, and data template use these features to avoid inheriting from the standard template.

Many of the commands in the Ribbon visible in the Content Editor abstract access to fields defined in the standard template, helping to avoid a long data entry form for users and providing simplified user interfaces instead of those provided by the underlying field types. For example, the Details command in the Layout group on the Presentation tab abstracts access to the `__Renderings` (`Sitecore.FieldIDs.LayoutField`) field defined in the Layout section of the standard template.

www.it-ebooks.info



Not all fields defined by the standard template logically apply to all items in which they appear. For example, layout details typically apply only to items under the home item of one of the managed sites, but you can apply layout details to any item. It is your responsibility to use the fields defined by the standard template correctly.

Technically the standard template does not define any fields. Instead it specifies a number of base templates, each of which defines a different section that contains several fields. These base templates inherit from the null template.

The following may be confusing and irrelevant, or it may be useful and interesting. Sitecore uses items based on data templates to define data templates and their components. A data template definition is a hierarchy of items rooted at a template definition item, which may contain a standard values item based on the data template as well as some number of section definition items, each containing field definition items. This means that Sitecore uses data templates to define the structure of definition items for data templates, section definition items, and field definition items. As mentioned previously in this section, these data templates do not inherit from the standard template or the null template, but only from a subset of the base templates used by the standard template that are useful for these types of definition items. Additionally, the standard template does not inherit from itself.

To specify that a data template does not inherit from the standard template or any other data template, do the following:

1. Select the definition item for the data template in the Content Editor or Template Manager, and then select the Content tab.
2. Select the View tab, and then check the Raw Values checkbox in the View group.



Remember to hide raw values afterward to improve usability.

3. Enter {00000000-0000-0000-0000-000000000000} (the null ID) as the value of the `__Base template` field in the Data section as shown in Figure 2-6.

To inherit only specific sections from the standard template, in the `__Base template` field of the data template, select one or more data templates from the `/sitecore/templates/System/Templates/Sections` folder that contains the base templates for the standard template.

To enhance the standard template, you can create a data template that inherits from the standard template and then update the `DefaultBaseTemplate` setting in the Web.config file to specify that template. Alternatively, create a data template that inherits from the null template and add that to the base templates for the default standard template.

www.it-ebooks.info

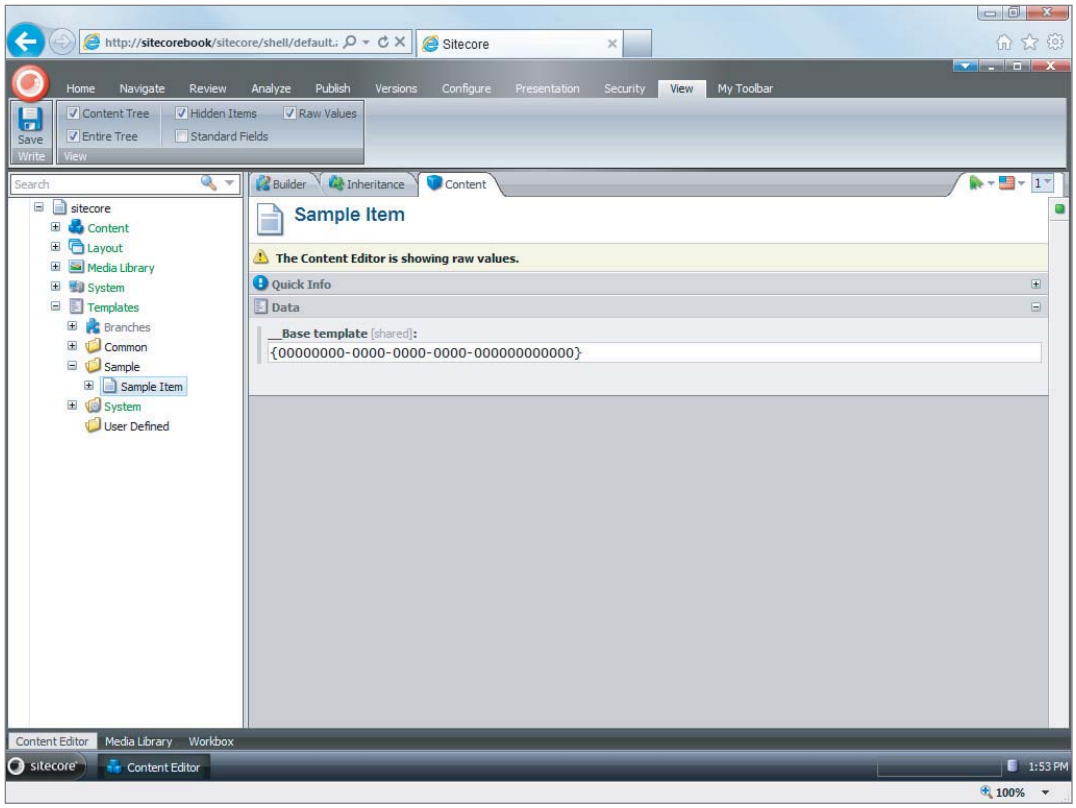


FIGURE 2-6

DATA VALIDATION

Sitecore provides extensive data validation facilities for field values and entire items. You can use these features to enforce constraints on data entered by CMS users. Using validation, you can define data entry requirements for individual field values, as well as for entire items and even hierarchies of items, meaning that you can validate almost anything. You can validate the name and location of each item. You can validate data while the user enters it with validation indicators appearing to the left of individual fields, in the Quick Action bar to the left of the Content Editor, and in the Validation bar to the right of the Content Editor. You can validate data when CMS users click the Validate command in the Proofing group on the Review tab in the Content Editor. You can use validation to prevent the user from submitting content to the next state in the workflow until before addressing a validation issue. You can provide validation actions that let the user correct validation issues automatically. You can specify different validation error levels to indicate the severity of the error and control how the system responds to it. You can suppress specific types of validation for individual items.

www.it-ebooks.info

OPTIMIZING DATA VALIDATION

The following tips and techniques can help you benefit from using data validation. For more information about validation, see Chapter 7. Avoid the legacy Validation and ValidationText properties in the Data section of data template field definition items.

- To maximize performance, configure validation for the Validate command and workflow rather than using the Quick Action bar or the Validation bar.
- Content Editor warnings can provide an alternative to validation. For information about Content Editor warnings, see Chapter 7.
- To disable the requirement that fields of type Image specify alternate text (ALT text), remove the Image Has Alt Text validator from the fields in the Validation section of the `/sitecore/system/Settings/Validation Rules/Field Types/Image` item in the Core database.
- To disable XHTML validation for Rich Text fields , remove the Is Xhtml validator from the fields in the Validation section of the `/sitecore/system/Settings/Validation Rules/Field Types/Rich Text` item in the Core database.
- To disable full page XHTML validation, remove the Full Page XHtml validator from the fields in the Validation section of the `/sitecore/system/Settings/Validation Rules/Global Rules` item in the Core database. Alternatively, disable (move or delete) the validation action from the workflow action the user chooses to submit content to the next state in the workflow. For information about workflow, see Chapter 9.
- In some versions of Sitecore the definition item for the Rich Text field type (the `/sitecore/system/Settings/Validation Rules/Field Types/Rich Text` item in the Core database) has the erroneous display name Multilist in multiple languages. If you see two children under the `/sitecore/system/Settings/Validation Rules/Field Types` item that appear as Multilist, click on each one until the item header shows Multilist [Rich Text], and clear the display name property for that item in all languages.

For more information about validation, see The Sitecore Data Definition Reference (`http://bit.ly/nmGuiB`), The Sitecore Client Configuration Cookbook (`http://bit.ly/qS8Dc1`), and my blog post at `http://bit.ly/pxabIE`, which links to a number of resources that include sample validators.

MANAGING MULTILINGUAL SOLUTIONS

Sitecore lets you translate your managed sites into any number of languages for any number of regions and cultures. Your site visitors from multiple regions and cultures often speak a single language with various distinctions. You can vary content within a single language to present specific variations of content to visitors from specific regions and cultures. Specifically, you can translate each field of each

www.it-ebooks.info

item for any number of language and region/culture combinations, and present content from one or more language and region/culture combinations on each page according to your requirements.

Because the Sitecore user interfaces are components of a Sitecore solution, they support internationalization just as your solution supports internationalization. To select a language to display in CMS user interfaces, on the Sitecore login screen, client Options to display the Options panel, and then select a language from the User Interface Language drop-down list.



You cannot translate the value of shared data template fields.

Language and Culture

Sitecore maintains a code for each language, for example `en` for English. In addition to differentiating content by language, you can also differentiate content by culture or region, for example by using the code US for the United States and GB for Great Britain. While the default `en` language provided by Sitecore does not specify a culture, Sitecore associates a culture with any language that you create. Together, the language and culture create the combined country code, for example `en-US`.

To change the flag associated with the default English language, do the following:

1. Select the `/sitecore/system/Languages/en` item in the Content Editor.
2. Click the flag that represents the icon for the item at the top of the editing pane. The Assign Icon dialog appears.
3. Select the Icons tab.
4. Select Flags from the drop-down list at the right. Figure 2-7 shows the Assign Icon dialog as it appears while making this selection.
5. Select the flag that the Sitecore user interfaces should display for the English language, and then click OK. The Assign Icon dialog disappears and you return to the Content Editor.

Optionally, in the `/sitecore/system/Languages/en` item that defines the default English language, enter the name of an alternate spell-checking dictionary file, such as `en-GB.tdf` for the English language in the Great Britain region/culture, for the value of the Dictionary field in the Data section. You can find these files in the `/sitecore/shell/Controls/Rich Text Editor/Dictionaries` subdirectory beneath the IIS document root. For more information and additional `.tdf` files that you can install, see `http://bit.ly/wDwpmf`.



You may prefer to leave the default English language alone and register new English languages for other cultures as described previously in this chapter.

If you implement multiple languages, you should always let the user select from the available content languages to override the current context language.

www.it-ebooks.info



FIGURE 2-7

Determining the Context Language

The `LanguageResolver` processor in the `httpRequestBegin` pipeline determines the context language (the `Sitecore.Context.Language` static property) for the duration of each request. Subsequently, Sitecore accesses versions of items in that language by default. The `Sitecore.Context.Language` provides a value even before the layout engine invokes the `httpRequestBegin` pipeline for the HTTP request. Under some conditions, the value of the `Sitecore.Context.Language` static property differs before and after invocation of the `LanguageResolver` processor.



Within Sitecore CMS user interfaces such as the desktop and the Content Editor, the context language is the CMS client language, and the content language (the `Sitecore.Context.ContentLanguage` *static property, also called the current language in this book) is the language of content edited within that user interface.*

A URL can specify the context language in at least the following ways, where that specification matches the name of a language definition item under the `/sitecore/system/Languages` item in the context database:

- From the `sc_lang` query string parameter, such as `sc_lang=en` for the default region-neutral English language

www.it-ebooks.info

- From the path, such as `/en/section/page.aspx` for the default English
- From the language associated with the context site (the `language` attribute of the corresponding `/configuration/sitecore/sites/site` element in the Web.config file) triggered by the requested URL

If the requested URL does not specify a language using a the `sc_lang` query string parameter or the first step in the path, and the `language` attribute of the `/configuration/sitecore/sites/site` element in the Web.config that represents the context site does not specify a value for the `language` attribute, Sitecore sets the context language to the value specified by the `DefaultLanguage` setting in the Web.config file.

You can configure multiple managed sites with different values for the `hostName` and `language` attributes of the `/configuration/sitecore/sites/site` elements in the Web.config file. If you do this, set the `languageEmbedding` attribute of the `/configuration/sitecore/linkManager/providers/add` element named `sitecore` in the Web.config file to `false` to prevent Sitecore from ever including the language in URLs. Otherwise, set the value of that `languageEmbedding` attribute to `always` to prevent Sitecore from generating multiple URLs for a single item (with and without the language in the path or query string). To allow the user to select a language, you must provide links or other features in the page that trigger the appropriate URLs for those languages.

Sitecore can use a cookie to store the user's language selection. For example, if the URL of an incoming HTTP request specifies a language in the path or query string, Sitecore sets that language as the context language, and issues a cookie to the client to store that language selection so that links to other pages on the site do not need to indicate the language.

You can use additional techniques to determine the user's preferred language, typically in a pipeline processor to precede, replace, or follow the default `LanguageResolver` process in the `httpRequestBegin` pipeline. For example, you can do the following:

- Evaluate the `HTTP_ACCEPT_LANGUAGE` HTTP header for preferred languages.
- Evaluate the language preference in the profile of a known user.

For more information about how Sitecore determines the context language, including an example pipeline processor that overrides that logic, see my blog post at `http://bit.ly/qOV7sg`. For an example of a web control that links to all languages that contain a version of an item, see my blog post at `http://bit.ly/omG5Fd`. For more information about web controls, see Chapter 3.

Falling Back to an Alternate Language

Sitecore does not automatically fall back to an alternate language if an item or a field value does not exist in the context language; Sitecore behaves as if the item has no field values or as if no value exists for the field. The LanguageFallback (`http://bit.ly/fCoZmi`) and LanguageFallbackItemProvider (`http://bit.ly/yT4xM3`) Sitecore Shared Source projects can help you implement fallback languages if you wish, but requirements can become complex quickly. For the benefit of your site visitors, I recommend against mixing content from multiple languages on a single page, or linking to content that does not exist in the context language. If you do link to content that does not exist in the context language, near the link, indicate the language that the user will see if they access that page, such as by showing that language name in parentheses after the link.

www.it-ebooks.info

MANAGING MEDIA

Use the Sitecore media library to manage binary media, including images, PDFs and other files. Media items are like any other items based on data templates. You can access the media library that contains media items by clicking the Media Library tab at the bottom of the Content Editor, by clicking the Media Library command on the Sitecore menu in the desktop, or by navigating to the `/sitecore/media library` item in the Content Editor. The Media Library application is the Content Editor application with the content tree rooted at the `/sitecore/media library` item, where Sitecore manages media. When you upload a file into the media library, Sitecore creates a media item with fields containing metadata about that file.

By default, Sitecore stores the binary components of media library items in a database using the field named Blob in the File section of the media item. Sitecore developers refer to such media items as database media. Alternatively, you can configure Sitecore to store the binary components of media items on the filesystem. Sitecore developers refer to such media items as file media. In the case of file media, Sitecore stores the binary component of the media item in the file specified in the File Path field in the File section of the media item.



Sitecore uses media items to represent both database media and file media. File media that CMS users manage and reference in content differ from files on the file system that developers manage and reference from code components.

In some cases, such as that of very large binary assets, Sitecore stores media items as file media automatically. Because Sitecore advises against the use of file media in favor of database media, this book does not describe file media.



When you upload a `.zip` file into the media library using the advanced user interface for uploading media, you can choose to extract the contents of that file to create media folders and media items based on its contents instead of creating a single media item representing the `.zip` file.

You do not have to use the Sitecore media library for every media resource in a Sitecore solution. The media library is for media managed by CMS users. If you do not require CMS services such as security, translation, versioning, locking, publishing, and workflow, and cannot benefit from additional features such as Sitecore media caching and dynamic image manipulation, you can store resources on the filesystem without using the media library, especially media used by developers rather than CMS users.



Changes to items in Sitecore databases and changes to the contents of files typically follow different deployment processes. Changes to items typically follow a publishing or workflow process; changes to files typically follow a release management process.

www.it-ebooks.info

By default, Sitecore does not maintain version information for media items and does not apply workflow to media items. To support versioning and workflow, set the `Media.UploadAsVersionableByDefault` setting in the Web.config file to `true`. This setting causes Sitecore to use data templates that support versioning for media items.



Under the default configuration, Sitecore can publish any media item at any time, including updates to those items.

Consider the impact of database media and caching on your storage requirements. Because of encoding, binary data can consume significantly more space in a relational database than on a filesystem. In addition to the number of media assets and their individual sizes, database storage requirements depend on whether you translate the binary component of media (different binary content and metadata for different languages), and for the Master database, how frequently you version media items. Finally, by default, each Sitecore instance caches media data to a local filesystem.

Sitecore Media Folders

Sitecore uses the `System/Media/Media folder` data template to structure the media library into a hierarchy. In the Sitecore content tree, everything is an item, and every item can have children. A folder is simply an item based on a data template that does not define any fields beyond those defined by the standard template. Standard values for the `System/Media/Media folder` data template specify a custom item editor (the Folder tab that appears when you select a media folder in the media library) and insert options that simplify creating nested media folders within a media folder, uploading new media items, and updating existing media items.



In general, the information architecture of the media library should follow the conventions used by the content information architecture. The names and locations of folders in the media library should match those of corresponding content items. To increase performance and usability, prevent media folders from containing hundreds of media items.

Media URLs

Sitecore constructs the URL for a media items much as it constructs the URL for a content item: based on the path to and name of the media item. Media URLs begin with the `~/media` prefix (as specified by the `Media.MediaLinkPrefix` setting in the Web.config file), continue with the path to the media item relative to the `/sitecore/media library` item, and end with the `.ashx` extension. For example, the default URL of the `/sitecore/media library/Images/MyImage` item would be `~/media/Images/MyImage`. The `~/media` prefix triggers the Sitecore media request handler, which processes query string parameters and media metadata, resizes images if required, and serves the HTTP response.

If you change the value of the `Media.MediaLinkPrefix` setting in the Web.config file, you should duplicate the `/configuration/sitecore/customHandlers/handler` element in that file

www.it-ebooks.info

with a value of `~/media/` for the `trigger` attribute below that existing `<handler>` element, and update the `trigger` attribute of the new `<handler>` element to the new value of the `Media.MediaLinkPrefix` setting. Also, add a `/configuration/sitecore/mediaLibrary/mediaPrefix/prefix` element to the Web.config file with a `value` attribute containing the value of the `Media.MediaLinkPrefix` setting.

The `.ashx` extension in the URLs of Sitecore media items causes IIS to service the HTTP request with ASP.NET instead of attempting to serve a file from disk. If your configuration of IIS uses ASP.NET for requests for media extensions, such as `.jpg` and `.pdf`, to replace the `.ashx` extension in media URLs with the original file extension of the media item, set the `Media.RequestExtension` setting in the Web.config file to an empty string. By default, Sitecore does not output a leading slash (`/`) as the first character in media URLs. Browsers interpret such URLs relative to the current page rather than relative to the document root of the IIS website. While IIS and hence Sitecore resolve these relative URLs correctly, they can result in multiple URLs for a single media item. More importantly, especially in deep information architectures and solutions with very long item names, such URLs can exceed IIS limitations for URL length.

ENSURING THAT MEDIA URLS BEGIN WITH A SLASH CHARACTER

Use the `Sitecore.StringUtil.EnsurePrefix()` static method to ensure that media URLs begin with a slash (`/`) rather than a tilde. For example, to access the URL of the media item referenced in the Image field named Image in the context item, use the following code:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
Sitecore.Data.Fields.ImageField image = item.Fields["Image"];

if (image != null && image.MediaItem != null)
{
 string url = Sitecore.StringUtil.EnsurePrefix('/',
  Sitecore.Resources.Media.MediaManager.GetMediaUrl(image.MediaItem));

 // use url
}
```

The MediaUrlTransformer (`http://bit.ly/sBb2m2`) Sitecore Shared Source project includes a `renderField` pipeline processor that adds the leading slash to images and links to media in field values that you render using that pipeline and constructs that invoke that pipeline.

The `Sitecore.Resources.Media.MediaUrlOptions` class exposes a number of properties that control how Sitecore generates a URL for a media item. Each such property corresponds to a query string parameter that can appear in media URLs. You can use the same query string parameter keys as attribute names for the `<sc:image>` XSL extension control that renders an `<img>` element, with equivalent results. Relevant properties of the `Sitecore.Resources.Media.MediaUrlOptions` class appear in Table 2-2.

www.it-ebooks.info

TABLE 2-2: Sitecore.Resources.Media.MediaUrlOptions Class Properties

PROPERTY	KEY	VALUE	DEFAULT	DESCRIPTION
`AllowStretch`	`as`	`0` or `1`	`0`	Allows an image to stretch beyond its original size
`BackgroundColor`	`bc`	Color name or hexadecimal code	`Black`	Background color to use when increasing an image size and `AllowStretch` is `false`
`Database`	`db`	Sitecore database name	Context database	Sitecore database from which to retrieve media item
`DisableBrowserCache`	`ts`	Timestamp	N/A	Unique timestamp in URL to prevent browser from using cached image
`DisableMediaCache`	`dmc`	`0` or `1`	`0`	Prevents the retrieval of cached data and the addition of an entry to the media cache for this request
`Height`	`h`	Positive integer	N/A	Height of the image in pixels
`Language`	`la`	Language name	Context language name	Language of the media item image
`MaxHeight`	`mx`	Positive integer	N/A	Maximum height of the image in pixels
`MaxWidth`	`mw`	Positive integer	N/A	Maximum width of the image in pixels
`Scale`	`sc`	Positive integer	N/A	Scaling factor for the image
`Thumbnail`	`thn`	`0` or `1`	`0`	Retrieves the default thumbnail of the image
`Version`	`vs`	Version number	Latest	Version number of the media item
`Width`	`w`	Positive integer	N/A	Width of the image in pixels

www.it-ebooks.info

Most of these properties apply only to images, but the `Language`, `Version`, `Database`, `DisableMediaCache`, and `DisableBrowserCache` properties apply to all types of media items.

Avoid constructing media URLs containing these query string parameters manually in favor of higher-level APIs. To retrieve the URL of the icon for an item, pass the `Sitecore.Data.Items.Item.Appearance.Icon` property to the `Sitecore.Resources.Images.GetThemedImageSource()` static method. To retrieve the URL of the thumbnail for an item, call the `Sitecore.Resources.Media.MediaManager.GetThumbnailUrl()` static method. You can use the `Sitecore.Resources.Media.MediaManager.GetMediaUrl()` static method with the `Sitecore.Resources.Media.MediaUrlOptions` class to construct a media URL programmatically. For more information about these APIs, see The Sitecore Content API Cookbook (`http://bit.ly/oC6wHw`).

Configuring the Media Library

A number of settings in the Web.config file affect the way Sitecore manages media. For more information about each setting, see the comments above that setting in the Web.config file.

Some of the settings available for media include:

- `Media.AutoSetAlt` — Automatically sets the Alt field (for alternate text) in image items to the name of the media item
- `Media.DisableFileMedia` — Enables or disables file media (as opposed to database media, which store the binary component of media items to the database)
- `Media.IncludeExtensionsInItemNames` — Controls whether Sitecore includes the original file name extension without the leading dot (`.`) character, such as `jpg`, in the names of media items (potentially useful for identifying file types in the content tree)
- `Media.MaxImageWidth` — Controls the maximum width for images in Rich Text Editor fields
- `Media.MaxSizeInDatabase` — Controls the maximum size of media allowed in the database
- `Media.MaxSizeInMemory` — Controls the maximum size of media to load into memory for transformation (resizing of an image, and so on)
- `Media.MediaLinkPrefix` — Controls the prefix used in URLs to identify media requests
- `Media.OpenContentEditorAfterUpload` — Controls whether the Content Editor opens the media item after a CMS user uploads a file so that the user can enter media metadata such as alternate (`alt`) text for images
- `Media.UseItemPaths` — Controls whether Sitecore uses item paths or IDs for media URLs



Do not set the `Media.UseItemPaths` *setting in the Web.config file to* `false`*. Doing so causes media URLs to contain the IDs of media items with curly braces (*`{}`*) and dashes (-) removed rather than media item paths and names.*

- `Media.WhitespaceReplacement` — Controls the character to use in item names to replace all whitespace in file names

www.it-ebooks.info

If processing of a media item exceeds either the `Media.MaxSizeInDatabase` or the `Media.MaxSizeInMemory` setting in the Web.config file, Sitecore may generate an error message (if you try to upload too large a media item), show a validation error (if you managed to upload a media item that is too large), or write an error message to the page trace, and may throw an apparently unrelated exception.

Additionally, the following settings in the Web.config file configure caching of media by clients as described in the comments above them in the Web.config file, as well as the resources in the following list (all of these URLs lead to pages under `http://msdn.microsoft.com`:

- `MediaResponse.Cacheability` — `http://bit.ly/nkapeM`
- `MediaResponse.CacheExtensions` — `http://bit.ly/o70LOe`
- `MediaResponse.MaxAge` — `http://bit.ly/qYhdJO`
- `MediaResponse.SlidingExpiration` — `http://bit.ly/qZb6NF`

Media Upload Watcher

The media upload watcher monitors the subdirectory specified by the `MediaFolder` setting in the Web.config file for certain types of changes. For each new subdirectory or file, the upload watcher creates a corresponding media folder or item in the media library.



If you use the media upload watcher to import large quantities of media, to help ensure success and to avoid overwhelming the system, import in small batches and monitor the Sitecore logs during the resultant processing. If you move or rename files and subdirectories under the subdirectory specified by the `MediaFolder` *setting, the media upload watcher creates new media folders and items but does not move or rename existing media items. If you delete files and subdirectories from the subdirectory specified by the* `MediaFolder` *setting, the media upload watcher does not delete the corresponding media items. If you move, rename, or delete items created from files and subdirectories written to the subdirectory specified by the* `MediaFolder` *setting, Sitecore does not move, rename, or delete the corresponding subdirectories and files from the subdirectory specified by the* `MediaFolder` *setting.*

To disable the media upload watcher, such as in the content delivery environments or if you do not use it or in the content management environment, comment out the `/configuration/system.webServer/modules/add` and `/configuration/system.web/httpModules/add` elements named `SitecoreUploadWatcher` in the Web.config file.



The media upload watcher depends on a running ASP.NET worker process. Before you add files or subdirectories to the subdirectory specified by the `MediaFolder` *setting in the Web.config file, use a browser to request an ASP.NET resource, such as the homepage of the site, to ensure that ASP.NET is active.*

www.it-ebooks.info

WebDAV

You can use WebDAV to enable CMS users to drag and drop files from their remote desktops into the Sitecore media library.



The File Drop Area field type uses the media library and WebDAV.

The Sitecore WebDAV implementation uses a third-party WebDAV engine. You do not need to enable any native WebDAV components in IIS.



The Sitecore WebDAV implementation is impressive, but supporting a wide range of WebDAV clients (browsers) can present challenges, especially considering the number of operating systems and browser versions combinations possible. Before overinvesting in WebDAV, ensure that a sufficient percentage of your CMS users access the solution using user agents for which you can support WebDAV, and that they intend to use the File Drop Area field type or other WebDAV features.

To disable WebDAV, change the extension of `/App_Config/Include/Sitecore.WebDAV.config` file, for example by adding `.example` (renaming it to `Sitecore.WebDAV.config.example`). Alternatively, set the `WebDAV.Enabled` setting to `false` in the `/App_Config/Include/Sitecore.WebDAV.config` file, or delete the file. You can also remove the `/configuration/system.webServer/handlers/add` and `/configuration/system.web/httpHandlers/add` elements that have `webdav` in their names or that otherwise indicate support for WebDAV from the `/web.config` file and any Web.config include files, such as anything that contains the string `sitecore_webDAV.ashx`.

The Sitecore WebDAV implementation generates its own log file to the same directory used for the Sitecore system log as described in Chapter 6. If you disable WebDAV, Sitecore continues to generate an empty WebDAV log file. To prevent Sitecore from generating the WebDAV log file, comment out the entire `/configuration/log4net/appender` element named `WebDAVLogFileAppender` in the `/web.config` file. If you comment this element out without disabling WebDAV, messages from the WebDAV subsystem appear in the standard Sitecore log file, possibly with less detail than they would provide in the WebDAV-specific logic, and possibly duplicated (each message from WebDAV may appear in the system log twice).

For more information about WebDAV, such as how to configure your server and clients to support it, see The Sitecore WebDAV Configuration Guide (`http://bit.ly/qeKdb4`).

Media Types

When you create a new media item, Sitecore uses configuration data for that type of media as defined by the `/configuration/sitecore/mediaLibrary/mediaTypes/mediaType` element in the Web.config file with a value for the `extensions` attribute that matches the original extension in the file name. If no such element exists, Sitecore uses media type configuration data from the

www.it-ebooks.info

/configuration/sitecore/mediaLibrary/mediaTypes/mediaType element in the Web.config file with a value for the extensions attribute that includes an asterisk (*).



Changing a data template specified in a /configuration/sitecore/media Library/mediaTypes/mediaType *element in the Web.config file does not update the data template associated with existing media items of that type.*

To configure how Sitecore manages a specific type of media, edit the existing /configuration/sitecore/mediaLibrary/mediaTypes/mediaType element in the Web.config file with a value for the extensions attribute that includes the file name extensions associated with that media type. If no such element exists, duplicate an existing /configuration/sitecore/mediaLibrary/mediaTypes/mediaType element, update the extensions attribute of that new <mediaType> element to specify the extension(s) of your media type, separated by comma (,) characters. Update the contents of the new <mediaType> element to meet your requirements.

You can configure Sitecore to allow versioning of all media items, and CMS users can designate whether Sitecore should version each media item they upload using the advanced media upload user interface. To configure the data template that Sitecore uses for items of a media type that support versioning, set the value of the <versionedTemplate> child element of the relevant /configuration/sitecore/mediaLibrary/mediaTypes/mediaType element in the Web.config file to the path to that data template relative to the /sitecore/templates item. To configure the data template that Sitecore will use for items of a media type that do not support versioning, set the value of the <sharedTemplate> child element of the relevant /configuration/sitecore/mediaLibrary/mediaTypes/mediaType element in the Web.config file to the path to that data template relative to the /sitecore/templates item.

To register a new media type, do the following:

1. Create a new data template for versioned items of the type.
2. Set the base templates for the new data template to include the System/Media/Versioned/File data template, System/Media/Versioned/Image data template, or the appropriate data template that supports versioning for the type of media that you register.
3. Create a new data template for unversioned media items of the type.
4. Set base templates for the new data template to include the System/Media/Unversioned/File data template, the System/Media/Unversioned/Image data template, or the appropriate data template that does not support versioning for the type of media that you register.
5. Optionally, create a class to manipulate media metadata when you create new media items of that type.
6. Optionally, create a class to manipulate media streams when you create new media items.
7. Disable any /configuration/mediaType elements in the /App_Config/MimeTypes.config file with a value for the extensions attribute that contains a file name extension used by the media type.

www.it-ebooks.info

8. Duplicate an existing `/configuration/sitecore/mediaLibrary/mediaTypes/mediaType` element beneath itself in the Web.config file.
9. Remove irrelevant elements from the new `/configuration/sitecore/mediaLibrary/mediaTypes/mediaType` element in the Web.config file, and update the contents of that `<mediaType>` element to suit your purposes for that media type.

If you set the value of the `<forceDownload>` element within a `/configuration/sitecore/mediaLibrary/mediaTypes/mediaType` element in the Web.config file to `true`, Sitecore adds a `Content-Disposition` HTTP header to the response that instructs browsers to download media items of that type instead of attempting to open them.

Configuring MIME Types

To set the Multipurpose Internet Mail Extensions (MIME) type for a media type, update the value of the `/configuration/mediaTypes` element in the `/App_Config/MimeTypes.config` file with a value for the `extensions` attribute that includes the file extension associated with that media type. If no such element exists, you can create it, or create a `<mimeType>` element beneath the `/configuration/sitecore/mediaLibrary/mediaTypes/mediaType` element in the Web.config file with a value in the `extensions` attribute that includes the original extension for the file type. Values in the `/App_Config/MimeTypes.config` file take precedence over values in `<mimeType>` elements in the Web.config file.



When you change a MIME type associated with a media type in the Web.config file, Sitecore does not update the MIME type associated with existing media items of that type. To do that, update the value of the MIME type field in the Information section of existing media items of that type.

If Sitecore cannot determine the MIME type for a media item, it uses the value of the `<mimeType>` element within the `/configuration/sitecore/mediaLibrary/mediaTypes/mediaType` element in the Web.config file with a value for the `extensions` attribute that includes an asterisk (`*`) character.

Media Path Provider

The media path provider implements APIs that translate file paths to media item paths and media item paths to file paths. To customize how Sitecore determines the name and path to use for a new media item when a CMS user uploads a file into the media library or with the media upload watcher, override the media path provider as follows:

1. Create a class that inherits from the type specified by the `type` attribute of the `/configuration/sitecore/mediaPath/providers/add` element named `default` in the Web.config file.
2. Override methods in the base class according to your requirements, typically by calling the corresponding methods in the base class and then altering their results.
3. Update the `type` attribute of the `/configuration/sitecore/mediaPath/providers/add` element named `default` in the Web.config file to the signature of your class.

www.it-ebooks.info

Caching Media

Sitecore manages a cache on the filesystem for some types of metadata and the binary components of media items automatically. For each managed site, Sitecore creates a subdirectory under the subdirectory specified by the `Media.CacheFolder` setting in the Web.config file, and stores media cached for that site in that subdirectory. Sitecore caches media items after it applies transformations, such as after it resizes images. To prevent Sitecore from caching media for a managed site, set the `cacheMedia` attribute of the `/configuration/sitecore/sites/site` element in the `Web.config` file that defines that site to `false`. To disable media caching for all sites, set the `Media.CachingEnabled` setting in the Web.config file to `false`.



You might see some performance improvement if you replicate the media cache between instances in load-balanced environments, but otherwise each instance will build its own media cache as needed.

SHARING CONTENT

You can easily share field values, items, branches of items, content in external systems, and other resources among presentation controls, pages, external systems, and other components. When rendering any item, almost every Sitecore solution includes data from other items in the repository. Sitecore provides the following facilities to allow you to share content as described elsewhere in this chapter or the remainder of this book:

- **Standard Values** — All items based on a data template automatically share any standard values defined for that data template.
- **Data Sources** — You can pass the same data source item to multiple presentation controls or to presentation controls used by multiple pages, devices, or other components to indicate from where those components should retrieve shared data.
- **Rendering Properties** — Layouts, sublayouts, and layout details can pass parameters including item paths and IDs to presentation controls, reusing those values among multiple pages.
- **Conditional Rendering** — With conditional rendering you can dynamically apply common data source items and rendering properties to presentation controls.
- **Selection Fields** — You can create fields in your data templates to allow users to specify items for any number of presentation components to reuse on any number of pages.
- **Devices** — Presentation components can generate markup around, and format individual field values to meet requirements for, specific devices and general categories of devices, such as web browsers and mobile devices.
- **Syndication** — RSS feeds typically reuse existing items that represent pages on a managed site.
- **Aliases** — Aliases reuse the same context item at different URLs.

www.it-ebooks.info

- **Publishing Targets** — You can publish any item to any number of publishing target databases to use that data in different environments.
- **Cascading Style Sheets (CSS)** — Presentation components can determine CSS to apply at runtime, reusing items, presentation components, or both with different styling.



I use the term presentation for markup and the term styling for CSS.

- **Clones** — Clones automatically share data with cloned items.
- **XML** — External systems can access XML representations or other serializations of items in Sitecore.
- **Web Services** — External systems can invoke web services to access Sitecore APIs.
- **Wildcards** — Wildcards conceptually support various content reuse scenarios that may or may not be realistic.
- **Snippets** — You can define reusable snippets that appear in the Rich Text Editor (RTE).



Sitecore expands snippets when you insert them into the RTE. If you subsequently change a snippet definition, Sitecore does not update the values of Rich Text fields in which you previously inserted that snippet.

- **Dictionary** — Any number of presentation components can use the Sitecore dictionary to translate shared character sequences.
- **Token Substitution** — You can use a number of approaches to substitute tokens in field values when CMS users save data, when Sitecore publishes that data, or when the layout engine renders that data.

For more information about the concepts described in this section, see The Sitecore Guide to Reusing and Sharing Data (`http://bit.ly/pjNlqG`).

IMPORTING CONTENT

Importing content from an existing web management solution into the Sitecore ASP.NET CMS is straightforward. Getting the data out of the source system might not be as easy.

Before you can import content, there are a few APIs to learn as described in The Sitecore Content API Cookbook (`http://bit.ly/oC6wHw`), and if you automate the creation of data templates, The Sitecore Data Definition API Cookbook (`http://bit.ly/ohVWlq`). Pick an approach to import data, such as cut and paste, a simple ASP.NET page to invoke Sitecore APIs, or a scheduled process to invoke those APIs.

www.it-ebooks.info

Most importantly, the data to import must be in a place from which you can import it. If the volume of that data is relatively small, and especially if the quality is poor (as in bad styling, bad image references, broken links, obsolescence, lacking SEO features, inconsistent voice, poor information architecture, and so forth), it may be more efficient to cut and paste than to automate. Either way, consider importing content in small batches, check the quality along the way, and try to export content from the system in advance of the import instead of building a solution dependent on that running system.

TIPS FOR IMPORTING CONTENT INTO SITECORE

Apply these suggestions for importing content into Sitecore:

- Remember to update references in imported field values, including internal and external references to content, media, and external resources such as CSS and JavaScript. Also consider how you will map legacy CSS designations to your new styles.
- Importing a large volume of content can exceed ASP.NET page generation time limits as well as browser request timeout limits. If you use an ASP.NET page to import content, use programming techniques to refresh the page periodically to indicate the current status of that process.
- Consider how you will map old content URLs to new URLs, how you will update references in your content to the new URLs, and how you will handle requests to those old URLs (for example, bookmarks and links from external sites including Internet search engines).
- Import media before importing content that references that media.

In some cases you may be able to use the XmlImporter (`http://bit.ly/zKFu2n`) Sitecore Shared Source project to import content. Kapow Software also provides a framework for importing content into Sitecore (see `http://bit.ly/zp7ap9`). For more information about importing content into the Sitecore ASP.NET CMS, see my blog post at `http://bit.ly/qnihna`.

TAKE HOME POINTS

This chapter provided a brief overview of fundamental Sitecore components relevant to information architecture. For additional suggestions for working with these constructs, see Chapter 11. *Items* represent entities as a collection of named field values defined by a data template. Sitecore arranges items in a hierarchy used to determine the default URL of each item. *Definition items* specify characteristics of system components, such as languages and devices. *Insert options* control what users can create beneath existing items.

Clones work as dynamic copies of existing items, in which you can override values from the cloned item. *Aliases* provide alternate URLs for content items. *Wildcards* map numerous HTTP requests to individual Sitecore items. You can use Sitecore to implement multilingual solutions, including

www.it-ebooks.info

different content for different regions and cultures. You can store binary media in a Sitecore database, and reference those media from your Sitecore content. You can use a variety of techniques to share content in a Sitecore database among various components and even external systems.

Data templates define the structures for a type of item. Data templates can define any number of visual sections, where each section can contain any number of fields. Each field defines a number of properties including the field's name, ID, data type, the versioning and translation features the field supports, and additional configuration for the field. Data templates can inherit sections, fields, and standard values from some number of base templates. Most data templates eventually inherit from the standard template that contains fields for common system properties such as creator and workflow state. You can use validation to enforce data entry requirements. *Standard values* provide field values that apply to all items based on a data template that do not override those values.

You can reverse-engineer page mockups to determine the requirements for your data templates. Once your data templates exist you can write a process to import existing content. In your process, create items and set field values based on existing data.

www.it-ebooks.info

3

The Layout Engine and Data Access APIs

WHAT'S IN THIS CHAPTER?

- Enhancing the ASP.NET page lifecycle
- Defining the Sitecore context
- Developing presentation components
- Accessing fundamental APIs and data access techniques
- Implementing conditional rendering and personalization
- Understanding the renderField pipeline

This chapter introduces fundamental concepts of the layout engine responsible for servicing HTTP requests with the Sitecore ASP.NET web content management system (CMS). After you read this chapter, you will have a greater understanding of how to use these constructs to build web solutions with Sitecore.

Among other responsibilities, the layout engine applies logic and merges predefined markup and logic in the presentation components that you implement with content from the CMS. Each presentation component can access the entire Sitecore information architecture including all field values to which the context user has read access. In addition to Application Programming Interfaces (APIs) and eXtensible Stylesheet Language (XSL) extensions provided by Sitecore, your presentation components can access any APIs and XSL constructs available on the ASP.NET platform, including external systems, web services, RSS feeds, and other resources. As with any website, the markup that you generate can reference styling, JavaScript, media, and other resources managed in Sitecore, the document root of the website, or at external locations.

www.it-ebooks.info

ASP.NET services HTTP requests by building a control tree, where controls can generate markup by implementing methods to handle events in the page lifecycle. Significant difference between a standalone ASP.NET solution and a Sitecore solution involves enhancements to the process for building and manipulating the control tree. The first involves declarations in items stored in the database such as to add controls to placeholders in existing components, while the second involves global and localized logic such as to change the data sources of those components, instead of relying primarily on markup and code stored in individual files. For each HTTP request that Sitecore handles (except when using specific ASP.NET handlers, such as the handler for media requests), the layout engine populates the Sitecore context, assembles the ASP.NET control tree based on not just markup and code in files, but also layout details for the context device in the context item. It then uses the rules engine to apply any conditional rendering required for those controls, such as to set properties of the controls. ASP.NET then renders the control tree as a page. Logic around individual controls and especially the `renderField` pipeline extends field values in the CMS, presentation controls, and other components of the page with features such as inline editing in the Page Editor and rendering information in the Sitecore's browser-based page debugger.



Sitecore continues to provide the Developer Center application for working with presentation components in the browser. Considering the productivity it allows for developers, as well as its intuitive integration with the IDE (Integrated Development Environment), its extensibility, its vast capabilities, and Sitecore's strong commitment to its further development, I recommend that you use the Sitecore Rocks extension for Microsoft Visual Studio rather than the Developer Center.

The layout engine does not include ASP.NET MVC (Model-View-Controller) functionality, which this book does not describe. For information about using ASP.NET MVC with Sitecore, see Integrating an ASP.NET MVC Web Application in Sitecore CMS (`http://bit.ly/qP3ISo`).

This chapter is intentionally somewhat terse to avoid duplicating Sitecore's own extensive documentation. I strongly recommend that you read that documentation before or after you read this chapter. For more information about the layout engine beyond that provided in this chapter, see The Sitecore Presentation Component Reference (`http://bit.ly/o2TdvV`), The Sitecore Presentation Component Cookbook (`http://bit.ly/nDo0Ek`), The Sitecore Presentation Component API Cookbook (`http://bit.ly/r76c6K`), and The Sitecore Guide to Troubleshooting Presentation Components (`http://bit.ly/ruhzKb`).

THE ASP.NET PAGE LIFECYCLE

Sitecore uses a number of techniques to enhance the standard ASP.NET page lifecycle, such as invoking pipelines (described in Chapter 7) at various points during request processing, as well as providing application-level exception management. If you are new to ASP.NET development, this section provides background information about the page lifecycle and describes how Sitecore implements these features. This is not a complete description of how IIS (Internet Information Services) uses ASP.NET to process HTTP requests, or of how Sitecore enhances the ASP.NET page lifecycle.

www.it-ebooks.info

Conceptually, IIS receives a request from the Internet using one of various Internet protocols, such as HTTP. After routing the request to the appropriate website typically based on host header bindings, IIS determines whether to use ASP.NET to process the request and, if so, which application within the website should service the request, which triggers the ASP.NET page lifecycle.



Where relevant, I use the term "website" to indicate an IIS site, and the term "managed site" to indicate an individual logical site managed by Sitecore within that website, such as the default published site named website or the site named shell that provides access to the Sitecore user interfaces.

The `Sitecore.Web.Application` class extends the default `System.Web.Application` class that is responsible for the ASP.NET application at the root of the IIS website hosting the Sitecore instance. In some cases multiple instances of the application exist for a single IIS website.



Because Sitecore URLs correspond to items in a database, not files and subdirectories on a filesystem, you cannot configure a subsection of Sitecore to use a separate application. For additional information about this issue, see "Creating a project in a virtual directory under Sitecore root" at `http://bit.ly/odz5dO`.

Some configurations of IIS do not use ASP.NET to service all requests, meaning that Sitecore effectively never has a chance to process those requests. Additionally, features such as the `preprocessRequest` pipeline and the `IgnoreUrlPrefixes` setting in the Web.config file can prevent Sitecore from establishing a complete Sitecore context for the request or even servicing the request at all. The easiest way to configure IIS to use ASP.NET to process all requests is to use IIS7 with an application pool using the Integrated managed pipeline mode. For instructions to configure various versions of IIS to use ASP.NET to process all HTTP requests, see The Sitecore Guide to Dynamic Links (`http://bit.ly/ri6Oww`).

ASP.NET uses a number of handler classes to service various types of requests, where the handler is responsible for constructing an HTTP response. Each handler implements the `System.Web.IHttpHandler` or `System.Web.IHttpAsyncHandler` interface. When servicing an HTTP request, ASP.NET parses the URL to determine the handler to trigger. Components of the URL, including the extension in what appears as the filename such as `.aspx` (web form), `.asmx` (web service), and `.ashx` (web handler), as well as prefixes such as `~/media`, trigger specific handlers. Not all URLs serviced by ASP.NET correspond to files under the document root subdirectory, but can instead trigger logic to generate an HTTP response. ASP.NET or the handler creates objects as required for the handler, and pushes those objects through the page lifecycle.

The `/global.asax` file at the application root specifies the class used for the ASP.NET application hosted at that location, and can define methods to add to that class. The default global application is `System.Web.HttpApplication`, which exposes methods that correspond to stages in the page lifecycle. The `/global.asax` file provided with Sitecore configures the `Sitecore.Web.Application`

www.it-ebooks.info

class as the global application, which inherits from the `System.Web.HttpApplication` class. The global application can be appropriate for logic that applies to an entire application; HTTP modules provide similar functionality in a more modular fashion. For more information about HTTP modules, see "HttpModules" at `http://bit.ly/AdbTG2`.

In a .NET presentation control, the static `Current` property of the `System.Web.HttpContext` class exposes an instance of that class that represents the current HTTP context, which exposes `Request` and `Response` properties that represent the HTTP request and the response under construction to service that request, respectively. You can read from the `Request` and write to the `Response`, such as to set the HTTP status code, cookies, and HTTP headers.

The Sitecore layout engine uses ASP.NET Web Forms (`.aspx` files) to service requests. An `.aspx` file contains markup that works as a prototype for the response, but can also reference a code-behind file that represents a class responsible for populating that prototype with information. That class inherits from the `System.Web.UI.Page` class. The handler for Web Forms creates an instance of that class, which then follows the page lifecycle.



The `System.Web.UI.Page` *abstraction is relatively expensive; HTTP handlers can provide solutions that are more efficient. The Sitecore media library uses a web handler (*`.ashx`*) to service requests. For more information about HTTP handlers and HTTP modules, see "HTTP Handlers and HTTP Modules Overview" at* `http://bit.ly/wUshMp`.

The `System.Web.UI.Page` class implements the `System.Web.IHttpHandler` interface and uses the `Controls` property to represent individual literal and dynamic components responsible for the presentation of elements within the page, each of which can generate markup and contains its own `Controls` property allowing nesting to any level. When the page lifecycle invokes the `Render()` method of the `System.Web.UI.Page` class, it passes a `System.Web.UI.HtmlTextWriter` object as an argument to the `Render()` method. That argument resolves to the `System.Web.HttpContext.Current.Response.OutputStream` property that represents the current HTTP response back to the requesting browser or other web client. Each control can write to the output stream, but also calls the `Render()` method of each of its child controls, recursively. Each control experiences the ASP.NET page lifecycle, including the ability to respond to page events such as user clicks, not just the `Render()` method.

Sitecore adds lightweight extensions to the ASP.NET page lifecycle. Most such functionality exists at the application level. Specifically, various events in the ASP.NET page lifecycle trigger Sitecore to invoke pipelines, including the `preprocessRequest`, `httpRequestBegin`, `renderLayout`, and `httpRequestEnd` pipelines, as well as Sitecore error handling.



The ASP.NET application lifecycle triggers additional pipelines, such as the `initialize` *and* `shutdown` *pipelines.*

For more information about the ASP.NET page lifecycle, see "ASP.NET Page Life Cycle Overview" at `http://bit.ly/oG9tut`.

www.it-ebooks.info

THE SITECORE CONTEXT

Sitecore defines a processing context for each HTTP request. The Sitecore context contains information about the user, the requested item, the requesting device, the managed site, and other information associated with the current HTTP request. For each HTTP request, Sitecore invokes the `httpRequestBegin` pipeline, which is largely responsible for populating the Sitecore context.

Properties of the Sitecore context indicate a variety of information, including the current managed site and database accessed, the security context, and the content item requested, including its language and version. Important static properties of the Sitecore context (the `Sitecore.Context` class in the `Sitecore.Kernel.dll` assembly) include the following:

- `Database` — Sitecore database associated with the current request, or default database for the context site
- `Device` — Sitecore device associated with the current request, or default device for the context site
- `Domain` — Sitecore security domain associated with the context site
- `Item` — Item in the context database requested by the web client (based on the path in the requested URL)
- `Items` — Keyed collection of values available throughout the HTTP request lifecycle



If you add items to this collection, be sure to use keys that Sitecore would never use.

- `RawURL` — Original URL requested by the web client before any modifications by Sitecore
- `Site` — Managed site associated with the current request
- `User` — Authenticated user, or anonymous user in the context security domain



Some properties of the Sitecore context, such as the context language, database, and security domain, can depend on properties of the context site. Both the Preview and Page Editor interfaces set the context site as they would for an HTTP request for a published site rather than the shell site used to implement the CMS. For these CMS user interfaces, Sitecore sets the context database to the Master database rather than the publishing target database those published sites would otherwise use by default.

For more information about the Sitecore context, see The Sitecore Presentation Component API Cookbook (`http://bit.ly/uauNa7`) and my blog post at `http://bit.ly/rdpqLA`.

www.it-ebooks.info

LAYOUT DETAILS AND LAYOUT DELTAS

Layout details define the presentation components that Sitecore should apply dynamically to render an item for a specific type of web client, such as a browser or mobile device. *Layout deltas* allow items to inherit layout details from other items (including standard values), storing only the differences between those inherited layout details and the layout details you define in the item, so that you can override individual presentation components as needed.



Define layout details in the standard values for each data template, and avoid defining layout details in individual items. If you need to override the layout details defined in a data template, create a new data template that inherits from the existing base template and apply new standard values for that data template.

For instructions to update layout details for an item programmatically, see my blog post at `http://bit.ly/ojwdXV` and note the comments about storing layout details in the item or storing layout deltas instead.

PRESENTATION COMPONENTS

This section describes the presentation components used by the layout engine, which include devices, layouts, sublayouts, web controls, XSL renderings, method renderings, URL renderings, placeholders, and data sources for presentation components. For more information about the topics described in this chapter, see The Sitecore Presentation Component Reference (`http://bit.ly/uauNa7`).



Layouts, sublayouts, XSL renderings, and web controls include both Sitecore items and filesystem components. You should check these files in with a source code management system. You typically deploy such files from a development environment through test environments to a production content management environment and any number of content delivery environments. The need to deploy changes to these files can affect whom you allow to edit these presentation components and the process for their deployment. Consider how you will migrate changes made by developers and any users that you allow to update presentation components. For more information on this topic, see Chapter 9.

Devices

Sitecore *devices* are logical constructs that allow Sitecore to apply different presentation components to a content item depending on aspects of the HTTP request. Devices can represent categories of appliances, such as personal computers, mobile phones, and tablets. Sitecore devices can, but do not necessarily, correspond directly to physical products or versions of products such as the Apple iPad 2. Devices can serve a wide range of purposes: for example, Sitecore uses layout details for the default device named Feed to format content for RSS feeds. You can even use devices to identify different browsers or versions of browsers.

www.it-ebooks.info

For each HTTP request, the `DeviceResolver` processor in the `httpRequestBegin` pipeline determines the context device from the `sc_device` query string parameter (which can specify the name of a device), by matching query string parameters against values in device definition items, by matching the user agent transmitted by the browser against values in device definition items, or from attributes of the context site.



The default `DeviceResolver` *processor in the* `httpRequestBegin` *pipeline does not support regular expressions. For an example of a similar processor that supports regular expressions, see* `http://bit.ly/qjooxr`.

The default devices are as follows:

- **Default** — Typically represents web browsers; applies to HTTP requests that do not trigger a specific device
- **Print** — Triggered by the query string parameter `p=1`; represents a printer
- **Feed** — Represents RSS readers



Technically, Sitecore uses layout details for the default device to render RSS feed definition items. Sitecore uses the first rendering added to any placeholder for the Feed device to specify the presentation control to render items when they appear in RSS feeds.

You can register any number of additional devices, such as one for mobile phones and another for tablets. You can register devices for specific hardware or even versions of hardware. You must define criteria to activate each device. You can associate devices with individual managed sites, such as a mobile site separate from the default site using the same content but with a different device. You can use the `sc_device` query string parameter to specify the name of a device, or you can define query string parameters or specify browser client user agent strings to match in your device definition items. If none of these techniques meets your requirements, you can implement an `httpRequestBegin` pipeline processor to resolve the context device according to your specific requirements. For an example that uses the Sitecore rules engine and the Wireless Universal Resource File (WURFL, pronounced wer-ful) database to determine the context device, see the MobileDeviceDetector (`http://bit.ly/gwryUW`) Sitecore Shared Source project. For more information about WURFL, see `http://wurfl.sourceforge.net`.



In some cases, you may wish to store the device in a cookie, to avoid the need to re-recognize the device on each request.

Layouts

Sitecore layouts are ASP.NET Web Forms (`.aspx` files) that define a markup superstructure typically shared to a number of pages. You can use anything in a layout that you can use in a standard ASP

www.it-ebooks.info

.NET web form, including codebehind. You might have layouts that you use for very specific purposes, but layouts are generally the most reusable presentation component. You typically have at least one layout per supported device, and if needed, another layout for the home page and any other pages that have unique designs or other requirements.

You can bind presentation controls to a layout (or a sublayout as described in a subsequent section of this chapter) statically, at design time, by adding the control to the markup in the `.aspx` file. Alternatively, you can bind presentation controls to placeholders in a layout (or a sublayout) dynamically, at runtime.



For greater reuse of content and code, use Sitecore layouts and layout details rather than ASP.NET master pages.

Placeholders and Placeholder Settings

Sitecore placeholders are ASP.NET controls to which you can dynamically bind presentation controls using layout details. You must define a key for each placeholder. When you bind a presentation control to a placeholder, reference that placeholder by its key. In the case of nested placeholders, such as when you bind a sublayout that contains a placeholder to a placeholder in a layout, you can also reference placeholders by full path (`/KeyOfPlaceholderInLayout/KeyOfPlaceholderInSublayout`).

Use placeholder settings to specify whether users can affect the controls bound to each placeholder in Page Editor, and, if so, what controls they can bind to that placeholder. For each placeholder key that you implement, create a placeholder settings definition item under the `/sitecore/layout/Placeholder Settings` item using the `System/Layout/Placeholder` data template. To control who can modify the presentation controls bound to a placeholder, restrict write access to the corresponding placeholder settings definition item.

The Data section of placeholder settings definition items contains the following fields:

- **Placeholder Key** — Key of the placeholder for which these settings apply
- **Allowed Controls** — Presentation controls the user is allowed to bind to the placeholder
- **Description** — Information relevant to users interested in binding controls to the placeholder, which can include a graphic that indicates the location of the placeholder on the page



Bind presentation components statically whenever possible (when every occurrence of the layout or sublayout uses those component(s) in the same place). Do not create excess placeholders; use them only for regions of the page that present different components under different conditions. Excessive use of placeholders has minimal impact on performance, but can increase unnecessary administration, as you may need to define repeatedly what binds to each placeholder, when static binding allows you to define that control placement for all uses of the layout or sublayout in a single location.

www.it-ebooks.info

Presentation Controls (Renderings)



Because they are largely interchangeable, I generally use the terms renderings *and* presentation controls *to include all sublayouts, XSL renderings, and web controls. The term* renderings *does not include placeholders, which are technically controls.*

In ASP.NET, controls are classes that can write to the HTTP response output stream to render part of a response to an HTTP request. Using a browser-based user interface to edit layout details, you can declare which Sitecore presentation controls to bind to each placeholder for each device, and you can define caching criteria for each presentation control. You can use the rules engine to pass parameters to presentation controls and to add and remove presentation controls at runtime.



If you use ASP.NET controls from a third party, test each to ensure that it functions correctly with the Sitecore layout engine. You may need to add some controls to the `/configuration/rendering/typesThatShouldNotBeExpanded` *element in the Web.config file. In some cases you may need to change the* `LayoutPageEvent` *setting in the Web.config file.*

Sublayouts

Sitecore sublayouts provide extensions around ASP.NET web user controls (`.ascx` files). Just like layouts, sublayouts support all the features provided by ASP.NET, including codebehind. Sublayouts suit a number of purposes:

- To render output like any other rendering or control
- To implement reusable page subdivisions, such as two-column and three-column sublayouts within a shared superstructure defined by a layout
- To implement web applications, such as HTML forms
- To create collections of controls that you can use in multiple contexts
- To dynamically generate any type of output required to service an HTTP request



Sublayouts are the most versatile type of presentation component. Only sublayouts support nested placeholders. Unlike XSL renderings, sublayouts support the Visual Studio debugger.

To enforce separation of concerns and maximize usability, I suggest that you minimize the number of layouts in favor of sublayouts. A layout can contain the minimal markup shared to all pages (including the `<html>`, `<head>`, `<body>`, and `<form>` elements), with embedded renderings or

www.it-ebooks.info

placeholders for everything else. Sublayouts typically contain features specific to individual pages or shared by a subset of the pages that use the same layout.

XSL Renderings

Sitecore makes it easy for you to present content using XSL renderings. You can use the `Sitecore.Web.UI.WebControls.XslFile` (`<sc:XslFile>`) web control class provided by Sitecore to invoke `.xslt` transformation files that can contain a variety of standard XSL elements and Sitecore XSL extension controls, and can invoke numerous Sitecore XSL extension methods. Sitecore also makes it easy for you to implement your own XSL extension methods, though extension controls are more difficult to implement than extension methods.

USING XSL RENDERINGS EFFECTIVELY

Considering the following points will help you make better use of XSL renderings:

- XSL renderings can be appropriate where nontechnical users must manage markup outside of content areas managed in the CMS or for developers with extensive knowledge of XSL.
- XSL extension controls are typically just syntactic sugar for invoking XSL extension methods.
- You can implement an entire Sitecore solution without using XSL.

For more information about XSL renderings, see The Sitecore Presentation Component XSL Reference (`http://bit.ly/qsVzvR`) as well as my blog posts at `http://bit.ly/pLZFud`, `http://bit.ly/rhwrTb`, `http://bit.ly/oUHHvh`, and `http://bit.ly/mXAhXl`. You might also find some value in the XslHelper (`http://bit.ly/rNISmG`) Sitecore Shared Source XSL extension library, even if just to further your understanding of XSL extension development.

Method Renderings

Method renderings invoke the method specified to the method rendering control. The method accepts no arguments, and the method rendering control writes the string returned from the method to the output stream.



Method renderings are uncommon and useful only when you need to invoke an existing method rather than creating a web control.

URL Renderings

URL renderings invoke the URL specified to the URL rendering control, and write the contents of the `<body>` element returned to the output stream. Unlike URLs specified to the `<iframe>` HTML element, which the web client runs, the server invokes URL renderings and transmits the response to the client as a component of the content of the page.

www.it-ebooks.info



URL renderings are uncommon and useful only when you need to embed the contents from one URL in the body of a page generated by Sitecore.

Web Controls

Sitecore web controls are classes that write to the HTTP response output stream. Classes that implement web controls derive from and extend the `Sitecore.Web.UI.WebControl` class, which derives from the `System.Web.UI.WebControls.WebControl` class. Web controls are more lightweight than sublayouts, with only one source code file to manage, and no need to deploy any source such as `.ascx` or `.cs`, but rather only the compiled assembly (`.dll`). Sitecore web controls implement the `DoRender()` method as required by the programming contract for the `Sitecore.Web.UI.WebControl` class, where web controls that derive directly from `System.Web.UI.WebControls.WebControl` implement the `Render()` method. The reason for this is that the `Sitecore.Web.UI.WebControl` class implements caching logic in the `Render()` method, which ASP.NET continues to call, and only calls the `DoRender()` method in your web controls if the layout engine cannot retrieve cached output generated previously by the control under equivalent processing conditions as described in Chapter 6.



Technically, all presentation controls use web controls. Sitecore uses the `Sitecore.Web.UI.WebControls.WebPage` *web control to implement URL renderings, the* `Sitecore.Web.UI.WebControls.Method` *web control to implement method renderings, the* `Sitecore.Web.UI.WebControls.Sublayout` *web control to invoke sublayouts, and the* `Sitecore.Web.UI.WebControls.XslFile` *web control to invoke XSL renderings.*

You can benefit from creating an abstract base class that inherits from the default `Sitecore.Web.UI.WebControl` abstract base class for web controls, and using your class as the base class for all your web controls rather than inheriting directly from the `Sitecore.Web.UI.WebControl`. Your abstract base class for web controls can do the following:

- Implement the `GetCachingID()` method to support output caching instead of implementing this in each web control. For example:

```
protected override string GetCachingID()
{
  string id = base.GetCachingID();

  if (String.IsNullOrEmpty(id))
  {
    id = this.GetType().ToString();
  }

  return id;
}
```

www.it-ebooks.info

- Implement the `Render()` method, for example to manage exceptions from the `DoRender()` methods of all web controls as described in Chapter 5
- Implement the `GetCachingKey()` method, for example to add custom caching criteria to cache keys



If you do not implement the `GetCachingID()` *method, Sitecore cannot cache the output of your web control.*

Rendering Parameters and Rendering Parameters Templates

You can pass parameters to presentation controls (renderings) to control their features, such as to set a style or indicate from where the rendering should retrieve some type of data. You can use `<xsl:param>` elements immediately after the root `<xsl:stylesheet>` element of an XSL rendering to define and accept parameters. For a web control to define and accept a parameter, add a property to the class that defines the web control. For information about accessing, defining, and accepting parameters in a sublayout, see the SublayoutParameterHelper (`http://bit.ly/rx1w8I`) Sitecore Shared Source project.



The names of rendering parameters cannot contain spaces.

Rendering parameters templates are simply data templates that you use to define rendering parameters. Sitecore uses rendering parameters templates to provide greater usability instead of requiring the user to enter key/value pairs.

To implement a custom rendering parameters template, follow these steps:

1. Add parameter definitions to the presentation control code. Specifically, for XSL renderings, add an `<xsl:param>` element to define the parameter name. For web controls, add a public property to the class that defines the web control. For more information about defining parameters for presentation controls, see the Sitecore Presentation Component Reference (`http://bit.ly/qdtczd`) and the Sitecore Presentation Component XSL Reference (`http://bit.ly/nEWh5x`).
2. Create a data template that inherits from the default rendering parameters template (`System/Layout/Rendering Parameters/Standard Rendering Parameters`) with fields named after the parameters defined in the presentation control. For instructions to create a data template, see the Sitecore Data Definition Cookbook (`http://bit.ly/qXSuJD`). Figure 3-1

www.it-ebooks.info

shows an example of a rendering parameters template named Sample Rendering Parameters that defines a single parameter named MyItem.

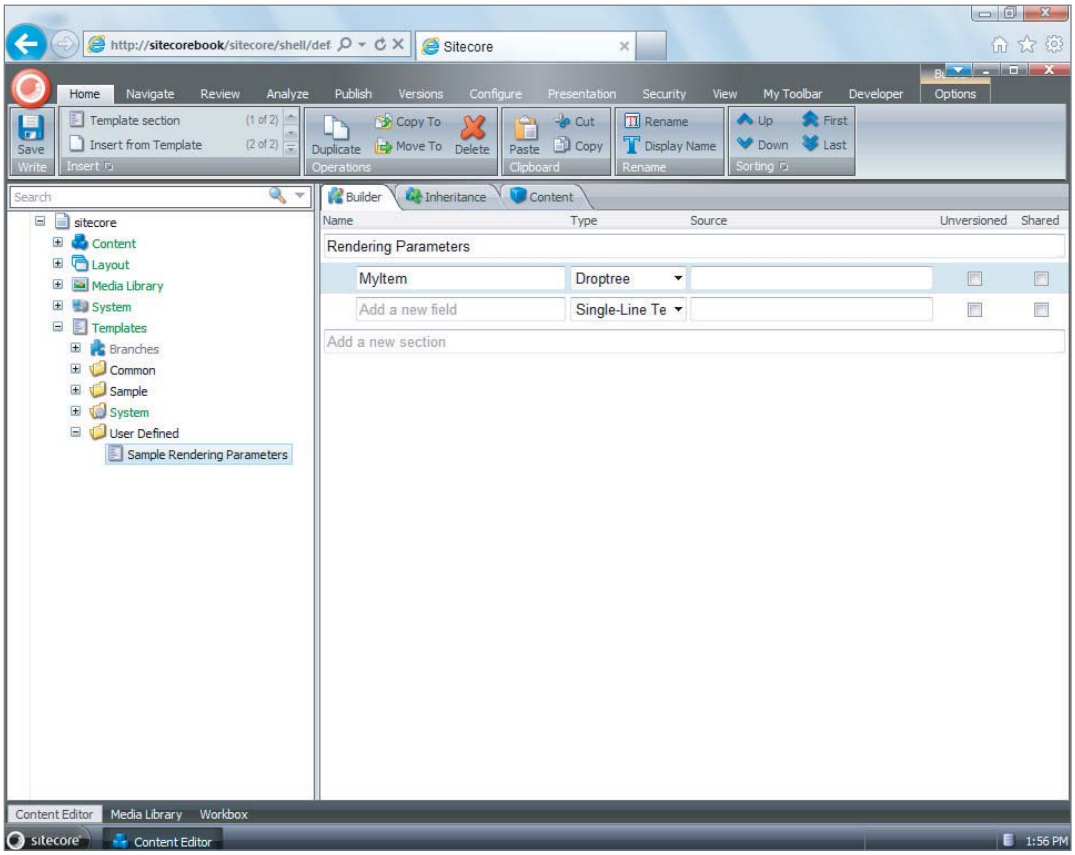


FIGURE 3-1

3. In the Content Editor, which you can access as described in Chapter 1, select the XSL rendering, web control, and/or sublayout definition item(s) that define the parameters specified in the new rendering parameters data template. In the Editor Options section, in the Parameters Template field, select the rendering parameters data template that you created previously. Figure 3-2 shows the `Sample/Sample Rendering` definition item configured to use the Sample Rendering Parameters template.

www.it-ebooks.info

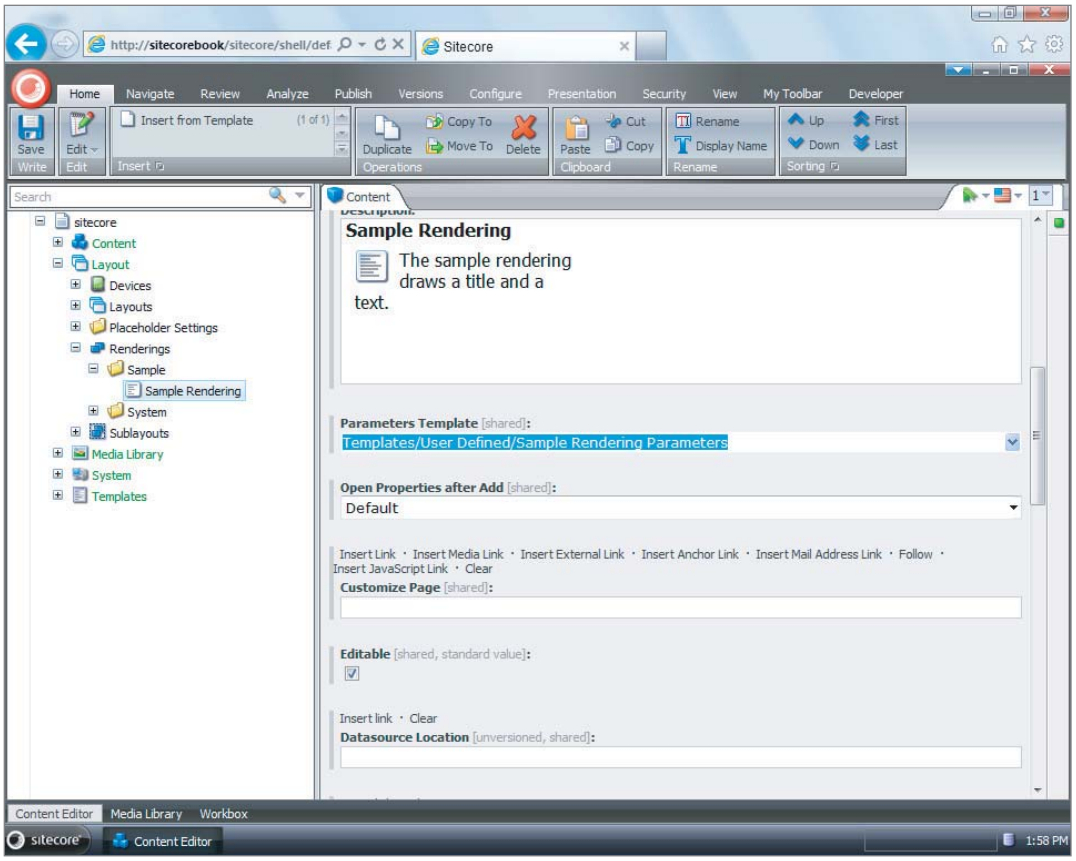


FIGURE 3-2

To specify a value for the parameter, update layout details for items that use the presentation control. In the Content Editor:

1. Select the item.
2. Select the Presentation tab.
3. Click Details in the Layout group. The Layout Details dialog appears as shown in Figure 3-3.
4. Click the rendering for which you want to define the parameter (for example, the rendering named Sample Rendering previously shown in Figure 3-3). The Control Properties dialog appears as shown in Figure 3-4.
5. Enter a value in the field that you defined in the rendering parameters template (My Item in Figure 3-4), and then click OK. The Control Properties dialog disappears and you return to the Layout Details dialog as previously shown in Figure 3-3.
6. Click OK. The Layout Details dialog closes and you return to the Content Editor.

www.it-ebooks.info

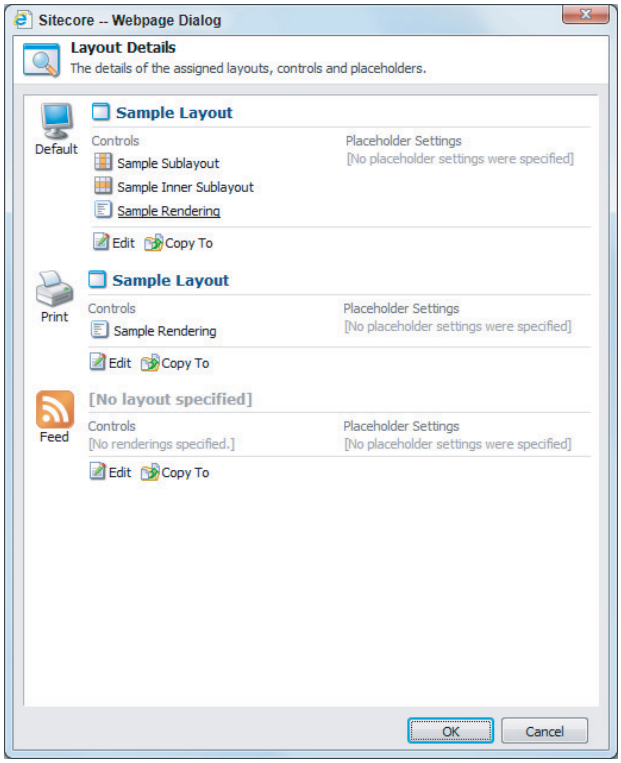


FIGURE 3-3

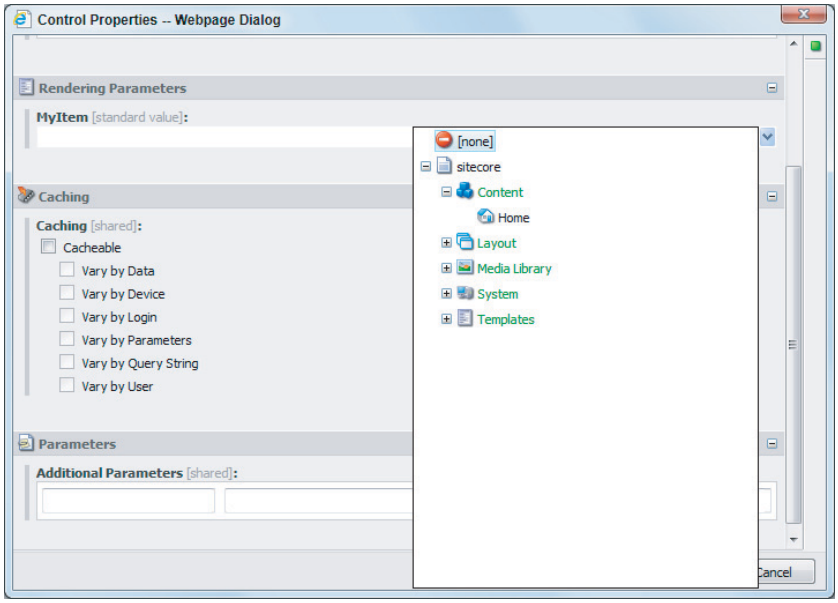


FIGURE 3-4

www.it-ebooks.info

Presentation Control Definition Item Properties

The data templates for presentation controls (sublayouts, XSL renderings, and web controls) provide a number of fields that you can use to manipulate how controls operate. You can edit these properties after selecting the definition item for a presentation control in the Content Editor. You can see some of these properties in Figure 3-2 shown previously.

Fields in the Editor Options section of all definition items for presentation controls serve the following purposes:

- **Description** — Textual explanation or even a graphic representation of the output of the control so users can use visual representations to select controls rather than item names.
- **Parameters Template** — Rendering parameters template for the presentation control, as described in the previous section.
- **Open Properties After Add** — If this checkbox is selected, Sitecore automatically opens the Control Properties dialog to allow the user to set additional properties after they add the control to layout details for an item.
- **Customize Page** — Legacy feature superseded by rendering parameters templates.
- **Editable** — Controls whether users of the Page Editor can update the rendering.
- **Datasource Location** — Location from which the user can select a data source for this presentation control, or under which they can create such an item.
- **Datasource Template** — Data template required for or to use when creating a data source item for this presentation control.
- **Compatible Rendering** — Limits the user's choice of presentation controls to replace this one.
- **Page Editor Buttons** — Commands to expose around the presentation control in the Page Editor.

Fields in the Caching section of definition items for presentation controls specify default criteria for the control. For more information about component caching criteria, see Chapter 6, The Sitecore Presentation Component Reference (`http://bit.ly/qdtczd`), and my blog post at `http://bit.ly/ndR0Bz`.

The fields that appear in the Data section of definition items for presentation controls depend on the type of control. Figure 3-5 shows some of the fields available for a web control.

The fields that can appear in definition items for the various types of presentation controls include the following:

- **Namespace** — .NET namespace containing the class that implements a web control
- **Tag** — Name of the .NET class within Namespace that implements a web control
- **TagPrefix** — ASP.NET prefix associated with the namespace containing the class identified by Namespace and Tag (matching the value of the `tagPrefix` attribute of one of the `/configuration/system.web/pages/controls/add` elements in the Web.config file)
- **Assembly** — .NET assembly (`.dll` file) containing the class that implements a web control

www.it-ebooks.info

- **Path** — Path to the code file for a sublayout or XSL rendering relative to the IIS document root
- **Placeholder** — Default placeholder for the control
- **Login control** — Alternate control to render if the user has not authenticated (typically a login control)
- **Data Source** — Default data source item for the presentation control

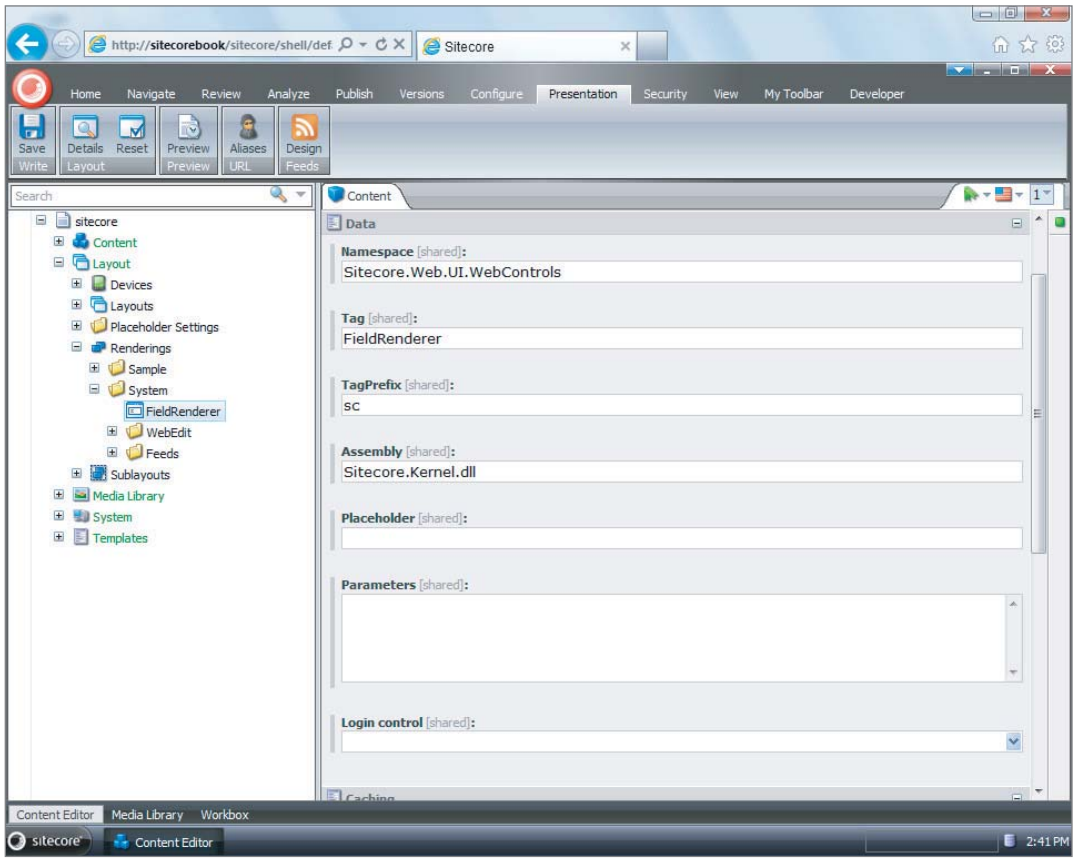


FIGURE 3-5

The Data Source (or Source) property of presentation components is entirely unrelated to the Source property in definition items for data template fields as described in Chapter 2. The data source of presentation components indicates from where they should retrieve data while rendering the site; the Source property of a data template field controls aspects of that field in data templates visible to CMS users.

www.it-ebooks.info

Choosing Presentation Technology

For each presentation control on a page, such as navigational elements or the footer, you can choose one of the presentation technologies listed in the previous five sections: sublayout, XSL rendering, method rendering, URL rendering, or web control. Alternatively, you can standardize on a single presentation technology for all site components, which has advantages for maintainability.

Most importantly, XSL renderings can have advantages, but they can also have significant limitations. While XSL is great for formatting markup, it is not very good for logic, even simple string parsing. You don't compile XSL renderings, so you don't get compile-time error reports, only runtime, and you can't step through XSL renderings using the Visual Studio debugger. Use XSL only if you already know its syntax, or if you need to let nontechnical CMS users update markup not stored in items.

Presentation Component Data Sources

The data source of each presentation component identifies an item from which that component should begin processing. Presentation components can retrieve field values from the data source item or from items related to it, or use the data source item for other purposes. Data sources aid in reusing content in multiple contexts, such as on multiple pages or for multiple devices, and in using a single rendering to process different items depending on the context. The default data source for each presentation component is the context item. You can specify the data source for each presentation component at design time, or you can use conditional rendering to apply a data source at runtime.



Technically, there is no data source for a layout; by convention, the Sitecore context item is the data source for a layout.

You can use the `$sc_item` parameter to access the data source of an XSL rendering. You can use the `GetItem()` method to access the data source of a web control. For information about accessing the data source of a sublayout, see the SublayoutParameterHelper (`http://bit.ly/rx1w8I`) Sitecore Shared Source project mentioned previously for accessing parameters passed to a sublayout.



Rather than accessing the context item directly, always code renderings to access the data source, even if you expect that the context item and the data will always be the same item. Multivariate testing, personalization, conditional rendering, and other Sitecore features can affect the data source, but do not always affect the context item.

You can define additional presentation control parameters to pass additional items similar to the data source.

www.it-ebooks.info



The data source for a presentation component is not the same as the source of a data template field. Developers (and in some cases, users) set the data source of a presentation template to control where it gets its data; developers set the source of data template fields to control the behavior of those fields.

Technically, you can store the data source items for a presentation control anywhere in the content tree. Practically, if the data source items also represent pages, they should exist beneath the home item of the managed site to which those pages belong. Otherwise, you can store data source items in the location of your choice, such as by creating a `/sitecore/content/global` folder with subfolders for the various types of data source items. Alternatively, you can store data source items as children or deeper descendants of the items that use them.

For more information about presentation component data sources, see The Sitecore Presentation Component Reference (`http://bit.ly/o2TdvV`) and my blog posts at `http://bit.ly/n2Xn4X` and `http://bit.ly/qoKzgS`. For instructions for adding presentation component data sources to Sitecore's internal link management database, see my blog post at `http://bit.ly/oZ6Z5j`.

Conditional Rendering and Personalization

You can use conditional rendering to personalize and otherwise dynamically control the experience of users of managed sites. Conditional rendering uses the Sitecore rules engine, which provides a browser-based user interface to define conditional rendering logic.

Before rendering a response to an HTTP request, the layout engine invokes the rules engine to apply conditional rendering rules that can invoke the following actions:

- **Hide Rendering** — Remove a presentation control
- **Set Data Source** — Set the data source of a presentation control
- **Set Parameters** — Pass parameters to a presentation control
- **Set Placeholder** — Move a presentation control from one placeholder to another
- **Set Rendering** — Replace a presentation control with another

You can define global conditional rendering rules under the `/sitecore/system/Settings/Rules/Conditional Renderings/Global Rules` item, or you can use layout details to define rules for each rendering.



Global conditional rendering rules that do not apply to most pages consume unnecessary central processing unit (CPU) capacity and other machine resources.

You can use the Content Editor to define conditional rendering rules for an individual presentation control. For example, you might want to set the `Winner` property of a web control to `true` if the visitor came to the site by searching for the term *sitecore*.

www.it-ebooks.info

First, to define a conditional rendering rule:

1. Create a conditional rendering rule definition item under the `/sitecore/system/Marketing Center/Personalization/Rules/Persona` item using the `System/Rules/Conditional Rendering Rule` data template. For example, if the visitor came to the site using search terms that included the search keyword *sitecore*, you could create a conditional rendering rule definition item named Searched for Sitecore as shown in Figure 3-6.

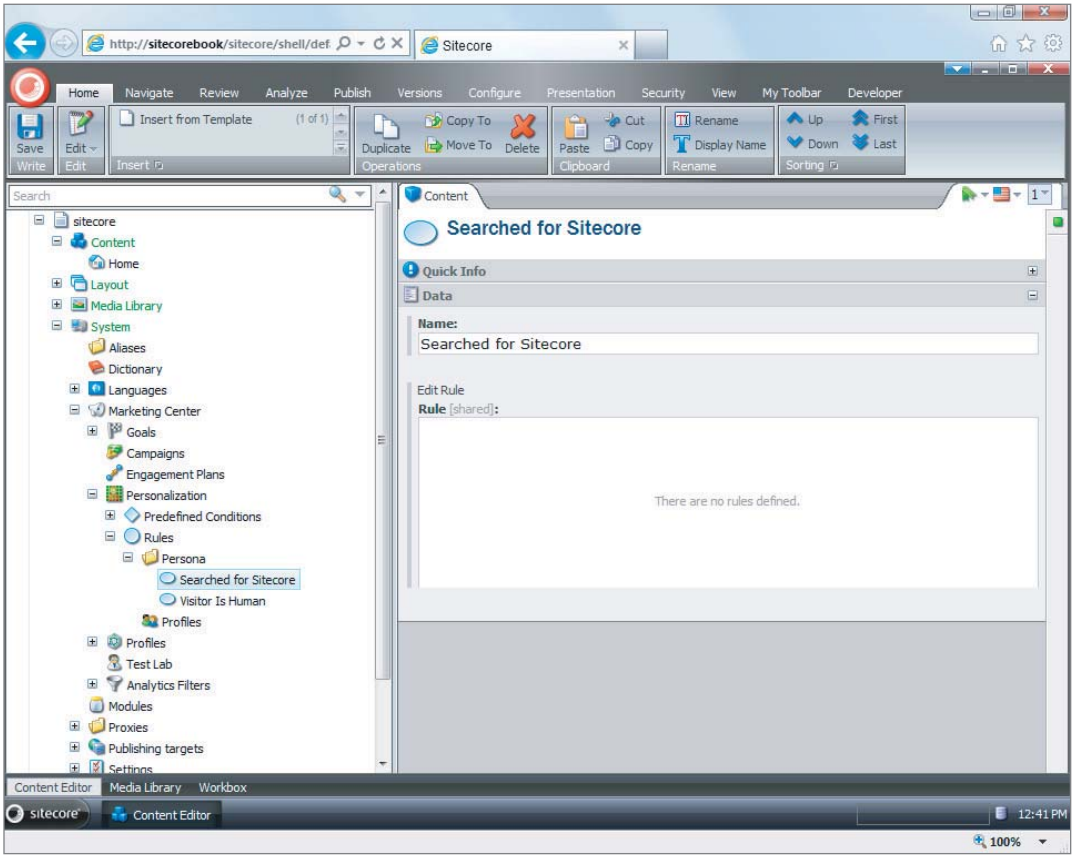


FIGURE 3-6

2. Click Edit Rule above the Rule field in the Data section. The Rule Set Editor appears as shown in Figure 3-7.

3. Select conditions from the list on the left and actions to invoke under those conditions from the list on the right to add them to the rule definition at the bottom. For example, click the Where The Search Keywords String Compares To The Specific Value condition and the Set Parameters To Value action. Figure 3-8 shows the Rule Set Editor after a condition and action are selected.

www.it-ebooks.info

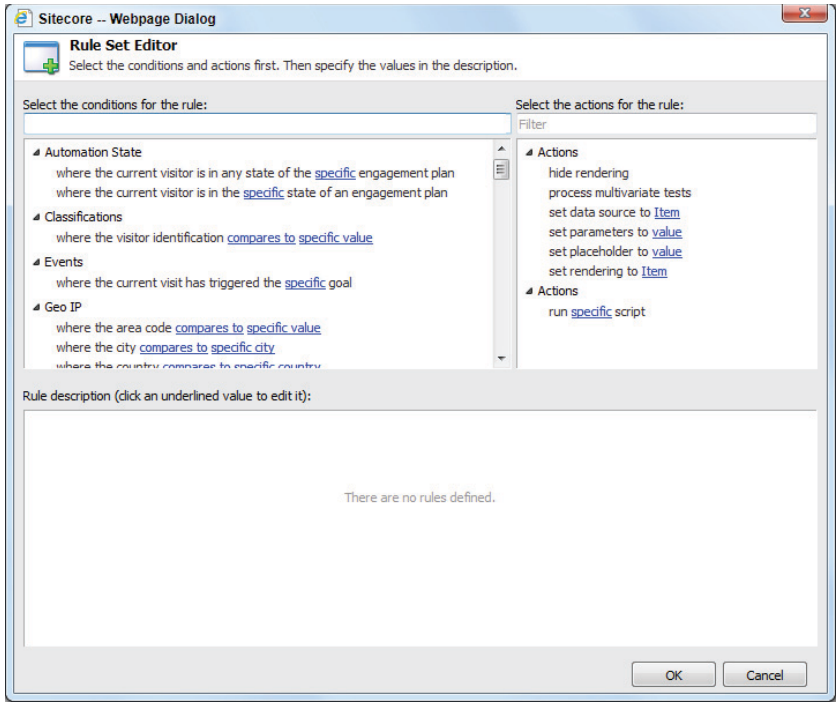


FIGURE 3-7

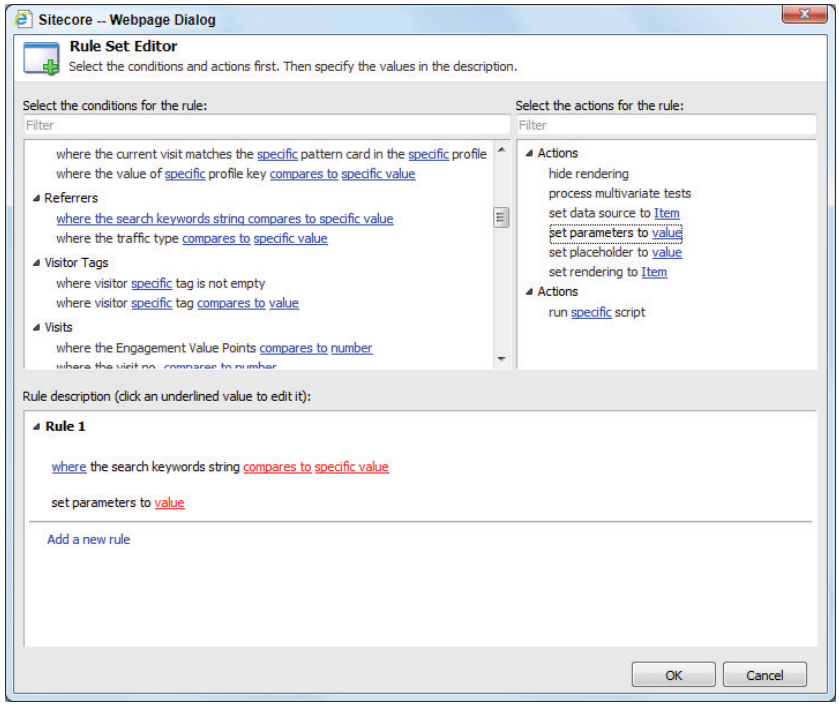


FIGURE 3-8

www.it-ebooks.info

4. In the rule definition at the bottom of the Rule Set Editor, click each of the links to set values for each parameter. For example, click Compares To. The Select Comparison dialog appears as shown in Figure 3-9.

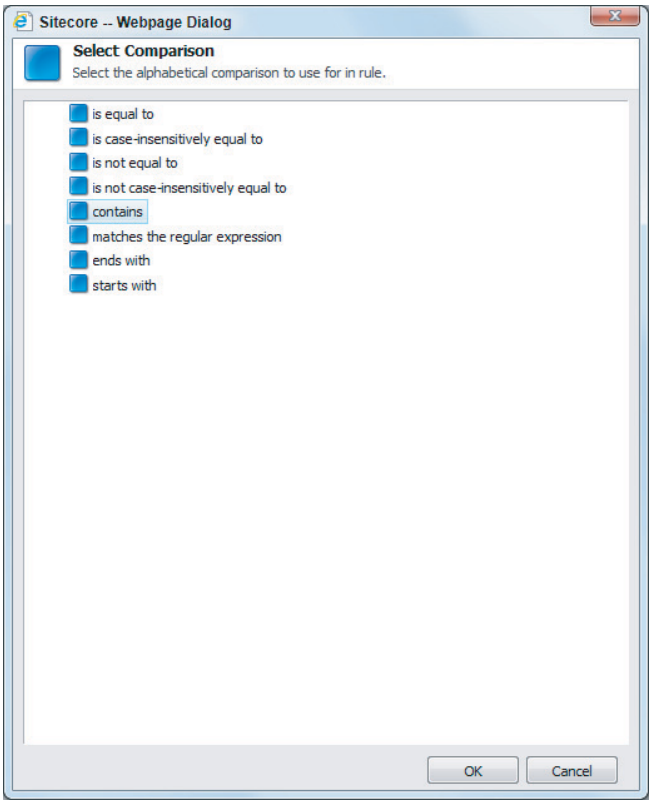


FIGURE 3-9

5. Select Contains, and then click OK. The Select Comparison dialog disappears and you return to the Rule Set Editor. Click Specific Value. The Enter value dialog appears as shown in Figure 3-10.

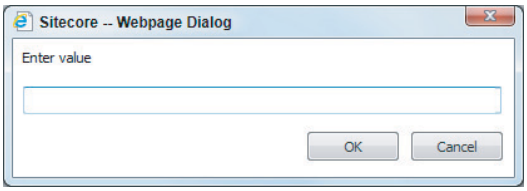


FIGURE 3-10

www.it-ebooks.info

6. Enter sitecore, and then click OK. You see the Rule Set Editor. Click Value. The Enter value dialog appears as shown in Figure 3-10. Enter **winner=true**, and then click OK. You see the Rule Set Editor as shown in Figure 3-11.

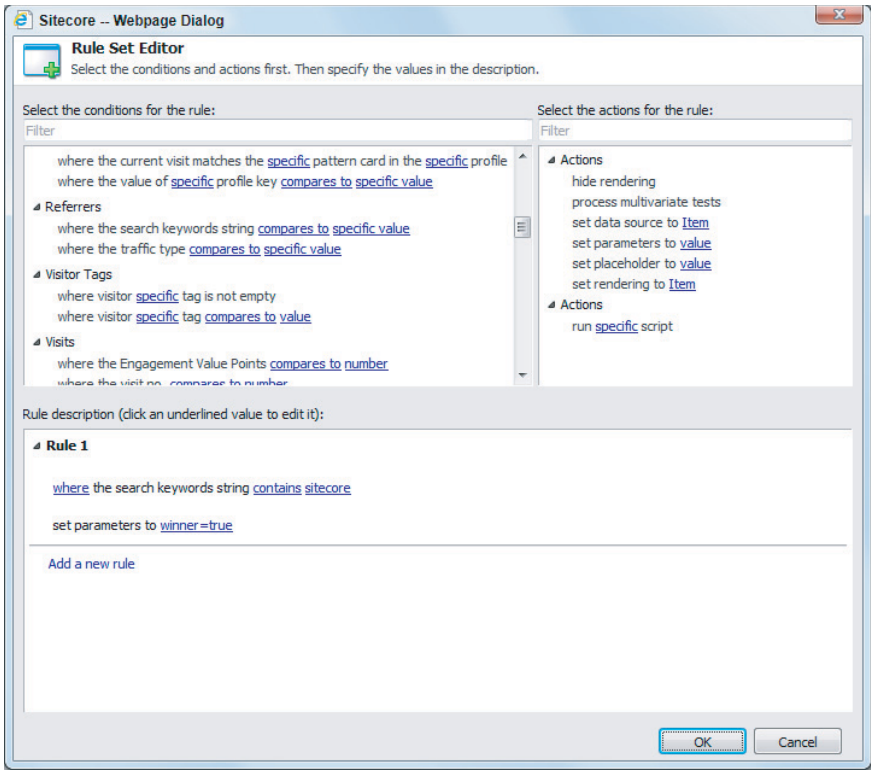


FIGURE 3-11

Next, to apply the conditional rendering rule to a presentation control, you may first need to enable personalization features in the Content Editor. To do so:

1. Click the Sitecore logo in the upper-left corner of the Content Editor. A menu appears as shown in Figure 3-12.
2. Click Application Options. The Application Options dialog appears as shown in Figure 3-13.
3. Click the View tab, then select Show the Personalization Section, and then click OK. The Application Options dialog disappears and you return to the Content Editor.

Finally, to apply the conditional rendering rule to an item in the Content Editor:

1. Select the item for which to apply the conditional rendering rule.

www.it-ebooks.info

2. Click the Presentation tab, and then click Details in the Layout group. The Layout Details dialog appears as previously shown in Figure 3-3.

3. Click the presentation control. The Control Properties dialog appears as shown in Figure 3-14.

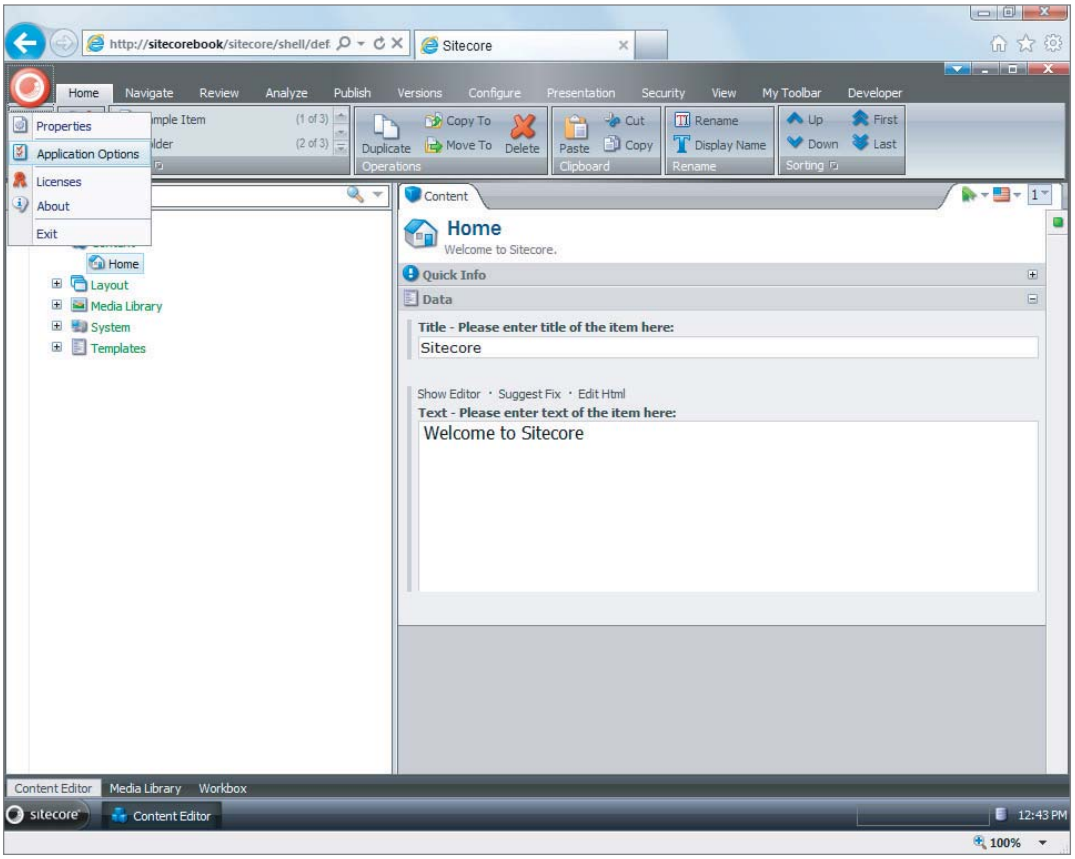


FIGURE 3-12

4. Select the conditional rendering rule in the Personalization section, and then click OK. The Control Properties dialog disappears and you return to the Layout Details dialog.

5. Click OK. The Layout Details dialog disappears and you return to the Content Editor.

For more information about the Sitecore rules engine and conditional rendering, see Chapter 7, The Sitecore Rules Engine Cookbook (`http://bit.ly/peToK6`), and my blog post at `http://bit.ly/nvy2fX`.

www.it-ebooks.info

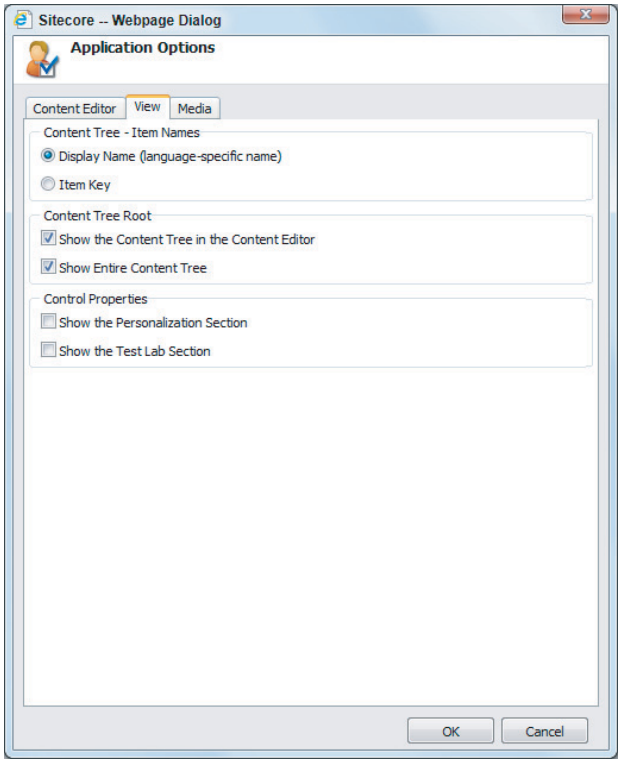


FIGURE 3-13

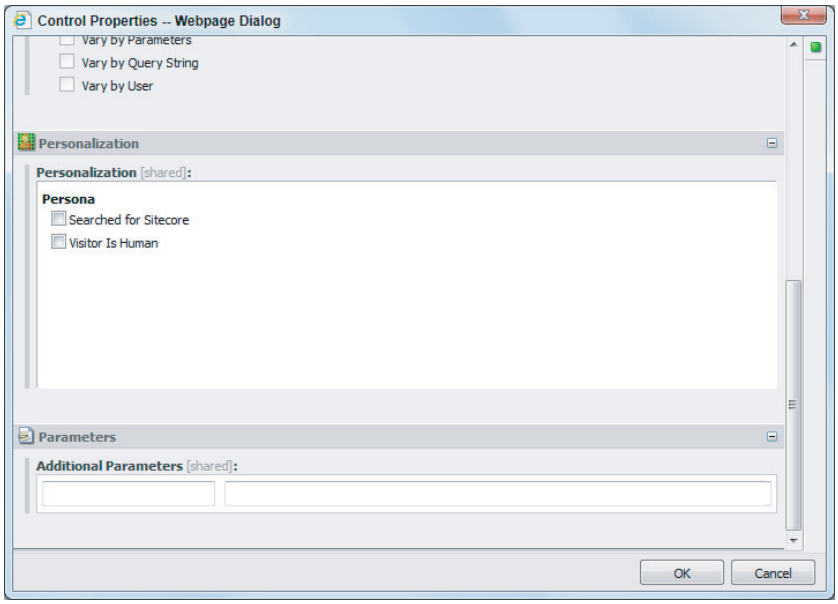


FIGURE 3-14

www.it-ebooks.info

The FieldRenderer Web Control and the renderField Pipeline

You can use the `FieldRenderer` web control (`Sitecore.Web.UI.WebControls.FieldRenderer`) to invoke the `renderField` pipeline against the value of a field in an item. The `renderField` pipeline transforms internal URLs containing IDs for both media and content items to friendly URLs, and adds inline editing, debugging, or other features in the Page Editor, the Sitecore browser-based debugger, and other user interfaces, and otherwise dynamically manipulates field values at runtime.



If you do not use the FieldRenderer web control to retrieve the values of RTE fields (or any other construct that invokes the `renderField` *pipeline or the appropriate APIs), field values could contain URLs with query string parameters containing IDs rather than friendly URLs. XSL extension controls such as* `<sc:text>` *and the* `sc:field()` *XSL extension method invoke the* `renderField` *pipeline.*

You can bind the FieldRenderer web control to a layout or sublayout with code such as the following, which renders the field named Title from the `/sitecore/content/home` item:

```
<sc:FieldRenderer runat="server"
  datasource="/sitecore/content/home" fieldname="title" />
```

If you do not specify the `datasource` property, the FieldRenderer web control retrieves the field from the context item by default. Alternatively, you can use the static `Render()` method of the `Sitecore.Web.UI.WebControls.FieldRenderer` class. For example, to retrieve the field named Title from the context item:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
string markup = Sitecore.Web.UI.WebControls.FieldRenderer.Render(item, "title");
```

The Page Mode

Your presentation components can generate different output depending on the Sitecore features active for the current user. You can check the page mode and generate different output when the user accesses the Page Editor, Preview, or Sitecore browser-based debugger, or when specific features are visible in those interfaces, without that output appearing on the published site.

You can use the following properties of the `Sitecore.Context.PageMode` property to determine the active page modes. To determine the page mode in XSL, you can use the `sc:pageMode()` XSL extension method with the values shown in parentheses:

- `IsDebugging` (`/debug`) — The browser-based debugger is enabled
- `IsNormal` (`/normal`) — No CMS features are enabled (the user is accessing the published site)
- `IsPageEditor` (`/pageEditor`) — The user is in the Page Editor
- `IsPageEditorEditing` (`/pageEditor/edit`) — The user is inline editing in the Page Editor
- `IsPreview` (`/preview`) — The user is in the Preview user interface
- `IsProfiling` (`/IsProfiling`) — Performance profiling is active

www.it-ebooks.info

For example, if you use a field named Title in the data template to allow the user to specify a value for the `<title>` element in an HTML page, that value only appears in the title bar of the browser, which the user cannot edit in the Page Editor. You can use code such as the following to expose an inline editing field for that value elsewhere in the page (assuming the `output` variable is the `System.Web.UI.HtmlTextWriter` argument passed to the `DoRender()` method of a web control):

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;

if (Sitecore.Context.PageMode.IsPageEditorEditing)
{
  output.Write(Sitecore.Web.UI.WebControls.FieldRenderer.Render(item, "title"));
}
```

Or in XSL:

```
<xsl:if test="sc:pageMode()/pageEditor/edit">
  <sc:text field="title" />
</xsl:if>
```

For more information about the page mode, see The Sitecore Client Configuration Cookbook (`http://bit.ly/n7fJbq`).

FUNDAMENTAL SITECORE APIS

You can achieve incredible results with Sitecore with minimal knowledge of its APIs. As described in The Sitecore Context section earlier in this chapter, the `Sitecore.Context` class exposes a number of important static properties of types that you will use throughout your Sitecore development. These properties include `Sitecore.Context.Item` (the context item), `Sitecore.Context.Site` (the context site), and `Sitecore.Context.Database` (the context database). For more information about the APIs described in this section, see The Sitecore Content API Cookbook (`http://bit.ly/eREbAx`), The Sitecore Data Definition API Cookbook (`http://bit.ly/nmUVTS`), and The Sitecore Presentation Component API Cookbook (`http://bit.ly/qTgZ0K`).



Unless otherwise specified, all APIs described in this book exist in the `Sitecore.Kernel.dll` *assembly, the* `Sitecore.Client.dll` *assembly, or possibly the* `Sitecore.Analytics.dll` *assembly.*

Sitecore.Configuration.Factory

You can use the static `Sitecore.Configuration.Factory` class to retrieve objects configured using the Sitecore configuration factory, which Sitecore uses to determine the .NET types to implement specific features. The static `Sitecore.Configuration.Factory` class includes methods such as the following:

- `GetDatabase()` — Retrieve a named database
- `GetDatabaseNames()` — Retrieve a list of the names of all databases
- `GetDatabases()` — Retrieve all databases

www.it-ebooks.info

- `GetDomain()` — Retrieve the named Sitecore security domain
- `GetDomainNames()` — Retrieve a list of the names of all security domains
- `GetIDTable()` — Retrieve the IDTable used to map Sitecore IDs to identifiers in external systems
- `GetItemComparer()` — Retrieve a comparer that implements the child sorting rule for an item
- `GetLinkDatabase()` — Retrieve the links database
- `GetMasterVariablesReplacer()` — Retrieve an object that replaces tokens in standard values, and branch templates (including the names of items in branch templates) on item creation
- `GetSiteNames()` — Retrieve a list of the names of the managed sites
- `GetSite()` — Retrieve information about the named site using the same class as `Sitecore.Context.Site`

For examples that use the `Sitecore.Configuration.Factory` class, see the following sections. For more information about the configuration factory, see Chapter 7.

Sitecore.Data.Database

The `Sitecore.Data.Database` class represents a Sitecore database, such as the Master or Core database or a publishing target database. You typically access the context database (`Sitecore.Context.Database`), the content database (`Sitecore.Context.ContentDatabase`, for use in Sitecore user interface components), or a named database. For example, to access the context database, you can use code based on the following:

```
Sitecore.Data.Database context = Sitecore.Context.Database;
```

To access the Master database by name:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
```



Always use Sitecore APIs to access Sitecore databases. Avoid SQL queries directly against a Sitecore database.

The name of the database that you pass to the `GetDatabase()` method of the `Sitecore.Configuration.Factory` static class is not the name of the database on the database server. The argument passed to this method is a token that maps to the actual database connection string (for SQL Server, defined in the `/App_Config/ConnectionStrings.config` file under the document root subdirectory of the IIS website hosting the Sitecore solution). Using tokens instead of actual database connection information allows the same code to run in different environments, with different connection string information for those tokens.

www.it-ebooks.info

Sitecore.Data.Items.Item

The `Sitecore.Data.Items.Item` class represents an item in a Sitecore database. Presentation components typically access the context item, the data source item for the presentation component, and/or any other specific items either hard-coded, passed to the presentation component as parameters, stored in fields of the item or in items somehow related to the item, or otherwise determined.



The `Sitecore.Data.Items.Item` *class actually represents a version of an item in a language. Properties of the* `Sitecore.Data.Items.Item` *class expose other languages and versions of the item.*

You can use the `GetItem()` method of the `Sitecore.Data.Database` class to retrieve the current version of an item in the context language from a database using either its path or its ID as the first parameter. For example, you can use code based on the following to retrieve the `/sitecore/content/home` item from the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Diagnostics.Assert.IsNotNull(master, "master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
```



Sitecore ignores character case when retrieving items. Whenever possible, reference items by ID rather than by path.

The `GetItem()` method of the `Sitecore.Data.Database` class provides additional signatures that you can use to retrieve items using the unique IDs of items or their paths, and to specify a language and version if needed. Except in Sitecore user interfaces such as Preview, when passed only the ID or path to an item, the `GetItem()` method of the `Sitecore.Data.Database` class returns the latest version of the specified item in the context language from that database. If you specify a language as the second parameter to the method, it returns the latest version of the item in that language. If you additionally specify a version identifier as the third parameter to the method, it returns that version of the item in the language specified by the second parameter.

The `Sitecore.Data.Items.Item` class exposes a number of properties, including the following:

- `Axes (Sitecore.Data.Item.ItemAxes)` — Exposes items related to the item, such as its ancestors and descendants
- `Children (Sitecore.Collections.ChildList)` — Lists the children of the item
- `Database (Sitecore.Data.Database)` — References the database containing the item
- `DisplayName (System.String)` — Optionally contains a user-friendly name for the item
- `Fields (Sitecore.Collections.FieldCollection)` — Provides access to the fields of the item
- `ID (Sitecore.Data.ID)` — Identifier of the item within the Sitecore database

www.it-ebooks.info

- `Key (System.String)` — Name of the item in lowercase
- `Language (Sitecore.Globalization.Language)` — The language of the item
- `Languages (Sitecore.Globalization.Language[])` — Languages available for the item
- `Links (Sitecore.Links.ItemLinks)` — Links from this item to other items as reported by the Sitecore links database, including broken links
- `Name (System.String)` — Name of the item
- `Parent (Sitecore.Data.Items.Item)` — Parent of the item, or `null` for the root item `/sitecore`
- `ParentID (Sitecore.Data.ID)` — ID of the parent item
- `Paths (Sitecore.Data.ItemPath)` — An object of the `Sitecore.Data.ItemPaths` class containing members that expose information about the item, such as the `FullPath` property containing the full path to an item and the `IsAncestorOf()` method that returns `true` if the item specified by the first argument is an ancestor of this item
- `Source (System.String)` and `SourceUri (Sitecore.Data.ItemUri)` — Source of a cloned item
- `Statistics (Sitecore.Data.Items.ItemStatistics)` — Provides access to properties of the item such as its creator, creation date and time, date and time of last update, and the user who performed that update
- `Template (Sitecore.Data.Items.TemplateItem)` — Data template associated with the item
- `TemplateID (Sitecore.Data.ID)` — ID of the data template associated with the item
- `TemplateName (System.String)` — Name of the data template associated with the item
- `Uri (Sitecore.Data.ItemUri)` — Identifying information about the item (its database, ID, language, and version)
- `Version (Sitecore.Data.Version)` — Provides information about the version of the item
- `Versions (Sitecore.Data.Items.ItemVersions)` — Provides access to all versions of the item
- `Visualization (Sitecore.Data.Items.ItemVisualization)` — Abstracts layout details of the item

You can use the `Fields` property of the `Sitecore.Data.Items.Item` class to access the fields of an item, or you can access the fields using the implicit collection exposed by the class. The difference is that the `Fields` property returns `null` for fields that do not exist, while the item returns an empty string. For an example that uses the `Fields` property, see the next section. The following code demonstrates how you can use the implicit collection to retrieve a field value:

```
string text = item["FieldName"];
// text contains either a value or an empty string
```

www.it-ebooks.info

Sitecore.Data.Fields.Field and Related Classes

You can use the `Sitecore.Data.Fields.Field` class to access any field in an item through the `Fields` property of `Sitecore.Data.Items.Item` objects. You can use code based on the following to determine whether the field named `FieldName` exists in an item retrieved previously:

```
Sitecore.Data.Fields.Field field = item.Fields["FieldName"];

if (field == null)
{
 // field does not exist
  return;
}

text = field.Value;
// text is either a value or an empty string
```



Sitecore lazily loads the `Fields` *collection of the* `Sitecore.Data.Items.Item` *class, which means it may not contain standard values or null values for fields with no standard value, and may not contain other inherited field values such as cloned field values. To force Sitecore to load all field values for an item, call the* `ReadAll()` *method of the* `Fields` *property of the* `Sitecore.Data.Items.Item` *class.*

Sitecore ignores character case when retrieving fields by name, meaning that you can refer to a field by name in code using your choice of uppercase and lowercase. Whenever possible, reference fields by ID rather than by name.

You can use additional classes in the same namespace, such as `Sitecore.Data.Fields.ImageField`, to access specific types of fields. For example:

```
Sitecore.Data.Fields.ImageField image = item.Fields["Image"];

if (image != null && image.MediaItem != null)
{
  // process image and/or image.MediaItem
}
```

While you can access any field value as a string, you can use the following classes in the `Sitecore.Data.Fields` namespace to access the specified types of fields using typed objects rather of simple text values:

- `CheckboxField` — Access fields of the Checkbox type
- `Field` — Access fields of any type, especially fields that store simple text values (you can use this class to access any field)
- `DateField` — Access fields of the Date and Datetime types
- `FileDropAreaField` — Access fields of the File Drop Area type
- `FileField` — Access fields of the File type
- `ImageField` — Access fields of the Image type

www.it-ebooks.info

- `LinkField` — Access fields of the General Link type
- `MultilistField` — Access fields of the Checklist, Multilist, Treelist, and TreelistEx types
- `WordDocumentField` — Access fields of the Word Document type
- `XmlField` — Access any type of field that stores an XML structure

The IFrame field type enables you to embed custom applications as fields in data templates. You are responsible for defining the storage format for IFrame field types, and for processing field values in that format. For more information about the IFrame field type, see The Sitecore Client Configuration Cookbook (`http://bit.ly/qS8Dc1`).

Sitecore.Data.ID

The `Siteore.Data.ID` class represents the unique identifier of a Sitecore item. To retrieve an item, you can pass an object of type `Sitecore.Data.ID` to the `GetItem()` method of the `Sitecore.Data.Database` class.



Compare items by ID, not by comparing objects of type `Sitecore.Data.Items.Item`.

Sitecore.Data.Items.EditContext, BeginEdit(), and EndEdit()

Before you can update an instance of the `Sitecore.Data.Items.Item` class, Sitecore requires that you either enter a `Sitecore.Data.Items.EditContext` class or that you invoke the `BeginEdit()` method of the `Sitecore.Data.Items.Item` class before the change and the `EndEdit()` method afterward. For example, to update the field named FieldName in the `/sitecore/content/home` item in the Master database:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Diagnostics.Assert.IsNotNull(master, "master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Diagnostics.Assert.IsNotNull(home, "home");

using(new Sitecore.Data.Items.EditContext(home))
{
  item["FieldName"] = "value";
}
```



Code that creates or updates items should run only in the content management environment (not in the content delivery environment), and should update the Master database (or in some rare cases, the Core database). If you write only to a publishing target database such as the default Web database, subsequent publishing from the Master database can remove those updates.

You can also use the `Sitecore.Data.Fields.Field` class and the other classes in the `Sitecore.Data.Fields` namespace to update field values.

www.it-ebooks.info

You can use the `Add()` method of the `Sitecore.Data.Items.Item` class to insert an item based on a data template or any number of items based on a branch template under an existing item. For example, to add a child named ChildName under the `/sitecore/content/home` item in the Master database using the same data template as that `/sitecore/content/home` item, you can use code based on the following:

```
Sitecore.Data.Database master = Sitecore.Configuration.Factory.GetDatabase("master");
Sitecore.Diagnostics.Assert.IsNotNull(master, "master");
Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
Sitecore.Diagnostics.Assert.IsNotNull(home, "home");
Sitecore.Data.ID template = home.TemplateID;
Sitecore.Data.Items.Item child = home.Add("ChildName", template);
```

You can then place the item in an editing state and update field values as described in the previous section.

For information about importing content into the Sitecore ASP.NET CMS, see my blog post at `http://bit.ly/qnihna`.

Sitecore.Data.Items.CustomItem

You can use the `Sitecore.Data.Items.CustomItem` abstract class as a base class for your own classes that represent items based on specific data templates. For example, if you have a data template for news articles, instead of accessing articles with the `Sitecore.Data.Items.Item` class, you can create a class that inherits from the `Sitecore.Data.Items.CustomItem` class and exposes properties that encapsulate access to the underlying item and its fields, such as a string for the title and a `Datetime` for the article date.

The `InnerItem` property of the `Sitecore.Data.Items.CustomItem` class specifies the `Sitecore.Data.Items.Item` associated with the `Sitecore.Data.Items.CustomItem` object. Typically, your constructor sets the `InnerItem` property. Your class should then provide getters for the relevant fields of the data template. For convenience, you can provide a method to convert from `Sitecore.Data.Items.Item` to your type implicitly, as shown here:




```
namespace SitecoreBook.Data.Items
{
  using System;

  public class NewsArticleItem : Sitecore.Data.Items.CustomItem
  {
    public NewsArticleItem(Sitecore.Data.Items.Item innerItem) : base(innerItem)
    {
      // if desired, validate the data template associated with innerItem
      // and throw an exception if the item is associated with a different template.
    }

    public static implicit operator NewsArticleItem(
      Sitecore.Data.Items.Item innerItem)
    {
      return new NewsArticleItem(innerItem);
    }

    public string Title
    {
```

www.it-ebooks.info

```
        get
        {
          return this.InnerItem["title"];
        }
      }

      public DateTime Date
      {
        get
        {
          string parse = this.InnerItem["date"];

          if (String.IsNullOrEmpty(parse))
          {
            return DateTime.MinValue;
          }

          return Sitecore.DateUtil.IsoDateToDateTime(parse);
        }
      }
    }
}
```

NewsArticleItem.cs

You can use your class in place of the `Sitecore.Data.Items.Item` class. For example, to access the context item using the `Sitecorebook.Data.Items.NewsArticleItem` class, you can use code based on the following

```
Sitecore.Data.Database db = Sitecore.Context.Database;
Sitecore.Diagnostics.Assert.IsNotNull(db, "db");
Sitecorebook.Data.Items.NewsArticleItem news = db.GetItem(
  Sitecore.Context.Item.ID);
```

You can also provide setters for the field values. You can take one of three approaches to place the item in an editing state before making the change:

- Require the user to place the `InnerItem` in an editing state, and to commit those changes after calling the setters. This is not very friendly to developers.
- In each setter, put the item in an editing state, change the field value, and commit the field. This is relatively expensive if you plan to update a number of fields.
- Add `BeginEdit()`, `CancelEdit()`, and `EndEdit()` methods to your class to call the corresponding methods of the `Editing` property of the `InnerItem` object.

Sitecore ID Constants

To reduce hard-coding, especially of item paths, Sitecore provides a number of static classes that expose properties that act as constants to define the IDs of elements. The following list includes the most important classes of this type:

- `Sitecore.FieldIDs` — IDs of common fields, primarily defined by the standard template
- `Sitecore.ItemIDs` — IDs of system items

www.it-ebooks.info

- `Sitecore.TemplateIDs` — IDs of system data templates

For example, to retrieve the `/sitecore/content` item in the context database, you can use code based on the following:

```
Sitecore.Data.Database db = Sitecore.Context.Database;
Sitecore.Diagnostics.Assert.IsNotNull(db, "db");
Sitecore.Data.Items.Item item = db.GetItem(Sitecore.ItemIDs.ContentRoot);
```



Implement classes similar to `Sitecore.ItemIDs` *to expose IDs rather than hard-coding IDs or paths.*

SITECORE DATA ACCESS TECHNIQUES

Depending on your requirements, you can use a variety of techniques to access Sitecore items. You can:

- Retrieve individual items from a Sitecore database
- Retrieve items using a Sitecore query
- Retrieve items using a fast query
- Retrieve items using a Lucene search index
- Retrieve items using the internal links database
- Use recursion to access all descendants of an item

Each of these approaches has advantages and disadvantages as described in the following sections. For more information about Sitecore data access techniques, see my blog post at `http://bit.ly/nf2G5z`.

Direct Item API Access

As described previously, you can use the `GetItem()` method of the `Sitecore.Data.Database` class to retrieve an item. As the first parameter, you can pass the ID of the item or its path. You can specify additional parameters to retrieve the item in a specific language and a specific version within a language.



To maximize performance, whenever possible, use item IDs rather than paths. If you use a path, Sitecore must determine the associated ID, which is often unnecessary. If you know the path or ID of the item, using the `GetItem()` *method of the* `Sitecore.Data.Database` *class is the best technique for selecting an item.*

You can also access items relative to other items, such as by using the `Parent` and `Children` properties of the `Sitecore.Data.Items.Item` class to access the parent or children of the item, or the `Axes` property that exposes methods such as `GetDescendants()`.

www.it-ebooks.info

Sitecore Query Access

You can use a Sitecore query to select zero or more items that match specific requirements from a Sitecore database. You can pass queries to the `GetItems()` and `SelectSingleItem()` methods of the `Sitecore.Data.Database` class. You can also use the `SelectItems()` and `SelectSingleItem()` methods of the `Axes` property of the `Sitecore.Data.Items.Item` class to select items relative to an existing item.

AVOIDING ISSUES WITH SITECORE QUERIES

The following techniques will help you avoid problems when working with queries.

- For optimal performance, construct queries relative to an item at the lowest level possible in the information architecture of the site rather than from the root item. The `SelectItems()` and `SelectSingleItem()` methods of the `Axes` property of the `Sitecore.Data.Items.Item` class automatically create queries relative to that item. Avoid the descendant axis and union operators.
- The `Query.MaxItems` setting in the Web.config file limits the number of items returned by a Sitecore query. For performance and usability, the default value of the `Query.MaxItems` setting is `100`. If you find drop-down and other lists restricted to 100 items, you can change the `Query.MaxItems` setting. If you set it to `0`, Sitecore does not limit query results.
- In some cases, such as when setting the Source property of a data template field to a Sitecore query, you must precede a query with the `query:` prefix.

For more information about Sitecore queries, including their syntax and examples, see The Sitecore Data Definition Reference (`http://bit.ly/jIUdfw`) and "Using Sitecore Query" (`http://bit.ly/ofjlX0`) on the Sitecore Developer Network (SDN).

Sitecore Fast Query Access

Sitecore fast query works much like Sitecore query, but rather than using abstraction APIs fast query invokes SQL queries directly against the database, resulting in better performance than standard Sitecore queries. To indicate that Sitecore should use fast query, preface the query with the token `fast:` or `query:fast:`.



Sitecore fast query requires Microsoft SQL Server (as opposed to Oracle).

For more information about Sitecore fast query, including its syntax, significant limitations, and examples, see `http://bit.ly/o0uXDn`.

www.it-ebooks.info

Lucene Search Index Access

Sitecore includes the Apache Lucene search engine that you can use to identify items in a Sitecore database that match specific criteria. Lucene is typically the most efficient, capable, and flexible out-of-the-box solution for identifying items to process, although your requirements may necessitate some specific configuration. For example, you can use Lucene to identify all items that contain a field that contains the ID of another item, or to implement internal search, faceted navigation, or metadata-driven presentation logic. You can integrate Apache Solr with Sitecore for additional features, and you can also integrate third-party search engines. For more information about Lucene, see `http://lucene.apache.org`. For more information about Solr, see `http://lucene.apache.org/solr`.

If you suspect that the search indexes are corrupt, rebuild them. To rebuild search indexes, in the Sitecore desktop, which you can access as described in Chapter 1, do the following:

1. Click the Sitecore button at the bottom left, and then click Control Panel from the menu that appears. The Control Panel appears as shown in Figure 3-15.

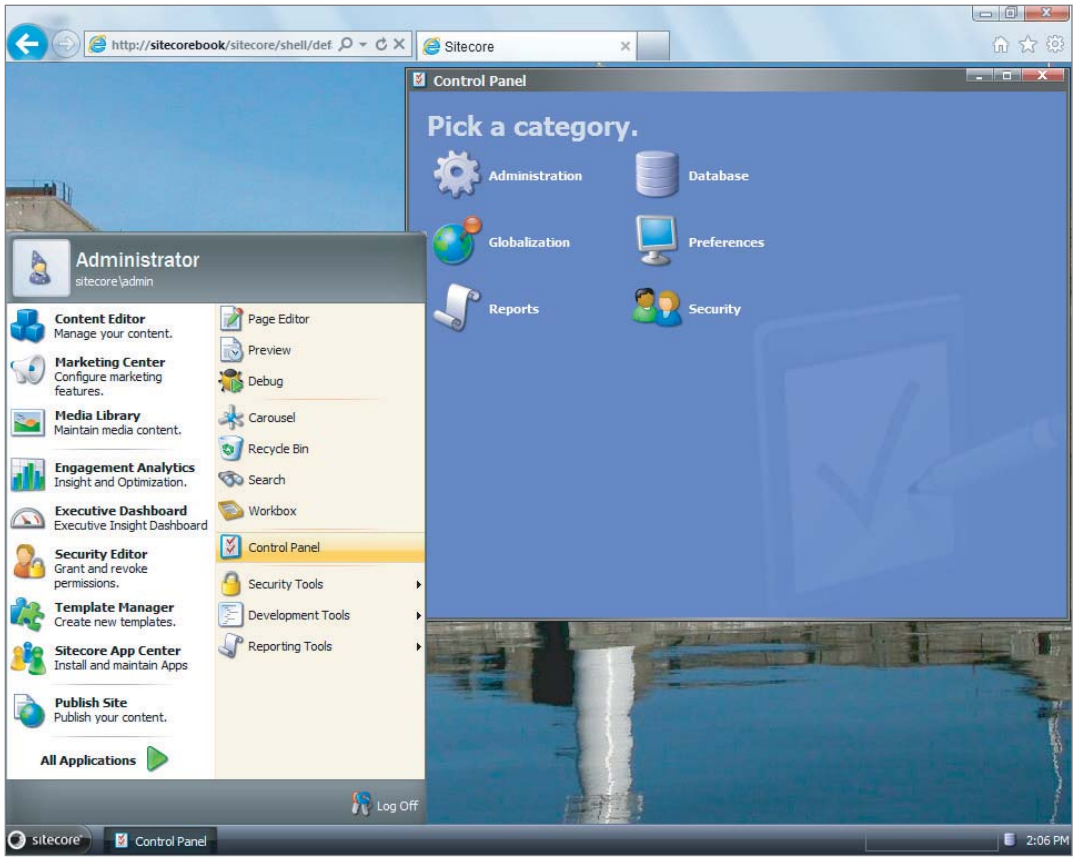


FIGURE 3-15

www.it-ebooks.info

Click Database. The Control Panel refreshes to show database commands as shown in Figure 3-16.

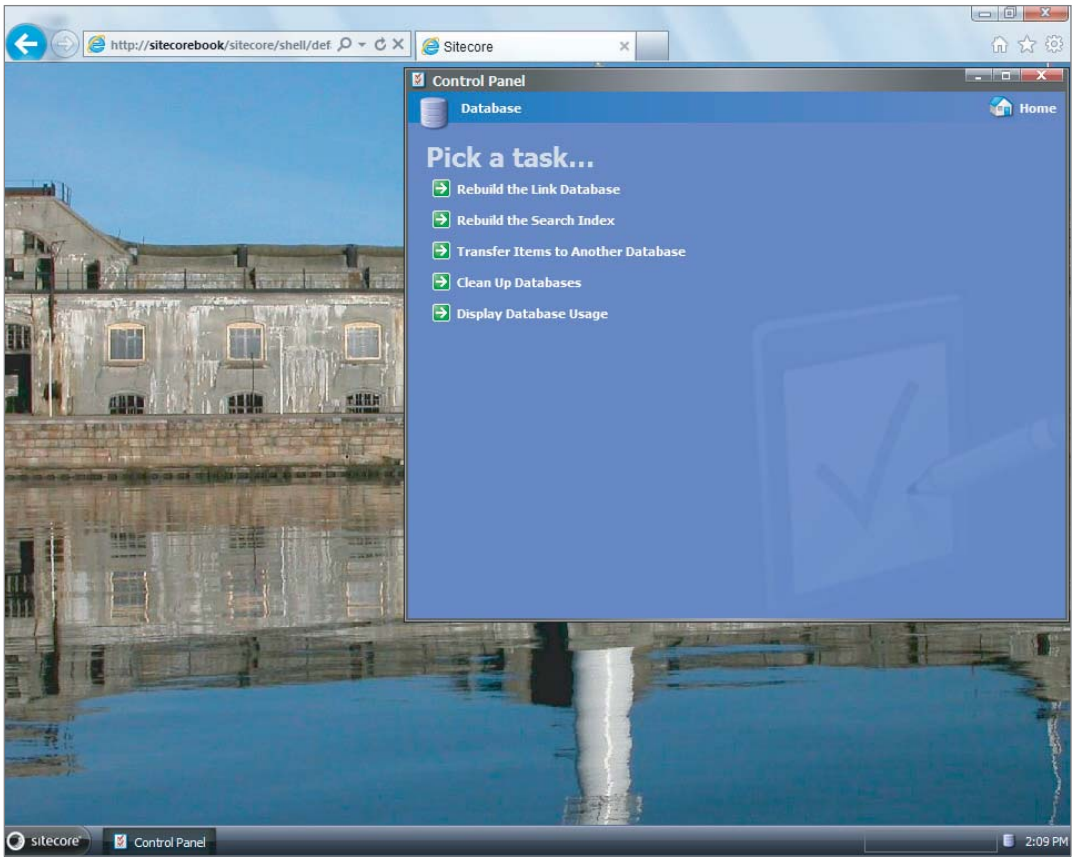


FIGURE 3-16

2. Click Rebuild The Search Index. If a welcome dialog appears, click Next. The Select Search Index dialog appears as shown in Figure 3-17.
3. Select the indexes to rebuild, and then click Rebuild. The Rebuilding dialog appears as shown in Figure 3-18.
4. After Sitecore rebuilds the selected search index or indexes, a completion page appears. Click OK. The Rebuild Search Index dialog closes and you return to the Sitecore desktop. You can close the Control Panel.

www.it-ebooks.info

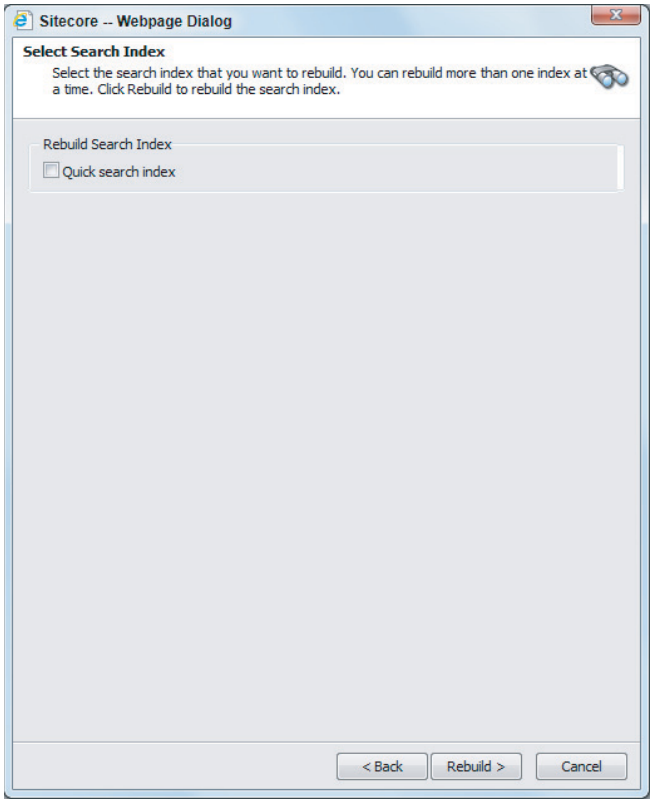


FIGURE 3-17

To rebuild all search indexes through the API, use code based on the following example:

```
Sitecore.Search.SearchConfiguration config =
  Sitecore.Configuration.Factory.CreateObject(
    "search/configuration", true) as Sitecore.Search.SearchConfiguration;
Sitecore.Diagnostics.Assert.IsNotNull(config, "config");

foreach(Sitecore.Search.Index index in config.Indexes.Values)
{
  if (index != null)
  {
    index.Rebuild();
  }
}
```

www.it-ebooks.info

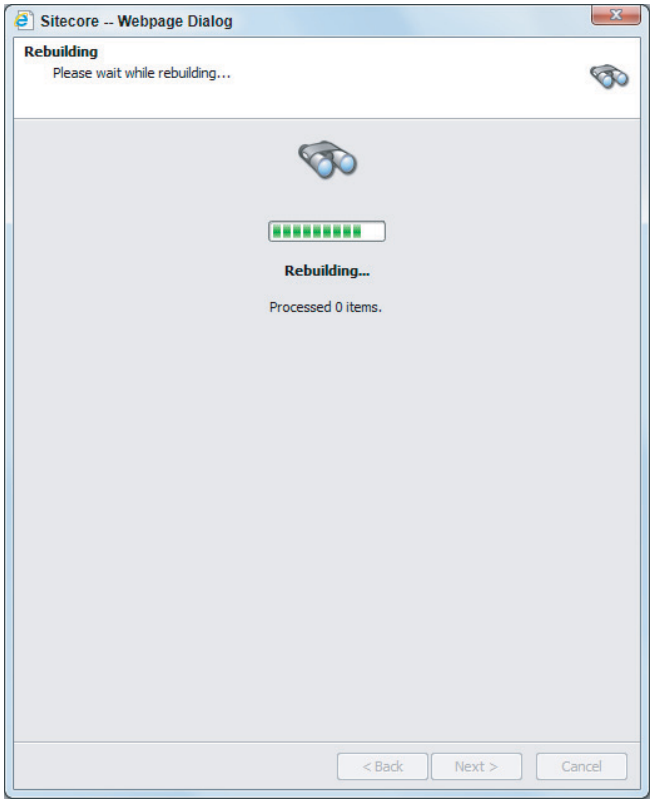


FIGURE 3-18

RUNNING CODE AS AN ADMINISTRATOR

All code invoked within Sitecore runs in the context of a specific user, which defaults to the context user. This user may or may not have the access rights required to perform the requested operation. To ensure that code has sufficient access rights, wrap it with the equivalent of a C# `using` statement that creates an instance of the `Sitecore.SecurityModel.SecurityDisabler` class or an instance of the `Sitecore.Security.Accounts.UserSwitcher` class (in which case the specified account must exist):

```
using (new Sitecore.Security.
     Accounts.UserSwitcher("extranet\\indexer", false))
{
  // code to run as the specified user
}
```

www.it-ebooks.info

If Sitecore fails to rebuild an index, do the following:

1. Move, delete, or rename any subdirectories or files in the subdirectory specified by the `IndexFolder` setting in the Web.config file.
2. Rebuild the search indexes.

For more information about using Lucene, including configuration instructions for solutions that separate the content management environment from the content delivery environment, see Sitecore Search and Indexing (`http://bit.ly/pbA9Pm`). For access to a custom search indexer with additional functionality, see the AdvancedDatabaseCrawler (`http://bit.ly/w4Idmx`) Sitecore Shared Source project.

Internal Links Database Access

Sitecore maintains an internal links database (actually a table in a database) that stores information about relationships among Sitecore items, specifically references to items in field values of other items. When you move, rename, or delete an item, Sitecore uses the links database to determine whether to present a wizard allowing you to update, remove, or break references to that item from other items.

When working with Sitecore items, you often follow references from one item (A) to another (B) contained in the field values of A. For example, a Rich Text field can contain embedded images, which are references to items in the media library, and can contain links to other content items, images, and other types of media items such as `.pdf` (Portable Document Format) files. Selection fields allow the user to select one or more items, and store the names, paths, or ID(s) of those items.

To identify references in fields in an item (A) to another item (B), check for the ID or path of item B in all fields of A (or in the case of fields that store the name of the selected item rather than its ID or path, the name of B). If you have the name or path of an item, pass that ID or path of item B from the field value in item A to the `GetItem()` method of the `Sitecore.Data.Database` class. You can use the links database to walk references backward, to locate all items that reference an item (find all items that link to B).

The table named Links, in the Core database by default, implements the Sitecore Links database, which contains records of internal links in all versions of all items in all languages in all databases. To specify the database to contain the links database, set the `ref` attribute of the first `/configuration/sitecore/LinkDatabase/param` element in the Web.config file to the name of that database.

The `/App_Config/FieldTypes.config` file under the document root of the IIS website hosting the Sitecore solution specifies the type to manage the links database for each data template field type. The `ValidateLinks()` method of each of the types specified by the `type` attribute of a `/configuration/fieldType` element in the `/App_Config/FieldTypes.config` file is responsible for identification of links and broken links in fields of that type. In some cases, multiple types that use a consistent storage format can use a single type to manage internal links.

The Links database maintains records for at least the following field types:

- Checklist
- Datasource

www.it-ebooks.info

- Droplink
- Droplist
- Droptree
- File
- File Drop Area
- General Link
- Grouped Droplink
- Grouped Droplist
- Image
- Internal Link
- Layout
- Multilist
- Rich Text (both embedded images and internal links, including links to media items)
- Template Field Source
- Thumbnail
- Treelist
- TreelistEx
- Version Link
- Word Document

You can use the static `Sitecore.Globals.LinkDatabase.GetReferrers()` method to locate all the items that contain references to another item. For example, to process all links in the context item:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
Sitecore.Diagnostics.Assert.IsNotNull(item, "item");
Sitecore.Links.ItemLink[] links = Sitecore.Globals.LinkDatabase.GetReferrers(item);
```

You are responsible for ordering the results returned from the links database, filtering those results to eliminate any not relevant to your database, language, and version of an item, and ignoring results in fields other than those that you care about or that specify items that no longer exist. You might also want to validate that the current field values still contain each reference.

Normally Sitecore events trigger updates to the links database. If you suspect that the links database is incomplete or invalid, rebuild it.



Depending on the system hardware, the number of items, the number of languages, the number of versions per language, and the number of links in each version, rebuilding the links database can take some time.

www.it-ebooks.info

To rebuild the links database, follow these steps in the Sitecore desktop:

1. Click the Sitecore button at the bottom left, and then click Control Panel from the menu that appears. The Control Panel appears as previously shown in Figure 3-15.
2. Click Database. The Control Panel refreshes to show database commands.
3. Click Rebuild the Link Database. If a welcome dialog appears, click Next. The Rebuild Link Database dialog appears as shown in Figure 3-19.
4. Select the databases for which to rebuild internal links database records, and then click Rebuild. The user interface refreshes for some time to indicate the status of the rebuild process, and then displays a summary of the process.
5. Click Finish. The Rebuild Link Database dialog disappears and you return to the Control Panel in the desktop. You can close the Control Panel.

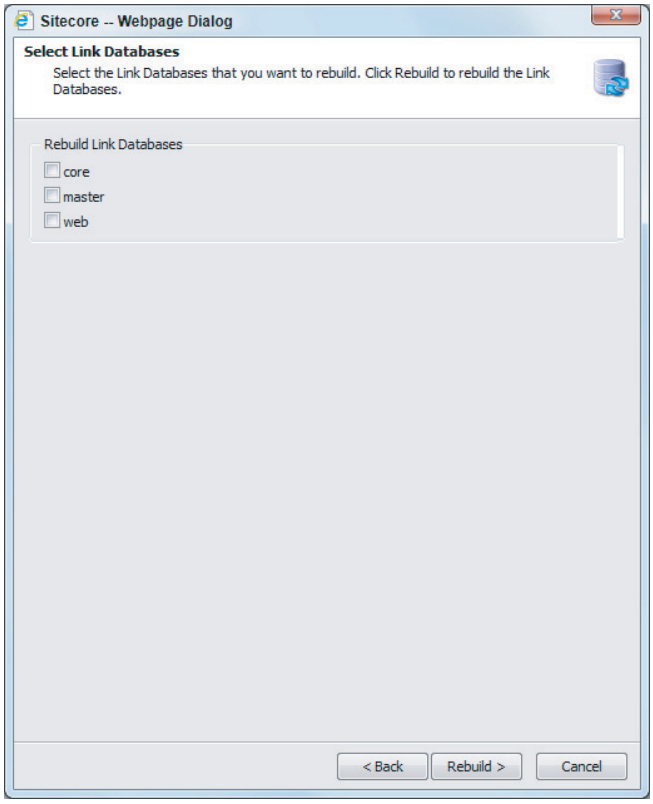


FIGURE 3-19

You can use the `Rebuild()` method of the `Sitecore.Globals.LinkDatabase` static property to rebuild the links database for a single Sitecore database. For example, to rebuild the links database records for all Sitecore databases, you can use code such as the following:

```
foreach(Sitecore.Data.Database database in
  Sitecore.Configuration.Factory.GetDatabases())
```

www.it-ebooks.info

```
{
  Sitecore.Globals.LinkDatabase.Rebuild(database);
}
```

Sitecore does not maintain records in the links database for the data sources of presentation components. Worse, layout details store the paths to data source items rather than storing their IDs. If you rename, move, or delete the data source of a presentation control, Sitecore does not prompt you to update the data sources of presentation components that use that item to the new path. For a sample event handler that updates paths in layout details to IDs, see the Sitecore Developer Network (SDN) forums thread at `http://bit.ly/qsngmX`. To configure Sitecore to maintain records in the links database for the sources of presentation components, see my blog post at `http://bit.ly/oZ6Z5j`. This solution does nothing for dynamic data sources such as queries, relative paths, and conditional rendering.

Recursion

You can use recursion to access the relatives of an item. You can process each descendant of an item, or you can process the ancestors of an item. You can process items in document order (from a root element down) or reverse document order. You can either include or exclude the root item for processing. You can sort sibling items to control the order of processing at each level in the information architecture.

To process all descendants of an item in document order, write a recursive method that accepts an instance of the `Sitecore.Data.Items.Item` class and calls itself for each child of that item, as shown here:




```
private void RecurseDescendant(Sitecore.Data.Items.Item item)
{
  Sitecore.Diagnostics.Assert.IsNotNull(item, "item");
    // process item

  foreach(Sitecore.Data.Items.Item child in item.Children)
  {
     RecurseDescendant(child);
  }
}
```

RecurseBranch.cs

Then pass an item to the recursive method. For example, to process the context item and all its descendants:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
RecurseDescendant(item);
```

To process items in reverse document order, move the processing to after the `foreach` loop.

To exclude the item itself from processing, pass each of its children to the recursive method. For example, to process only the descendants of the context item:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
Sitecore.Diagnostics.Assert.IsNotNull(item, "item");
```

www.it-ebooks.info

```
foreach(Sitecore.Data.Items.Item child in item.Children)
{
  RecurseDescendant(child);
}
```

To control the order of processing sibling items, sort `Sitecore.Collections.ChildList` (`item.Children`), and then iterate that sorted list when calling the recursive method.

Depending on the depth of the tree and the number of items it contains, descendant recursion can use more memory than other means of accessing items. For example, you can use a Lucene index to retrieve the IDs of the items you need rather than the objects of the items, as used in recursion.

To access the ancestors of an item, you can:

- Implement a recursive method
- Implement a while loop that terminates when an instance of `Sitecore.Data.Items.Item` is `null`, and, within the loop, set that item to its parent
- Use the `Sitecore.Data.Items.Item.Axes.GetAncestors()` method

For example, to implement a recursive method that processes an item and all its ancestors in document order, create a method that accepts an item and passes the parent of that item to itself unless that parent is `null`:




```
private void RecurseAncestor(Sitecore.Data.Items.Item item)
{
  Sitecore.Diagnostics.Assert.IsNotNull(item, "item");

  if (item.Parent != null)
  {
    RecurseAncestor(item.Parent);
  }

// process item…
}
```

RecurseAncestors.cs

To process the ancestors of an item in reverse document order, move the processing before the `if` statement. Other than document order or reverse document order, you do not need to consider the order of processing ancestors.

To exclude the item itself from processing, pass its parent to the recursive method. For example, to process only the descendants of the context item, use code such as the following:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
RecurseAncestor(item.Parent);
```

Unlike processing all descendants of an item, unless the logic is extremely intensive, processing all of the ancestors of an item has minimal performance impact.

www.it-ebooks.info

Sitecore Web Services

You can use web services with Sitecore in a number of ways:

- Your presentation controls and other components can invoke any web service (not just Sitecore web services).
- Other systems can invoke the web services provided by Sitecore.
- You can implement additional web services that invoke Sitecore APIs.

Sitecore provides two sets of web services:

- Visual Sitecore Service (Good Old Web Service) at `/sitecore/shell/WebService/service.asmx`
- Sitecore Web Service 2 (Hard Rock Web Service) at `/sitecore/shell/WebService/service2.asmx`



Sitecore CMS provides the Visual Sitecore Service. Sitecore Rocks can install the Hard Rock Web Service.

For more information about Sitecore web services, see The Sitecore Web Service Reference (`http://bit.ly/y09xcC`). For information about implementing web services that invoke Sitecore APIs, see The Sitecore Presentation Component Cookbook (`http://bit.ly/nDo0Ek`).

SYNDICATE CONTENT WITH SITECORE

Sitecore can syndicate content on your published sites, but also provides *client RSS feeds* that syndicate workflow and item update information. For more information about RSS feeds with Sitecore, see The Sitecore Presentation Component Cookbook (`http://bit.ly/nDo0Ek`) and The Sitecore Content Author's Cookbook (`http://bit.ly/qaSgz5`).

Public RSS Feeds

A Sitecore public RSS feed consists of a feed definition item that specifies feed entry items to include in the feed. Feed definition items use the `System/Feeds/RSS Feed` data template. Layout details for the Default device in feed definition items use the `System/Feed Delivery Layout` layout to render an RSS feed from the items specified in the Source field of the feed definition item.



The Source field in definition items for RSS feeds is entirely unrelated to the Source property in definition items for data template fields as described in Chapter 2. The Source field in RSS feed definition items indicates the items to include in RSS feeds; the Source field in data template fields controls aspects of fields in data templates for CMS users.

www.it-ebooks.info

The Source field in the Data section of RSS feed definition items (which may have the title Items in some versions of Sitecore) specifies which items to include in a feed. In the Source field you can specify an individual item or a Sitecore query. If you specify an item, the feed can include any descendants of that item that define layout details for the `System/Feeds/FeedRenderer` web control for the Default device. Feeds sort and retrieve only the newest items automatically.

Feed entry items can use any data template. Layout details for the Feed device for feed entry items pass parameters to the `System/Feeds/FeedRenderer` web control (or any control that inherits from the `Sitecore.Web.UI.WebControls.FeedRenderer` class) to specify the control to render the feed entry item and the fields in the data template for that control to render (using the TitleField, BodyField, and DateField presentation control parameters). To configure presentation for a feed entry item, click the Design command in the Feeds group, on the Presentation tab of the Content Editor.

AVOIDING ISSUES WITH RSS FEEDS

The following tips will help you avoid potential problems with the RSS feeds you create.

- Unlike most commands, which update the item selected in the Content Editor, the Design command in the Feeds group updates layout details in the standard values item for the data template associated with that item. If you use a derivative of the `System/Feeds/FeedRenderer` web control, the Design command adds the default `System/Feeds/FeedRenderer` web control as the first presentation control in layout details for the device named Feed. Use the Design command to configure the default `System/Feeds/FeedRenderer` web control, specify TitleField, TextField, and DateField, and update layout details to use the alternate FeedRenderer; thereafter do not use the Design command for that item.
- The `System/Feeds/FeedRenderer` web control or its derivative should be the first presentation component listed in layout details for the Feed device.
- The `Caching.SmallCacheSize` setting in the Web.config file limits the size of the cache for RSS feed output. If Sitecore does not cache your RSS feeds, you may need to increase the value of this setting. This setting does not appear in the default Web.config file, which may result in a default value smaller than you need to cache your RSS feeds.

By default, feed definition items use the `Sitecore.Syndication.PublicFeed` class to render RSS feeds. The `GetSourceItems()` method of that class determines which items can appear in the feed and the `GetItems()` method limits those results according to the `Feeds.MaximumItemsInFeed` setting in the Web.config file. For an example that dynamically configures the number of items in an RSS feed, see my blog post at `http://bit.ly/pRSbLN`.

To define custom logic to determine which items to include in an RSS feed, create a class that inherits from the `Sitecore.Syndication.PublicFeed` class. In that class, override the

www.it-ebooks.info

`GetSourceItems()` method to determine the items to include. To configure a feed definition item to use your class, enter the signature of your class in the Type field (found in the Extensibility section) of a feed definition item. For an example of a custom RSS feed, see my blog post at `http://bit.ly/nTSHf1`. For an example of a solution that overrides the URLs of feed entry items, see my blog post at `http://bit.ly/pT4WQx`.



Before you can use certain RSS APIs, you need to add a reference to the `System.ServiceModel.Web.dll` *system assembly to your Visual Studio project.*

Client RSS Feeds

Client RSS feeds provide information about items in the CMS to CMS users. For more information about client RSS feeds, see The Sitecore Client Configuration Cookbook (`http://bit.ly/psP2O1`).



Sitecore client RSS feeds use a different infrastructure from public RSS feeds.

Sitecore provides three types of client RSS feeds:

- **Workflow** — Contains an entry for each state change for each item in a specific workflow. The Workflow Feed appears as an RSS icon for each type of workflow in the Sitecore Workbox.
- **Workflow State** — Contains an entry for each time a version enters or leaves a specific workflow state. The Workflow State Feed appears as an RSS icon for each state of each workflow shown in the Workbox.
- **Item Updated** — Contains an entry for each workflow state change for an item or for each new version of the item. The Item Updated Feed appears in the Content Editor if you click the Subscribe command in the Proofing group on the Review tab.

You can set the `ClientFeeds.Enabled` setting in the Web.config file to `false` to disable Sitecore client RSS feeds and remove the corresponding elements from Sitecore user interfaces such as the Workbox. The `ClientFeeds.ItemExpiration` setting in the Web.config file controls how long entries remain in client RSS feeds. The `ClientFeeds.MaximumItemsInFeed` setting in the Web.config file controls the maximum number of items in a client RSS feed.

LAYOUT ENGINE BEST PRACTICES

I believe in the following best practices for the Sitecore layout engine:

- Apply layout details to the standard values of data templates rather than to individual items.
- Do not use ASP.NET master pages.

www.it-ebooks.info

- Apply caching to each presentation control unless one of its ancestors caches under the same caching criteria or criteria less restrictive. When working with nested presentation components, for flexibility, cache the output of individual leaf renderings; for reuse of larger sections of output, cache the output of the parent or higher sublayout.
- For the relevant field types, especially Rich Text Editor (RTE) fields, always use constructs that invoke the `renderField` pipeline to access field values.
- Rather than using attributes of the HTML `<img>` element or uploading multiple versions of an image, use Sitecore dynamic image manipulation options to resize images on the server before transmission to the client.
- Implement complex presentation components as sublayouts or web controls rather than XSL renderings.
- Minimize the number of layouts in favor of sublayouts.
- Avoid caching output at the user level.
- Consider edge caching systems and content delivery networks (CDNs) for optimal performance, especially for media items, but ensure you have the ability to clear those caches. For more information about these types of solutions, see Chapter 6.
- Instead of assuming that the context item is the data source for a rendering, always explicitly access the data source.
- To support internationalization and localization, avoid hard-coding text in presentation components in favor of fields in items or the Sitecore dictionary.



This is not a comprehensive list of recommendations for optimizing performance. For more suggestions, see Chapter 11.

TAKE HOME POINTS

This chapter provided a brief overview of the ASP.NET page lifecycle, the Sitecore layout engine, and fundamental data access APIs, as well as best practices for working with Sitecore presentation components.

The Sitecore context contains information about the current HTTP request, such as the anonymous or known user, the general or specific device used, and the managed site accessed. Sitecore uses layout details in the context item to determine which layout (`.aspx` file) to use to service the request, what ASP.NET controls to bind to placeholders in that layout, and the controls to bind to placeholders nested into that layout using sublayouts. You can use placeholder settings to control which users in the Page Editor can bind which controls to each placeholder.

The types of controls you can bind to a placeholder include sublayouts (`.ascx` files, also called web user controls), web controls, XSL renderings, method renderings, and URL renderings. Of these, sublayouts are the most versatile, while web controls are more lightweight and XSL renderings can offer advantages to nontechnical business users. Developers rarely use URL renderings or method renderings.

www.it-ebooks.info

You can pass parameters to renderings, and you can implement rendering parameters templates to control the types of those parameters. Each rendering also supports a number of properties that control its behavior. One of the most important parameters for each rendering is its data source item, which tells the rendering from where to retrieve data. Through conditional rendering you can control rendering properties dynamically using the Sitecore rules engine.

You can use a number of techniques to access Sitecore items. You can retrieve items from a database by path or ID. You can query for Sitecore items using a syntax based on XPath, and with some limitations you can use the fast query engine. You can use a search engine or the internal links database to determine the items to process. You can access Sitecore through its default web services or through the web services installed by Sitecore Rocks, or you can write your own web services to invoke Sitecore APIs. You can syndicate Sitecore items as RSS feeds, and CMS users can access Sitecore client RSS feeds to get information about items and workflows.

www.it-ebooks.info

4

Sitecore Security

WHAT'S IN THIS CHAPTER?

- Evaluating access rights and security inheritance
- Applying access rights with the Security Editor
- Examining access rights with the Access Viewer
- Understanding security domains, accounts, profiles, and roles
- Integrating and extending the Sitecore security model

Sitecore provides a comprehensive security infrastructure that you can use to secure any item in any Sitecore database. This model uses concepts familiar to security administrators experienced with Windows domains, New Technology File System (NTFS), and Access Control Lists. Concepts of the Sitecore security infrastructure include:

- Access rights
- Users
- User profiles
- Roles (groups)
- Domains
- Security inheritance

After reading this chapter, you will have a greater understanding of the purpose of each access right, as well as of how Sitecore evaluates rights and inheritance.

You can use a number of techniques to integrate Sitecore with external security systems. The Sitecore security model uses the provider pattern, which lets you plug in alternate implementations for each component of the model. Sitecore supplies default ASP.NET security

www.it-ebooks.info

providers for authentication, user profile management, and role management. Sitecore provides optional modules that enable you to use Microsoft Dynamics Customer Relationship Management (CRM) and Active Directory (AD) for authentication. You can replace the default providers with these or with custom provider implementations that access external systems. Alternatively, you can implement virtual users, in which case you first invoke APIs (Application Programmer's Interfaces) to authenticate against the external system, and then invoke APIs to build a virtual user and associate it with roles in Sitecore.

For additional information about Sitecore security, see The Sitecore Security Reference (`http://bit.ly/t2AoRF`), The Sitecore Security Administrator's Cookbook (`http://bit.ly/rtN2ow`), The Sitecore Security API Cookbook (`http://bit.ly/tRRxiU`), and the Security section (`http://bit.ly/t6cNzK`) on the Sitecore Developer Network (`http://sdn.sitecore.net`).



This book does not describe how to secure the ASP.NET infrastructure hosting your Sitecore solutions. For some helpful pointers in this regard, see The Sitecore Security Hardening Guide (`http://bit.ly/swFfRp`).

ACCESS RIGHTS

Access rights control which users can access features and functions of the CMS environment and on the managed websites. You can apply access rights for users or roles. To simplify maintenance, you should apply access rights for roles, unless that would require you to create numerous roles containing individual users to achieve the same end.

Sitecore stores access rule definitions in each item using the `__Security` (`Sitecore.FieldIDs.Security`) field defined by the standard template. To get an idea of how Sitecore serializes access rights, apply access rights to an item, view the standard fields, view raw values, and investigate the value of this field (titled Security) in the Security section. To see rights that you apply in the content management appear in the content delivery environment, publish the items containing updated security descriptors.



If you set the `embedAclInItems` attribute of the `/configuration/sitecore/authorization/providers/add` element in the Web.config file to `false`, Sitecore stores access rights in the `AccessControl` table of the database specified by the `connectionStringName` attribute of that element. If this attribute in both content management and content delivery environments specifies the same database, then you do not need to publish updates to access rules, and changes take effect immediately. If these attributes specify separate databases, you must implement a strategy to replicate changes from the database used by the content management environment to the database(s) used by the content delivery environment.

www.it-ebooks.info

Each `/configuration/sitecore/accessRights/rights/add` element in the Web.config file defines an access right. The default access rights include the following:

- `field:read` — Apply the `field:read` access right in data template field definition items to control which users can read values from the field. You can use the `field:read` access right to hide fields in user interfaces such as the Content Editor.
- `field:write` — Use the `field:write` access right to make fields read-only in user interfaces such as the Content Editor, preventing CMS users from updating the contents of a field.
- `item:read` — Use the `item:read` access right to control whether users can access items. If a user cannot access an item (or, assuming security inheritance, its descendants), then that user cannot perform other operations on that item and its descendants either, except in certain cases, such as when deleting an item containing hidden descendants.



While all access rights can affect both CMS users and visitors to the managed web sites, most access rights apply primarily in the content management environment. The `item:read` *access right is an exception, and is particularly important in the content delivery environment, where it controls which items visitors can access.*

- `item:write` — Use the `item:write` access right to control which CMS users can update existing items.
- `item:rename` — Use the `item:rename` access right to control which CMS users can rename existing items.
- `item:create` — Use the `item:create` access right to control which CMS users can use insert options to create items beneath existing items.
- `item:delete` — Use the `item:delete` access right to control which users can delete an item and its descendants.
- `item:admin` — Use the `item:admin` access right to control which CMS users can update security access rules for an item and its descendants.
- `language:read` — Apply the `language:read` access right to language definition items to control whether accounts can read items in those languages.
- `language:write` — Apply the `language:write` access right to language definition items to control whether accounts can update items in those languages.
- `site:enter` — Apply the `site:enter` access right to the home items of each managed site to control access to the entire site.
- `insert:show` — Apply the `insert:show` access right to definition items for data templates, command templates, and branch templates to control which CMS users can use these items to insert additional items.

www.it-ebooks.info

- `workflowState:delete` — Apply the `workflowState:delete` access right to definition items for workflow states to control whether CMS users can delete items with versions in those workflow states.
- `workflowState:write` — Apply the `workflowState:write` access right to definition items for workflow states to control whether CMS users can write to items in those workflow states.
- `workflowCommand:execute` — Apply the `workflowCommand:execute` access right to definition items for workflow commands to control whether CMS users can see and invoke those commands.



Some access rights depend on others. For example, the `field:write` *access right is irrelevant when an account does not have* `field:read` *access to that field. Furthermore, a user cannot update a field in an item to which the user does not have the* `item:read` *and* `item:write` *access rights. The access rights that affect workflow features further depend on the user's access to the items associated with those workflow constructs; for example, if a user does not have read access to an item, they cannot move the item from one workflow state to another, regardless of the their* `workflowState:write` *access rights.*

The `Sitecore.Security.AccessControl.AccessRight` class represents an access right. You can use the `IsAllowed()` method of the `Sitecore.Security.AccessControl.AuthorizationManager` class to determine whether a user has an access right to an item. As an example, you could use the following code as a starting point to determine whether the context user has write access to the context item (for more information about the context user, see the Context User section later in this chapter):




```
Sitecore.Security.Accounts.User user = Sitecore.Context.User;
Sitecore.Diagnostics.Assert.IsNotNull(user, "context user");
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
Sitecore.Diagnostics.Assert.IsNotNull(item, "context item");
if (Sitecore.Security.AccessControl.AuthorizationManager.IsAllowed(
  item,
  AccessRight.ItemWrite,
  Context.User))
{
  // the context user has write access to the context item
}
```

CheckWriteAccess.cs

www.it-ebooks.info



You can add custom access rights to use for your own purposes. For an example that describes how you can implement a custom access right to control whether a user can publish each individual item, see my blog post at `http://bit.ly/q9FQvl`.

Inheriting Access Rights

By default, each item automatically inherits access rights from its parent. If you do not specify access rights for an item, it inherits access rights from its parent; the access rights defined for the parent apply to the item. If you do not specify access rights for the parent item, then access rights defined for the parent item of that parent (the grandparent) apply to that item (the grandchild), and so on to the top of the content tree. For a more detailed explanation of how Sitecore applies inherited access rights, see the next section, "Evaluating Access Rights."

As described in subsequent sections, you can use the Security Editor or the Content Editor to disable security inheritance for any item, meaning that the item no longer inherits access rights from its parent. For instructions to grant and deny rights, and to enable and disable inheritance, see the section "Setting Access Rights." When applying inherited access rights for the item or any of its descendants, after processing an item that disables security inheritance to its descendants, Sitecore stops evaluating access rights. If the item and its inheritance chain do not grant an access right, Sitecore denies that right by default. In addition to disabling access right inheritance completely for an item, you can also disable inheritance for individual access rights for individual users and roles.

If the inheritance chain to the root item or to an item that disables security inheritance does not allow an access right for a user or any of that user's roles, Sitecore denies that right by default. After you break security inheritance for an item, unless you have granted access rights to specific roles or users, only administrators can access that item until you grant access rights to additional roles or users.

Evaluating Access Rights

Figure 4-1 and the list that follows approximate the process that Sitecore follows to determine whether to grant or deny an access right to a user.

- When the user is a Sitecore administrator, the system always allows the access right.
- When the item explicitly denies the access right to the user, the system denies the access right.
- When the item explicitly allows the access right to the user, the system allows the access right.

www.it-ebooks.info

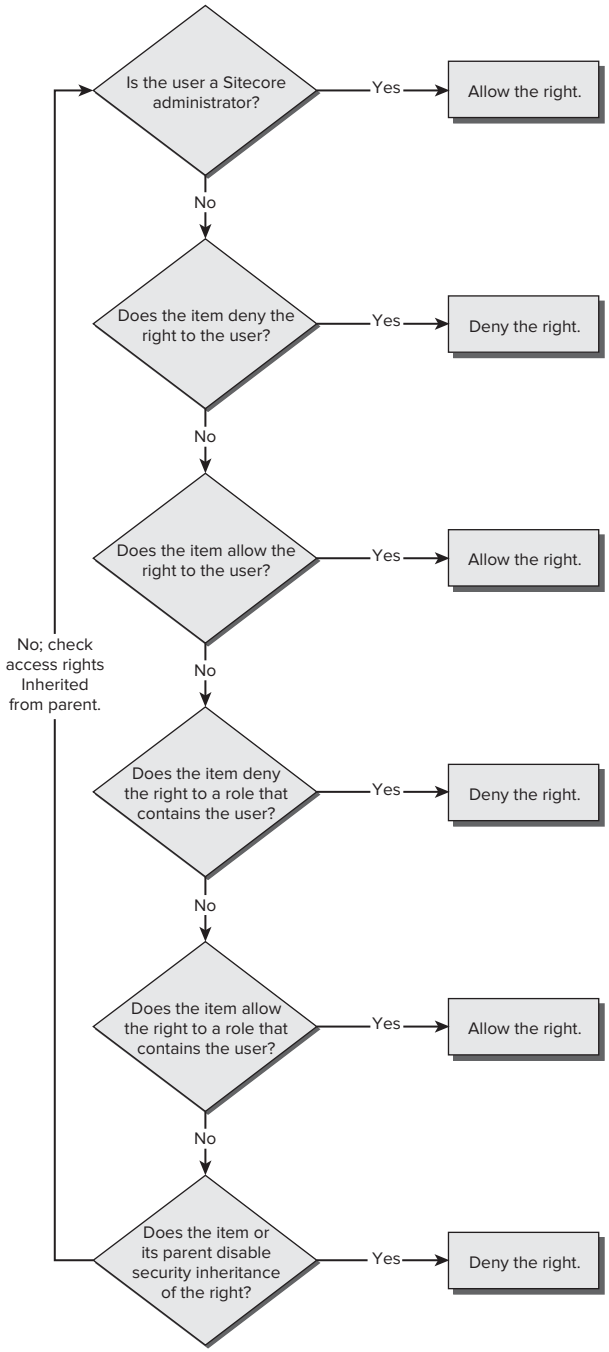


FIGURE 4-1

www.it-ebooks.info



Access rights assigned to a user override rights assigned to their roles. If you allow or deny a right to a user, you allow or deny that right for that user regardless of whether you allow or deny that right to any of their roles.

- When the item explicitly denies the access right to any of the roles that contain the user, the system denies the access right.
- When the item explicitly allows the access right to any of the roles that contain the user, the system allows the access right.
- When the item explicitly disables security inheritance for the access right, the system denies the access right.
- If none of the previous conditions result in allowance or denial of the access right, and the item does not disable inheritance of that access right from its parent item, Sitecore evaluates the same criteria sequentially on the ancestors of the item, starting with the parent of the item and progressing towards the top of the tree. Sitecore stops evaluating access right inheritance at the nearest ancestor that disables security inheritance, respecting options for disabling inheritance for individual users, roles, access rights, and the entire item.

As you can see, a user's explicit rights override the rights provided by a user's roles, and Sitecore stops evaluating security inheritance as soon as it finds a reason to allow or deny the right.

The Sitecore security infrastructure implicitly and automatically affects almost all operations on Sitecore items. If a presentation component attempts to access an item to which the context user does not have read access, Sitecore behaves as if that item does not exist. If the user somehow requests the URL of a restricted item, Sitecore redirects the browser to the URL of the friendly error page specified by the `NoAccessUrl` setting in the Web.config file.

By default, Sitecore applies the access rights of the context user. Most importantly, the read access right controls whether the user can access an item in any way. If the context user does not have read access to the item specified by the first argument to the `GetItem()` method of the `Sitecore.Data.Database` class, that method returns `null` rather than a `readSitecore.Data.Items.Item` object. Other item access techniques return `null` or exclude items from lists.

Ensure that each `Sitecore.Data.Items.Item` is not `null` and contains at least one version in the context language (by checking the `Count` property of the `Versions` property of the item) before using that item in your code, such as when you retrieve items specified in the field of another item or from a search index. For example, given the ID of the item in the query string parameter named `sc_item`, the following code confirms that the context user can read that item from the context database:

www.it-ebooks.info




```
string id = Sitecore.Web.WebUtil.GetQueryString("sc_item");

if (!String.IsNullOrEmpty(id))
{
  Sitecore.Data.Database db = Sitecore.Context.Database;
  Sitecore.Diagnostics.Assert.IsNotNull(
    db,
    "db");
  Sitecore.Data.Items.Item item = Sitecore.Context.Database.GetItem(id);

  if (item != null && item.Versions.Count > 0)
  {
    // the item exists, the context user can read that item,
    // and that item contains at least one version in the context language
  }
}
```

ConfirmItemNotNull.cs

If a user cannot access an item, that user cannot perform other operations on it, such as updating or deleting it, except in certain cases, such as when a user deletes an item, which deletes descendants of that item to which that user may not have rights to delete.

Setting Access Rights

You can use the browser-based Security Editor application to apply access rights to items for users and roles. To access the Security Editor, log in to the Sitecore desktop, click the Sitecore button, click Security Tools, and then click Security Editor. Figure 4-2 shows the Security Editor.

Alternatively, you can also use the Content Editor to apply access rights. To do so, click the Security tab in the Content Editor, and then click Assign in the Security group. Because its primary selection is an item, the Content Editor is most efficient when you need to set access rights for an individual item or a small number of items, especially when you need to apply access rights for a number of users or roles. The Security Editor is more efficient when you need to apply access rights for a number of items or a branch, especially when you repeatedly apply access rights for an individual account.

When you need to apply a consistent pattern of access rights to more than one item, first consider whether you can instead use security inheritance. Otherwise, consider scripting the application and (potentially) the maintenance of access rights, including updates and publishing. If you must apply the same access rights to a number of items through the user interface, consider using the Content Editor to copy and paste the raw value of the `__Security` (`Sitecore.FieldIDs.Security`) field defined in the standard template, instead of using the Security Editor to apply the same access rights repeatedly. If you choose to use the Security Editor, minimize the number of times you must select a account by setting access rights for

www.it-ebooks.info

each account on a number of items, and then moving on to the next account, rather than setting access rights item by item.

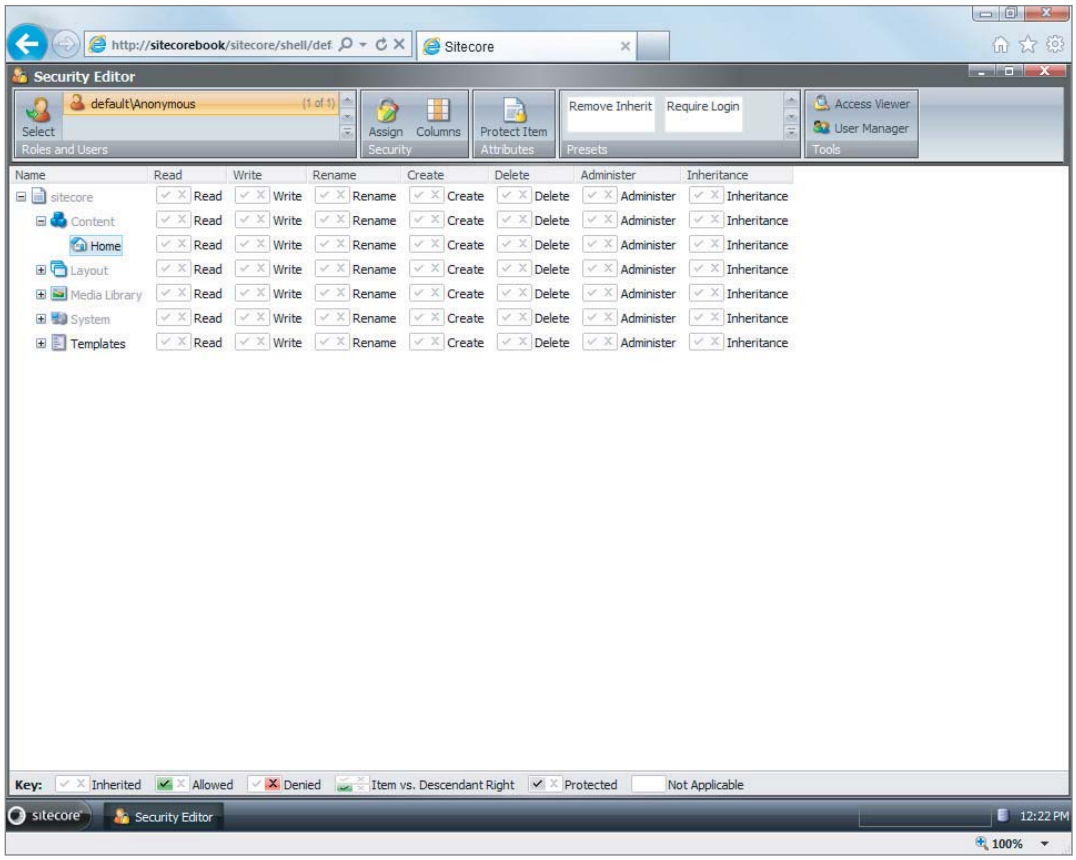


FIGURE 4-2

Interpreting Access Rights with the Access Viewer

You can use the browser-based Access Viewer application to review effective access rights for a user. To access the Security Editor, log in to the Sitecore desktop, click the Sitecore button, click Security Tools, and then click Access Viewer. Figure 4-3 shows the Access Viewer.

Use the following steps to select a user or role for which to evaluate access rights in Access Viewer:

1. Click Select in the Users group on the Ribbon in the Access Viewer. The Select an Account dialog appears as shown in Figure 4-4.
2. Select Roles or Users from the radio group at the top.

www.it-ebooks.info

3. Select the account from the list at the bottom, and then click OK. The Select an Account dialog closes and the Access Viewer refreshes to show effective access rights for the selected account.

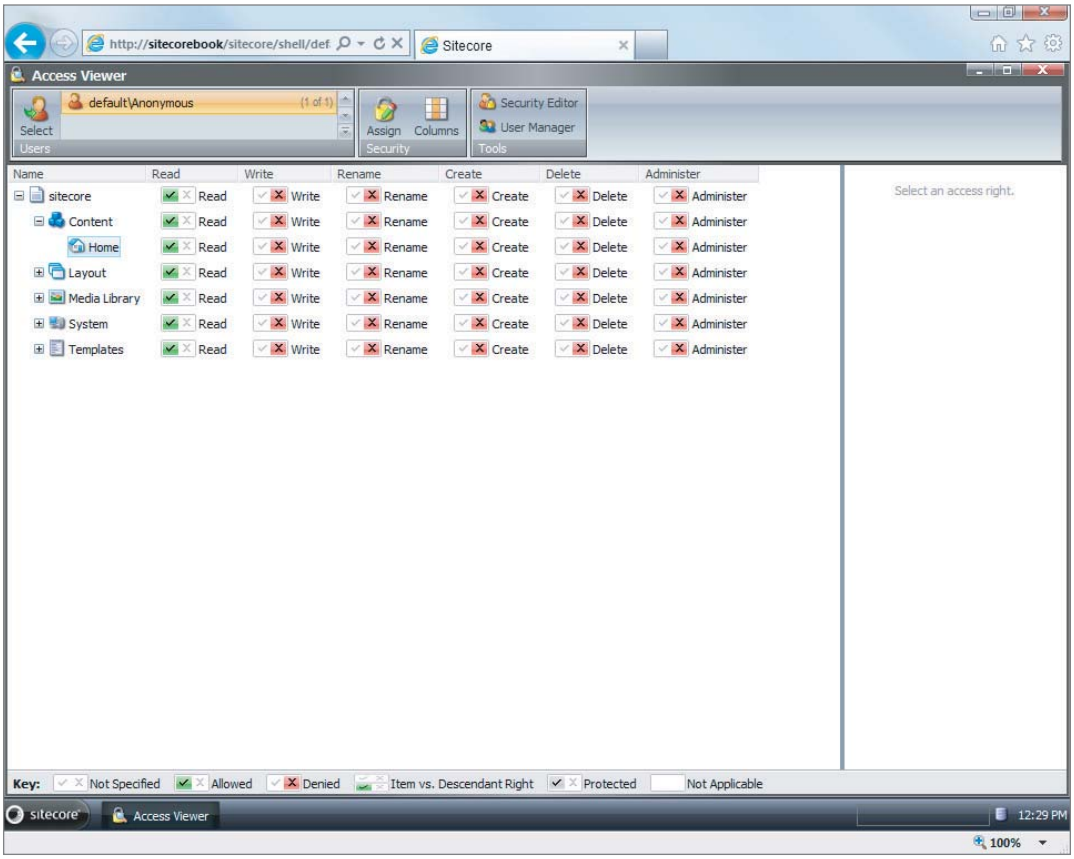


FIGURE 4-3

The Access Viewer evaluates effective access rights for the account you select, such as inheritance and role membership, including nested roles (as described in a subsequent section of this chapter), and displays the results. Green indicates that an item allows access for the selected account; red indicates that the system denies that right to that account. If you click an access right, the Access Viewer shows a description of that right, and an explanation of why the system allows or denies that right to the selected account.

You cannot use the Access Viewer to update access rights for an item; you can use the Access Viewer only to see how access rights apply. To update access rights for an item, use the Security Editor or the Content Editor as explained in the previous section, "Setting Access Rights."

www.it-ebooks.info

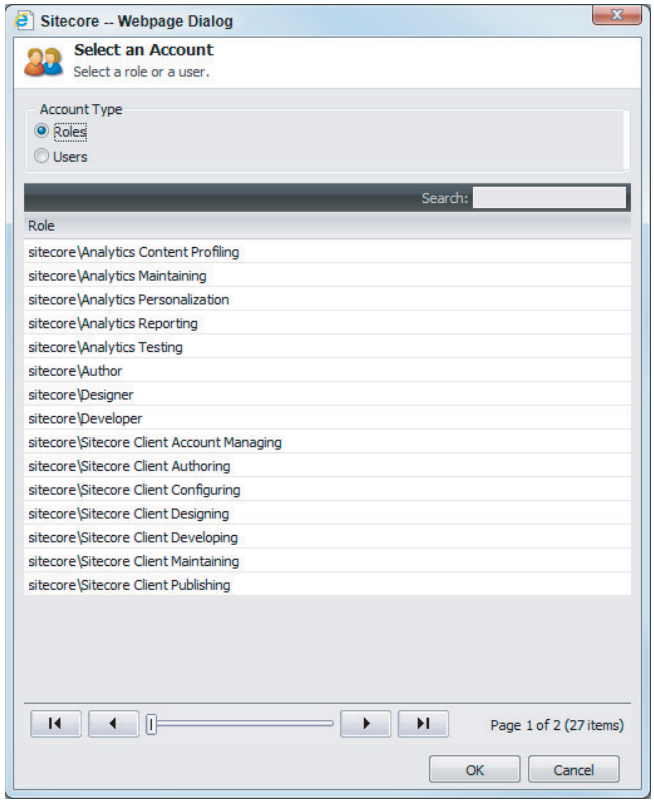


FIGURE 4-4

Securing Data Template Fields

Sometimes you need to control which fields CMS users can read and write within items to which those users have read and write access. To provide this level of control, you can apply the `field:read` and `field:write` access rights to field definition items in data templates. If you do not apply access rights for a data template field, the user's access rights for the item apply to that field.



Do not confuse the `field:read` *access right with the* `item:read` *access right or the* `field:write` *access right with the* `item:write` *access right. Field access rights defined in definition items for data template fields control which users can read and write values in those fields of items to which they also have the corresponding item access rights. You would not normally define item access rights in definition items for individual data template fields, where they control which users can update those definition items as opposed to updating fields in items based on those data templates.*

www.it-ebooks.info

To apply access rights to a data template field, do the following:

1. Select a definition item for a data template field in the Content Editor or the Template Manager.
2. Select the Security tab and then click the Assign command in the Security group. The Security Settings dialog appears as shown in Figure 4-5.

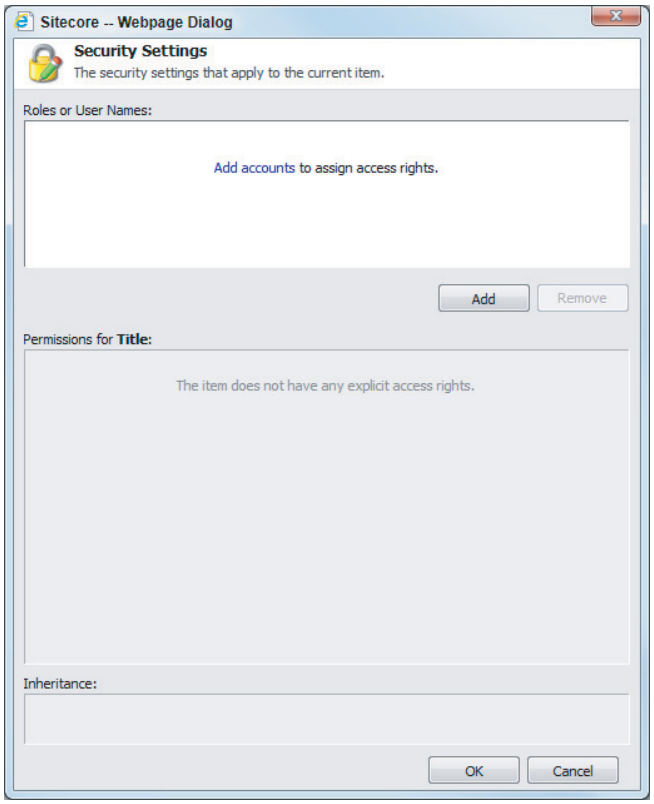


FIGURE 4-5

3. Click Add accounts. The Select an Account dialog appears as shown previously in Figure 4-4.
4. Select a role, and then click OK. Alternatively, you can select a user if you absolutely must apply security for a user rather than a role.
5. Grant or deny the Field Read (`field:read`) and Field Write (`field:write`) access rights in the definition item for the data template field to the role as shown in Figure 4-6, which demonstrates granting Field Read and Field Write to the Designer role in the Sitecore security domain. (For more information about security domains, see the "Security Domains" section later in this chapter.) Then click OK. You see the Content Editor.

www.it-ebooks.info

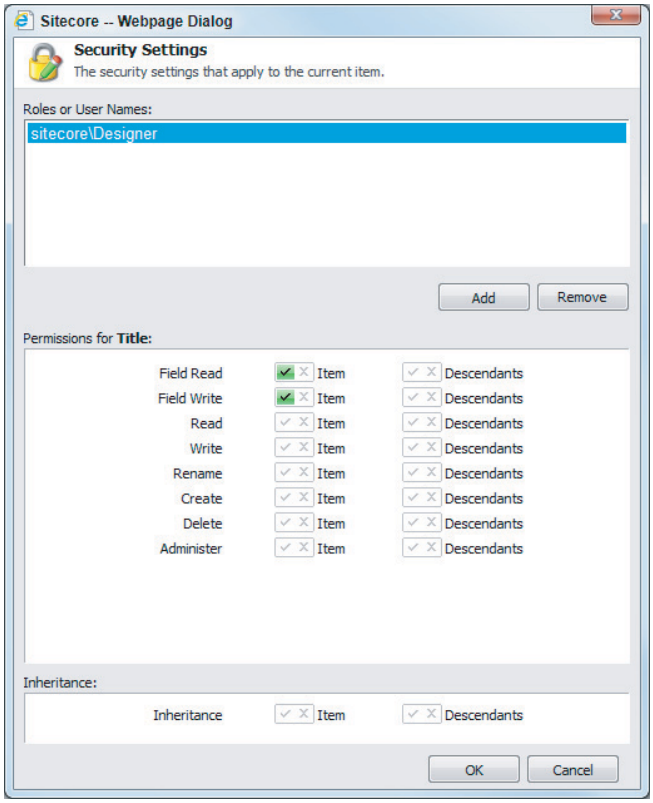


FIGURE 4-6



You should grant rights to read fields to any account to which you grant rights to write those fields. You can grant read access to everyone, and grant write access to specific roles and even users.

Alternatively, in the Security Editor, do the following:

1. Click the Columns command in the Security group on the Ribbon in the Security Editor. The Columns dialog appears as shown in Figure 4-7.
2. Select Field Read and Field Write, and click OK. You see the Security Editor.
3. Select the definition item for the data template field in the content tree of the Security Editor as shown in Figure 4-8.
4. Click the Select command in the Roles and Users group on the Ribbon in the Security Editor to select a role or a user.
5. Use the Security Editor to grant or deny the Field Read and Field Write access rights.

www.it-ebooks.info

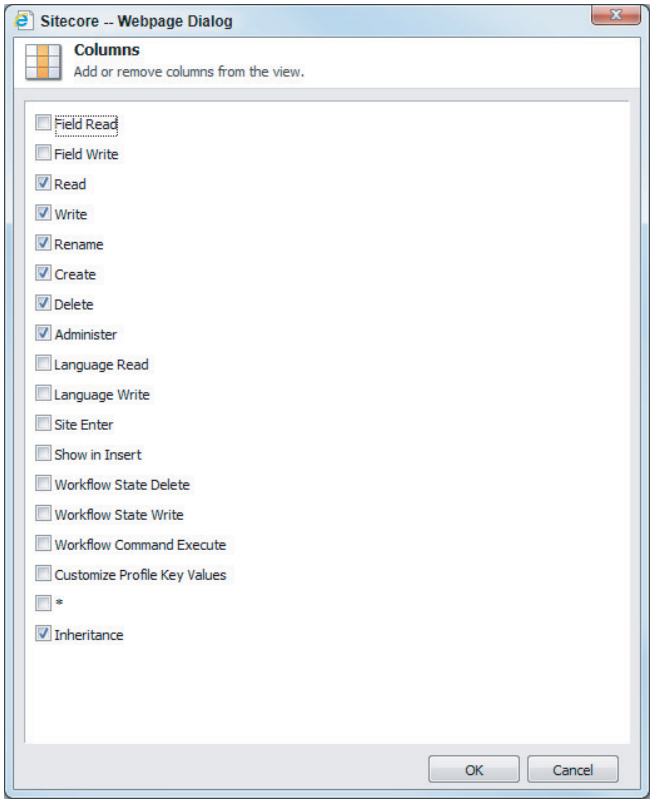


FIGURE 4-7

CMS user interfaces such as the Content Editor apply the `field:read` and `field:write` access rights to items that include such data template fields. To support data template field access rights in user interfaces such as the Page Editor, your presentation components must respect data template field access rights. You can use the `CanRead` and `CanWrite` properties of the `Sitecore.Data.Fields.Field` class to determine whether a user can read and write to a field. The following code demonstrates how to confirm that the context user has read and write access to the Title field in the context item, as well as read and write access to the context item itself.




```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
Sitecore.Diagnostics.Assert.IsNotNull(item, "item");

if (Sitecore.Security.AccessControl.AuthorizationManager.IsAllowed(
  item,
  Sitecore.Security.AccessControl.AccessRight.ItemWrite,
  Sitecore.Context.User))
{
  Sitecore.Data.Fields.Field title = item.Fields["title"];
  Sitecore.Diagnostics.Assert.IsNotNull(title, "title");

  if (title != null
    && title.CanRead
```

www.it-ebooks.info

```
        && title.CanWrite)
    {
      // the context user has write access to the field
    }
  }
```

CanWriteTitle.cs

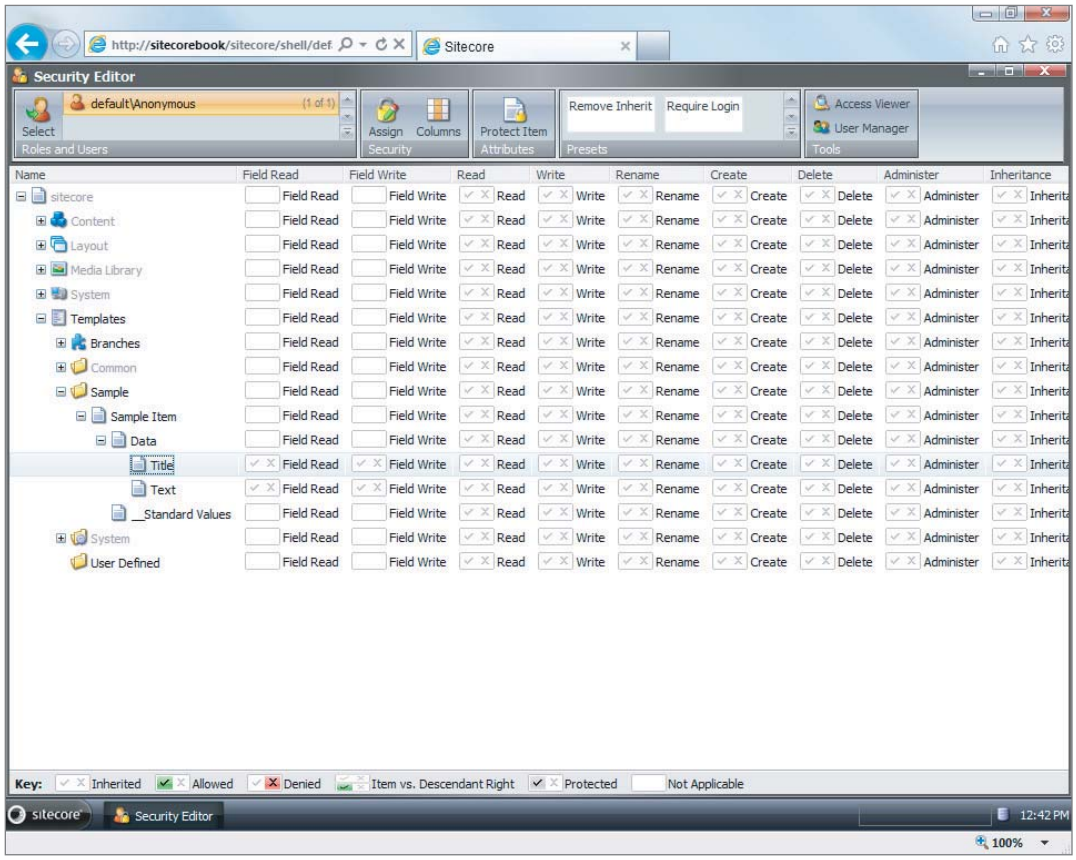


FIGURE 4-8

Applying Access Rights with APIs

You can use APIs to set access rights for an item. For example, you cannot set the `site:enter` access right through any Sitecore user interface. To set this access right programmatically, add an access rule to the list of access rules associated with the item.

The `Sitecore.Security.AccessControl.AccessRight` class represents an access right. The `Sitecore.Security.AccessControl.AccessRule` class represents an access rule, which can associate an account with the allowance or denial of an access right. The `Sitecore.Security`

www.it-ebooks.info

`.AccessControl.AccessRuleCollection` class represents a list of access rules, such as the access rules associated with an item. The `GetAccessRules()` method of the `Sitecore.Security.AccessControl.ItemSecurity` class exposed by the `Security` property of the `Sitecore.Data.Items.Item` class returns an `Sitecore.Security.AccessControl.AccessRuleCollection` object that represents the access rules associated with the item. You can use the `SetAccessRules()` method of the `Sitecore.Security.AccessControl.ItemSecurity` class to update the list of access rules associated with an item.

The example shown in Listing 4-1 denies the `site:enter` access right in the `/sitecore/content/home` item to the Anonymous user in the Extranet security domain.



LISTING 4-1: Denying the site:enter access right

```
namespace SitecoreBook.Web.UI.WebControls
{
  using System.Web.UI;

  public class DenySiteAccessToExtranetAnonymous : Sitecore.Web.UI.WebControl
  {
    // set an access right for an account in an item
    protected void SetAccessRight(Sitecore.Data.Items.Item item,
      Sitecore.Security.Accounts.Account account,
      Sitecore.Security.AccessControl.AccessRight right,
      bool allow)
    {
      Sitecore.Diagnostics.Assert.IsNotNull(item, "item");
      Sitecore.Diagnostics.Assert.IsNotNull(account, "account");

      // the new list of access rules to apply to the item
      Sitecore.Security.AccessControl.AccessRuleCollection updated = new
        Sitecore.Security.AccessControl.AccessRuleCollection();

      foreach (Sitecore.Security.AccessControl.AccessRule existingRule
        in item.Security.GetAccessRules())
      {
        if (existingRule.Account.Name != account.Name
          || existingRule.Account.AccountType != account.AccountType
          || existingRule.AccessRight != right)
        {
          updated.Add(existingRule);
        }
      }

      // indicates whether to allow or deny the access right
      Sitecore.Security.AccessControl.AccessPermission permission =
        allow ? Sitecore.Security.AccessControl.AccessPermission.Allow :
        Sitecore.Security.AccessControl.AccessPermission.Deny;

      // add an access right according to the parameters passed to the method
      updated.Add(Sitecore.Security.AccessControl.AccessRule.Create(
        account,
```

www.it-ebooks.info

```
            right,
            Sitecore.Security.AccessControl.PropagationType.Any,
            permission));

          // update access rules for the item
          item.Security.SetAccessRules(updated);
        }

        protected override void DoRender(HtmlTextWriter output)
        {
          // explicitly reference the Master database
          Sitecore.Data.Database master =
            Sitecore.Configuration.Factory.GetDatabase("master");
          Sitecore.Diagnostics.Assert.IsNotNull(master, "master");

          // retrieve the home item by path
          Sitecore.Data.Items.Item home = master.GetItem("/sitecore/content/home");
          Sitecore.Diagnostics.Assert.IsNotNull(home, "home");

          // retrieve the extranet security domain from the configuration factory
          Sitecore.Security.Domains.Domain extranet =
            Sitecore.Configuration.Factory.GetDomain("extranet");
          Sitecore.Diagnostics.Assert.IsNotNull(extranet, "extranet");

          // the Anonymous user in the extranet domain
          Sitecore.Security.Accounts.User anonymous = extranet.GetAnonymousUser();
          Sitecore.Diagnostics.Assert.IsNotNull(anonymous, "anonymous");

          // apply the site:enter access right for the extranet\anonymous user
          // for the /sitecore/content/home item in the Master database
          this.SetAccessRight(
            home,
            anonymous,
            Sitecore.Security.AccessControl.AccessRight.SiteEnter,
            false /*allow*/ );
        }
      }
    }
```

DenySiteAccessToExtranetAnonymous.cs

The `SetAccessRight()` method in this example contains code to update the access rules for an item to grant or deny a specific right to an account. This method accepts four arguments:

1. The item for which to set the access right
2. The account for which to set the access right
3. The access right to set
4. A Boolean indicating whether to grant (`true`) or deny (`false`) the access right

This method retrieves the existing access rights for the item into a list using the `Sitecore.Security.AccessControl.AccessRuleCollection` class, excluding any existing access rights

www.it-ebooks.info

that exist for that account in order to replace any such rule. It then adds a `Sitecore.Security.AccessControl.AccessRule` object to that collection to grant or deny the right to the user as specified by the arguments to the method. Finally, it uses the `SetAccessRules()` method of the `Security` property (which uses the `Sitecore.Security.AccessControl.ItemSecurity` class) of the `Sitecore.Data.Items.Item` passed to the method to update the access rules associated with the item to those listed in the collection.

The `DoRender()` method in this example constructs arguments to pass to the `SetAccessRight()` method described previously. The `DoRender()` method uses the `GetDatabase()` method of the static `Sitecore.Configuration.Factory` class to retrieve a `Sitecore.Data.Database` object that represents the Master database, and then uses the `GetItem()` method of that object to retrieve the `/sitecore/content/home` item from that database. Next, it uses the `GetDomain()` method of the same `Sitecore.Configuration.Factory` class to retrieve an instance of the `Sitecore.Domains.Domain.Domain` object that represents the Extranet security domain. It then uses the `GetAnonymousUser()` method of that `Sitecore.Domains.Domain.Domain` object to retrieve the Anonymous user in that domain. Finally, it calls the `SetAccessRight()` method, passing arguments to add or update an access rule denying the `site:enter` access right to that Anonymous user in that domain for that item.



I implemented this as a web control for my own convenience, but you can call equivalent methods from other types of components.

SECURITY ACCOUNTS AND DOMAINS

Because they serve similar purposes and support similar operations, ASP.NET refers to both users and roles using the term *security accounts*, which includes both users and roles. In Sitecore, a *security domain* is a collection of security accounts.



In some contexts, the terms role *and* group *are largely interchangeable. Groups often belong to operating systems, while roles often belong to applications. In Sitecore a* group *is a collection of commands in the Ribbon of a user interface such as the Content Editor. Excluding the fact that you can apply access rights to group definition items to control who can access those groups of commands in user interfaces, in Sitecore, groups have nothing to do with security.*

Users

Users are individual people with authentication credentials for a Sitecore solution, whether in the CMS or on one or more of the published websites. A user can be a member of any number of security roles.

You can use the `Sitecore.Security.Accounts.User` class to access a user. You can use the static `Exists()` method of this class to determine whether a user exists in a domain. For example, the

www.it-ebooks.info

following code shows how to determine if the default user named admin exists in the Sitecore security domain:

```
if (Sitecore.Security.Accounts.User.Exists("sitecore\\admin"))
{
  Sitecore.Diagnostics.Log.Error(
    "grant admin to a real user and remove the default Sitecore\\admin user.",
    this);
}
```

You can use the static `Sitecore.Security.Accounts.User.FromName()` method to retrieve a user by name. The first argument to this method specifies the user name, including the name of the security domain that contains the user. The second parameter tells the system whether to update its internal record of when the user last accessed the solution. For example, the following code shows how to determine whether the default admin user exists in the Sitecore security domain:

```
Sitecore.Security.Accounts.User admin =
  Sitecore.Security.Accounts.User.FromName("sitecore\\admin", false);
Sitecore.Diagnostics.Assert.IsNull(admin, "sitecore\\admin exists");
```

The `System.Web.Security.MembershipUser` class provides an alternative means of accessing users that you can use if the `Sitecore.Security.Accounts.User` class does not provide functionality that you require. Important methods in the `System.Web.Security.MembershipUser` class include:

- `ChangePassword()` — Change the password associated with the user to a new value (requires the existing password).
- `ChangePasswordQuestionAndAnswer()` — Change the security question and answer associated with the user to new values (requires the existing password).
- `ResetPassword()` — Reset the password associated with the user to a random string (requires the answer to the security question associated with the user).
- `UnlockUser()` — Unlock the user after that user has reached the limit for invalid login attempts.

For example, you can pass the name of a user (including the security domain) to the static `GetUser()` method of the `System.Web.Security.Membership` class to retrieve that user as a `System.Web.Security.Membership` object. The following example accesses the context user as an instance of that class:

```
Sitecore.Security.Accounts.User user = Sitecore.Context.User;
Sitecore.Diagnostics.Assert.IsNotNull(user, "context user");
System.Web.Security.MembershipUser membershipUser =
  System.Web.Security.Membership.GetUser(user.Name);
```

Conversely, you can use the static `FromName()` method of the `Sitecore.Security.Accounts.User` class to retrieve a `Sitecore.Security.Accounts.User` object based on the `UserName` property of a `System.Web.Security.MembershipUser` object. Sitecore does not provide a method to access all known users on the system. You can use the static `GetAllUsers()` method of the `System.Web.Security.Membership` method to iterate all users using `System.Web.Security.MembershipUser`

www.it-ebooks.info

objects, and convert each to a `Sitecore.Security.Accounts.Account` object only if needed, as in the following example:

```
foreach(System.Web.Security.MembershipUser membershipUser
  in System.Web.Security.Membership.GetAllUsers())
{
  Sitecore.Security.Accounts.User user =
    Sitecore.Security.Accounts.User.FromName(membershipUser.UserName, false);
  // process user
}
```

Alternatively, the `GetUsers()` method of the `Sitecore.Security.Domains.Domain` class returns all of the users in that security domain. You can use the `GetDomainNames()` method of the static `Sitecore.Configuration.Factory` class to retrieve the names of all security domains, and the `GetDomain()` method of that class to retrieve a security domain by name. The following example web control demonstrates how to use these methods to iterate all users in all domains, which is functionally equivalent to using the `GetAllUsers()` method of the `System.Web.Security.Membership` class. You can determine which approach performs better for your load of users and purpose, including sorting, filtering, and any additional requirements. The `GetUsers()` method of the `Sitecore.Security.Domains.Domain` class may be more efficient than the `GetAllUsers()` method of the `System.Web.Security.Membership` class when you need to iterate the users in a single Sitecore security domain.




```
namespace SitecoreBook.Web.UI.WebControls
{
  public class ListDomainsAndUsers : Sitecore.Web.UI.WebControl
  {
    protected override void DoRender(System.Web.UI.HtmlTextWriter output)
    {
      // iterate the names of all Sitecore security domains
      foreach(string domainName
        in Sitecore.Configuration.Factory.GetDomainNames())
      {
        // retrieve the domain object
        Sitecore.Security.Domains.Domain domain =
          Sitecore.Configuration.Factory.GetDomain(domainName);

        // ensure the domain exists
        Sitecore.Diagnostics.Assert.IsNotNull(domain, domainName);

        // list the name of the domain
        output.WriteLine("<strong>" + domain.Name + "</strong>");

        // if the domain contains at least one user
        if (domain.GetUserCount() > 0)
        {
          // write an unordered list
          output.WriteLine("<ul>");

          // iterate the users in that domain
          foreach (Sitecore.Security.Accounts.User user in domain.GetUsers())
          {
            // list the name of the user
            output.WriteLine("<li>" + user.Name + "</li>");
          }
```

www.it-ebooks.info

```
            output.WriteLine("</ul>");
          }
          else
          {
            output.WriteLine(" (no users)");
          }
        }
      }
    }
  }
```

ListDomainsAndUsers.cs

User Profiles

Each user has a profile, which can define any number of properties specific to that user, such as the user's full name and e-mail address. You can update user profiles in the User Manager. To access the User Manager, log in to the Sitecore desktop, click the Sitecore button, click Security Tools, and then click User Manager. Figure 4-9 shows the User Manager.

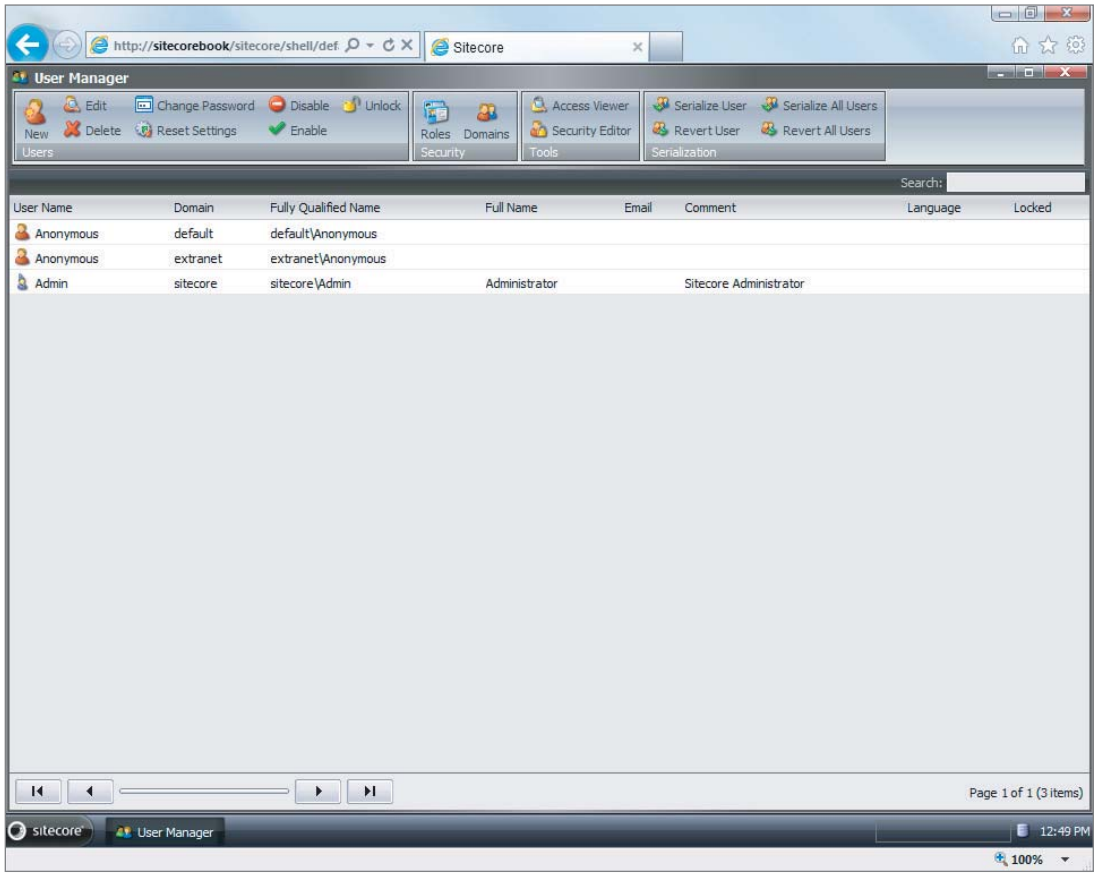


FIGURE 4-9

www.it-ebooks.info

To access the profile of a user, double-click that user in the User Manager. The Edit User dialog appears as shown in Figure 4-10.

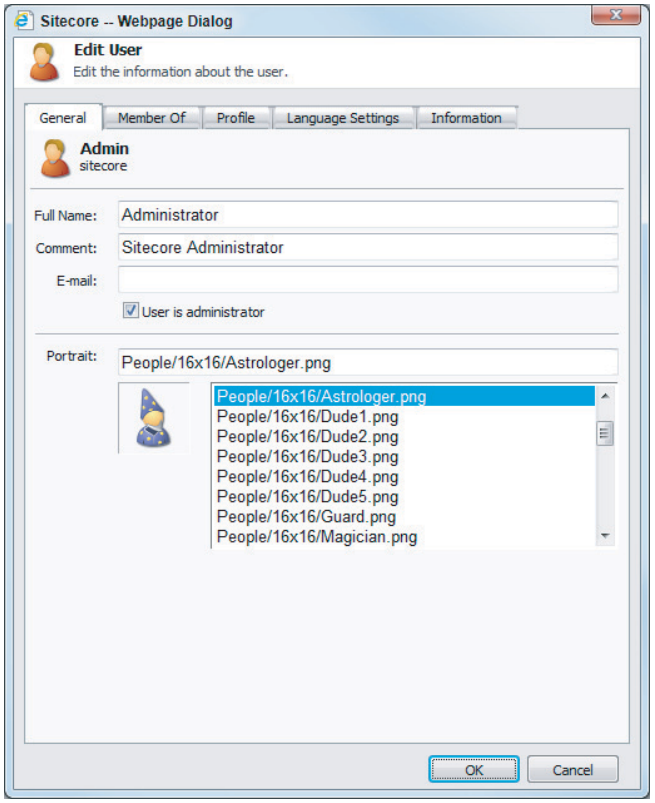


FIGURE 4-10

You can use the `Sitecore.Security.UserProfile` class to access user profiles. The `Profile` property of the `Sitecore.Security.Accounts.User` class conveniently exposes an instance of this type that represents the profile of the user.

To update user profile properties, the system must authenticate the user. To work with an authenticated user, you can either update the profile for the context user that has already logged in, or you can pass `true` as the second argument to the `Sitecore.Security.Accounts.User.FromName()` method when you retrieve a user explicitly.

The `Sitecore.Security.UserProfile` class exposes basic user profile properties such as the `Email` property for an email address. You can use the `GetCustomProperty()` and `SetCustomProperty()` methods of the `Sitecore.Security.UserProfile` class to get and set custom user profile properties. If you prefer, the `Sitecore.Security.UserProfile` class also exposes a collection that you can use to access profile values. The following example shows how to set custom properties using both approaches:

```
Sitecore.Security.Accounts.User user = Sitecore.Context.User;
```

www.it-ebooks.info

```
Sitecore.Diagnostics.Assert.IsNotNull(user, "context user");
Sitecore.Security.UserProfile profile = user.Profile;
Sitecore.Diagnostics.Assert.IsNotNull(profile, "profile");
profile.SetCustomProperty("First", "One");
profile["Second"] = "Two";
profile.Save();
```

The first four lines populate and validate the `Sitecore.Security.UserProfile` variable named `profile` with the profile of the context user. The next two lines demonstrate the two ways you can set custom profile properties by setting `profile` properties named `First` and `Second` for that user. The last line saves those changes to the user's profile.



To optimize performance, as shown in the example, invoke the `Save()` *method of the* `Sitecore.Security.UserProfile` *class after updating multiple profile properties rather than after setting each property.*

The `GetCustomPropertyNames()` method of the `Sitecore.Security.UserProfile` class returns the names of the custom profile properties defined for the user, but not the names of all profile properties defined for all users.

To expose custom profile properties in the User Manager, add fields to the `System/Security/User` data template in the Coredatabase using field names that correspond to the names of the user `profile` properties (for example, `First` and `Second` in the previous example). The User Manager application exposes such properties automatically on the Profile tab when you edit a user.



Be sure to make this change in the Core database rather than the Master database, or the fields will not appear in the User Manager.

The default `profile` properties exposed by the `System/Security/User` data template in the Core database consist of the following:

- **Portrait:** Appears with the username in the User Manager, on the Sitecore menu when that user logs in, and potentially elsewhere
- **Wallpaper:** Wallpaper image shown when the user logs in to the Sitecore desktop

The Security API Cookbook (`http://bit.ly/uWPDpK`) contains much more information about working with security profiles, including a sample user profile self-management form.

Anonymous Users

Each security domain can contain a user named Anonymous to represent all visitors who access the solution without providing authentication credentials. Sitecore ensures that the Anonymous user exists in all security domains that require such a user. You do not need to create or delete

www.it-ebooks.info

Anonymous users in any security domain. The Anonymous user in each security domain can have either no access or limited read access to the solution, but should not have read access to everything, and should have write access to nothing.

While Anonymous is a user, it actually represents a class of visitors who access the solution without authenticating. This class includes both actual visitors and robots such as search engines. For this reason, the Anonymous user is an exception to the rule that you should always apply access rights for roles rather than for users — the Anonymous user functions something like a role. One easy way to deny access to all users who have not authenticated is to deny access to the Anonymous user.

You can use the `GetAnonymousUser()` method of the `Sitecore.Security.Domains.Domain` class to access the Anonymous user for a security domain.



There is no Anonymous user in the Sitecore security domain, as all CMS users must authenticate to access the system.

Context User

The context user is the user associated with the current HTTP request. For each HTTP request, the `UserResolver` processor in the `httpRequestBegin` pipeline determines the context user. If the `UserResolver` processor cannot identify a known user, the context user is the Anonymous user in the security domain associated with the context site.

You can access the context user through the static `Sitecore.Context.User` property. For example, the following code shows how to check for the context user safely:

```
Sitecore.Diagnostics.Assert.IsNotNull(Sitecore.Context.User, "context user");
Sitecore.Security.Accounts.User = Sitecore.Context.User;
```

Roles

A *role* is a collection of users. Roles simplify the application of access rights by grouping users.

The `Sitecore.Security.Accounts.Role` class represents a security role. Use the following methods in the classes specified to perform essential tasks with this class:

- `Sitecore.Security.Accounts.Role.Exists()` — Determines if a role exists.
- `Sitecore.Security.Accounts.Role.FromName()` — Retrieves a role, much like the static `FromName()` method in the `Sitecore.Security.Accounts.User` class described in the previous section, but without the second argument that indicates whether to update the internal record indicating when the account last accessed the solution.
- `Sitecore.Security.Accounts.Role.IsMember()` — Determines whether a user is a member of a role. The first parameter specifies the account, which can be a `Sitecore.Security.Accounts.User` or `Sitecore.Security.Accounts.Role` class. The second parameter indicates whether to return `true` if `Sitecore.Security.Accounts.Role` is the global Everyone role or the Everyone role for the domain associated with the user. The third

www.it-ebooks.info

parameter indicates whether to apply nested roles as described in a subsequent section of this chapter.

- `System.Web.Security.Roles.CreateRole()` — Creates a role. Include the security domain name with the role name that you pass as the argument to this method.
- `System.Web.Security.Roles.DeleteRole()` — Deletes a role. Depending on the number of roles, the number of users, and whether you use nested roles as described in a subsequent section of this chapter, this operation can take some time.

To access the roles associated with a user, you can use the `Roles` property of the `Sitecore.Security.Accounts.User` class, which exposes the `Sitecore.Security.Accounts.UserRoles` class. The `Sitecore.Security.Accounts.UserRoles` class exposes the `Add()`, `Contains()`, `Remove()`, and `RemoveAll()` methods, which add a user to a role, determine whether a user's roles directly include another role (not respecting nested roles as described in a subsequent section of this chapter), remove a user from a role, and remove a user from all roles, respectively. The `Sitecore.Security.Accounts.User` class exposes the `IsInRole()` method to determine whether a user is a member of a role, respecting nested roles as described in a subsequent section of this chapter.

When you delete a role, Sitecore does not update access rules defined in items to reflect that deletion. Because the role does not exist, such access rules are irrelevant, and cannot apply. While this means that you can restore access rights by recreating the role, it continues to appear in user interfaces, which could confuse users. You are responsible for updating access rights in items that refer to deleted roles. The most common means of doing so are the following:

- A manual process in the Sitecore user interface
- A scheduled process to iterate all items and remove access rights assigned to roles that no longer exist
- An event handler, pipeline processor, or other solution to remove access rights when a user removes a role

The example scheduled agent shown in Listing 4-2 removes access rules associated with users and roles that do not exist for all items in the database(s) specified to the class as a configuration parameter.



LISTING 4-2: Using an agent, RemoveInvalidAccessRulesTask.cs, to fix access rules

```
namespace Sitecore.SitecoreBook.Tasks
{
  using System.Collections.Generic;

  public class RemoveInvalidAccessRulesTask
  {
    // constructor initializes the list of databases to process
    public RemoveInvalidAccessRulesTask()
    {
      this.Databases = new List<string>();
    }
```

continues

www.it-ebooks.info

LISTING 4-2 *(continued)*

```
// the databases to process, set in the Web.config file
public List<string> Databases
{
  get;
  set;
}

// for each database, remove invalid access rights, starting from the root item
public void Run()
{
  foreach (string dbName in this.Databases)
  {
    Sitecore.Data.Database database =
      Sitecore.Configuration.Factory.GetDatabase(dbName);
    Sitecore.Diagnostics.Assert.IsNotNull(database, "datbase " + dbName);
    Sitecore.Diagnostics.Log.Info(
      this + " : begin processing database " + dbName, this);
    this.RemoveInvalidAccessRights(database.GetRootItem());
    Sitecore.Diagnostics.Log.Info(
      this + " : completed processing database " + dbName, this);
  }
}

// remove invalid access rights from the item and its descendants
protected void RemoveInvalidAccessRights(Sitecore.Data.Items.Item item)
{
  Sitecore.Diagnostics.Assert.IsNotNull(item, "item");

  // new list of access rules to apply to item
  Sitecore.Security.AccessControl.AccessRuleCollection updated =
    new Sitecore.Security.AccessControl.AccessRuleCollection();

  // indicates that an access rule with an invalid account existed
  bool modified = false;

  // for each access rule associated with the item
  foreach (Sitecore.Security.AccessControl.AccessRule rule
    in item.Security.GetAccessRules())
  {
    // if the account is valid, add it to the new list of access rules
    if (this.UserExists(rule.Account) || this.RoleExists(rule.Account))
    {
      updated.Add(rule);
    }
    // otherwise, exclude it from the new list,
    // and set the flag to apply changes
    else
    {
      modified = true;
      string message = this
        + " : removing "
        + rule.Account.AccountType
```

www.it-ebooks.info

```
            + " "
            + rule.Account.Name
            + " from "
            + item.Paths.FullPath
            + " in " + item.Database.Name;
          Sitecore.Diagnostics.Log.Warn(message, this);
        }
      }

      if (modified)
      {
        item.Security.SetAccessRules(updated);
      }

      foreach (Sitecore.Data.Items.Item child in item.Children)
      {
        this.RemoveInvalidAccessRights(child);
      }
    }

    // returns true if the role exists or is the Everyone role
    protected bool RoleExists(Sitecore.Security.Accounts.Account account)
    {
      return account.AccountType == Sitecore.Security.Accounts.AccountType.Role
        && (Sitecore.Security.Accounts.Role.Exists(account.Name)
        || account.Domain.IsEveryoneRole(account.Name));
    }

    // returns true if the account represents a user that exists
    protected bool UserExists(Sitecore.Security.Accounts.Account account)
    {
      return account.AccountType == Sitecore.Security.Accounts.AccountType.User
        && Sitecore.Security.Accounts.User.Exists(account.Name);
    }
  }
}
```

RemoveInvalidAccessRulesTask.cs

In this agent:

- The `Databases` property specifies a list of databases from which to remove access rights that specify users and roles that do not exist.
- The Sitecore scheduling engine invokes the `Run()` method of the agent.
- The `Run()` method invokes the recursive `RemoveInvalidAccesssRules()` method for the root item in each database specified by the `Databases` property.
- The recursive `RemoveInvalidAccesssRules()` method sets the `modified` flag to indicate whether the item contains invalid access rules, creates the `updated` list to contain valid access rules for the item, and then iterates the access rules for the items adding valid rules to the `updated` list and setting the `modified` flag in the case of an invalid rule.

www.it-ebooks.info

- If the item contains invalid access rules, the agent logs a warning and updates the access rules for the item to those specified in the `updated` list.
- The `RemoveInvalidAccessRules()` method calls itself for each child of the item, recursively.

To schedule this agent to run approximately once daily, add a `/configuration/sitecore/scheduling/agent` element to the Web.config file based on the following example that will remove invalid access rules from the Master and Core databases:




```
<agent
  type="Sitecore.Sharedsource.Tasks.RemoveInvalidAccessRules,assembly"
  method="Run"
  interval="1.00:00:00">
  <databases hint="list">
    <database>master</database>
    <database>core</database>
  </databases>
</agent>
```

RemoveInvalidAccessRules.config



You do not need to include publishing target databases such as the Web in the list of databases. Updating the Master database and publishing those changes to the publishing target databases corrects this issue in those target databases.

For more information about scheduling tasks with Sitecore, see my blog post at `http://bit.ly/shQhEG`. For more information about passing parameters to types specified in the Web.config file, see my blog post at `http://bit.ly/A7Ae1s`.

Everyone Roles

The Everyone role automatically contains all users in a security domain, including the Anonymous user. Each security domain can contain an Everyone role. An additional Everyone role associated with no security domain contains all users in all security domains. You can use the Everyone role to allow an access right to all members of a domain.



You cannot use the Everyone role to deny an access right to Sitecore administrators. A user with the User Is Administrator check box selected on the General tab of the user's profile has all access rights to all items, regardless of the access rights defined for those items. Unless you implement custom code to block functions at runtime rather than using access rights, you cannot deny those functions to Sitecore administrators.

www.it-ebooks.info

You do not need to create or delete the Everyone role in any security domain. Technically, you can remove or rename the Everyone role in each security domain by updating the `/App_Config/Security/Domains.config` file according to the comments in that file, but I advise against making such changes without specific direction from Sitecore's customer service department.

The Everyone role is virtual, and does not exist in the role management provider. If you use the static `Exists()` method of the `Sitecore.Security.Accounts.Role` class to check whether the Everyone role exists, that method returns `false`. The list of roles associated with each user never includes the Everyone role. You can use the `GetEveryoneRole()` method of the `Sitecore.Security.Domains.Domain` class to access the Everyone role in a security domain.

Nested Roles

Sitecore supports nested roles, meaning that roles can be members of other roles, or that each role can contain other roles. Nested roles can simplify security configuration in some scenarios, such as when some roles are subsets of others. Figure 4-11 shows the nesting of roles within the Sitecore Client Users role in the Sitecore security domain.

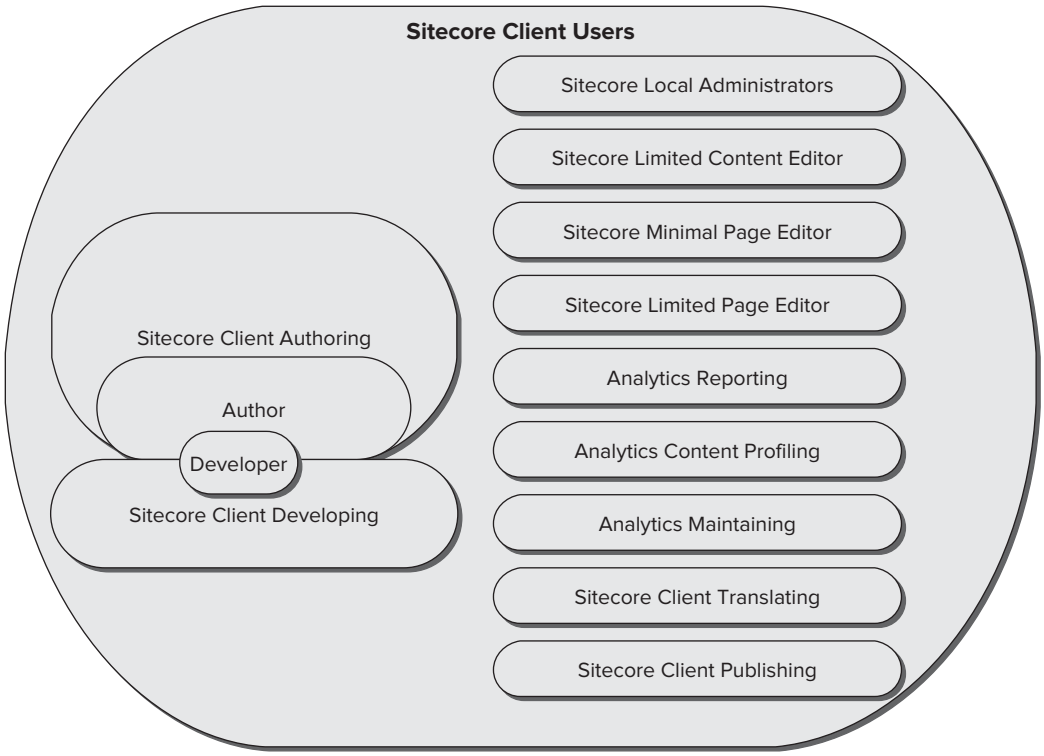


FIGURE 4-11

As the diagram depicts, the various roles listed on the right, from Sitecore Client Administrators through Sitecore Client Publishing, are all members of the Sitecore Client Users role, as are the

www.it-ebooks.info

Sitecore Client Authoring and the Sitecore Client Developing roles. Stated in reverse, the Sitecore Client Users role contains these roles. The Author role is a member of the Sitecore Client Authoring role, and hence an indirect member of the Sitecore Client Users role. The Developer role is a member of both the Author role and the Sitecore Client Developing role, and hence an indirect member of the Sitecore Client Users role.

You can use the static `Sitecore.Security.Accounts.RolesInRolesManager` class to configure nested roles. You can access the static `RolesInRolesSupported` property of this class to determine whether the solution supports nested roles. Relevant static methods in the `Sitecore.Security.Accounts.RolesInRolesManager` class are:

- `IsRoleInRole()` — Determines whether a role contains a nested role
- `AddRoleToRole()` — Adds a nested role to a role
- `RemoveRoleFromRole()` — Removes a nested role from a role
- `GetRolesInRole()` — Retrieves roles nested within the role. This method optionally includes roles nested within the requested roles, no matter how deep the nesting becomes.

Sitecore Client Roles

Sitecore predefines the Sitecore client roles to provide limited access to the Sitecore application, its user interfaces, and their features. For example, to grant a user access to publishing features in the Sitecore user interfaces, add that user to the Sitecore Client Publishing role in the Sitecore security domain.

For more information about the Sitecore client roles, see the Sitecore Client Configuration Cookbook (`http://bit.ly/qS8Dc1`) and the Sitecore Security Reference (`http://bit.ly/t2AoRF`).

Security Domains

Security domains segregate groups of security accounts, such as CMS users and users of the published websites. Each user and role exists within a domain.

Each `/domains/domain` element in the `/App_Config/Security/Domains.config` file configures a Sitecore security domain. You can update this file to add or remove security domains and to update properties of existing security domains.

Sitecore includes three default security domains:

- **Sitecore**: Contains the CMS accounts
- **Extranet**: Contains accounts for the published websites
- **Default**: Acts as a default security domain for accounts that do not belong to either of the other security domains

The `Sitecore.Security.Domains.Domain` class represents a Sitecore security domain. You can use the static `Sitecore.Context.Domain` property to access the context domain, which is the security domain associated with the context site. If the context site is `null`, or does not specify a security domain, the `Sitecore.Context.Domain` property returns the default Sitecore security domain specified by the `/App_Config/Security/Domains.config` file.

www.it-ebooks.info

Alternatively, you can use the static `Sitecore.Configuration.Factory.GetDomain()` method to access a named domain. For example, you can use the following code to access the domain named extranet:

```
Sitecore.Security.Domains.Domain extranet =
  Sitecore.Configuration.Factory.GetDomain("extranet");
Sitecore.Diagnostics.Assert.IsNotNull(extranet, "extranet");
```

PREVENTING SITECORE FROM APPLYING SECURITY

While it is generally desirable to configure access rights exactly as required for users to perform necessary operations, it is not always practical. In some cases you might need to override the access rights defined for the context user. For example, you might want to tease a visitor with a value in a field of an item to which that visitor does not have read access.

To give a user access rights that security rules would otherwise deny him for the duration of a block of code, you can use either the `Sitecore.Security.Accounts.UserSwitcher` class to impersonate a specific user that has the required access or the `Sitecore.SecurityModel.SecurityDisabler` class to disable Sitecore security within that block of code.



While these techniques are as secure as the code you write within the contained blocks of code, whenever possible, use access rights to provide CMS users with access to the items they need to perform their responsibilities, and to provide visitors to the published sites with read access to the items appropriate to their levels of privilege.

Impersonating a Sitecore User

You can use the `Sitecore.Security.Accounts.UserSwitcher` class to impersonate a specific user for the duration of a block of code. The `Sitecore.Security.Accounts.UserSwitcher` class causes a block of code to function as if the context user were the specified user. Any objects, properties, methods, or other components accessed within that block of code maintain the overridden level of access. At the close of the block of code, functionality automatically reverts to that of the original context user. For example, you can use the following code to impersonate a user named API in the Sitecore security domain:

```
string username = "sitecore\\api";

if (Sitecore.Security.Accounts.User.Exists(username))
{
  Sitecore.Security.Accounts.User apiUser =
    Sitecore.Security.Accounts.User.FromName(username, false);

  using (new Sitecore.Security.Accounts.UserSwitcher(apiUser))
  {
    // code to execute as the user named API in the Sitecore security domain
  }
}
```

www.it-ebooks.info

Disabling Sitecore Security

You use the `Sitecore.SecurityModel.SecurityDisabler` class to disable Sitecore security for the duration of a block of code. Code within such a block runs as if the context user were a Sitecore administrator — that code has all rights to all items, regardless of any access rights defined in or inherited by those items. The following code shows how to create a `Sitecore.SecurityModel.SecurityDisabler` block:

```
using (new Sitecore.SecurityModel.SecurityDisabler())
{
  // code to run as a user with administrative rights
}
```

You can create an instance of the `Sitecore.SecurityModel.SecurityEnabler` class to enable security within a block of code that disables security using the `Sitecore.SecurityModel.SecurityDisabler` class. The following code demonstrates both classes, with comments to indicate which code runs with the access rights of the context user and which runs with administrative access:

```
// code to run as the context user

using (new Sitecore.SecurityModel.SecurityDisabler())
{
  // code to run as a user with administrative rights

  using (new Sitecore.SecurityModel.SecurityEnabler())
  {
    // code to run as the context user
  }

  // code to run as a user with administrative rights
}

// code to run as the context user.
```

REQUIRING AUTHENTICATION FOR A MANAGED WEBSITE

To specify a log-in page for each managed website, set the `loginPage` attribute of the `/configuration/sitecore/sites/site` element in the Web.config file that defines that managed website. If you manage only one website under the default configuration, configure the `loginPage` attribute of the `/configuration/sitecore/sites/site` element in the Web.config file named `website` that represents the default published website. You can set the `loginPage` attribute to the URL of a file or an item relative to the document root of the website.



If you set the `loginPage` *attribute to the URL of an item rather than a file, ensure that the Anonymous user in the default security domain for the managed website has read access to that item. For example, you may have to provide read access to the Extranet security domain for the default managed website named website.*

www.it-ebooks.info

If a web client requests a URL that corresponds to an item to which that visitor does not have read access, and the `loginPage` attribute is absent for the context site, Sitecore redirects the browser to the URL specified by the `NoAccessUrl` setting in the Web.config file. If the `loginPage` attribute is present in that site definition, Sitecore redirects the browser to the page that attribute specifies.

Optionally, you can set the `requireLogin` attribute of the `/configuration/sitecore/sites/site` element in the Web.config file to `true` to require all users to authenticate in order to access the managed website. If the `requireLogin` attribute is `true` and a web client requests the URL of an item to which the visitor does not have read access, Sitecore redirects the browser to the URL specified by the `loginPage` attribute. If you do not set the `requireLogin` attribute to `true`, users who do not authenticate can access any portion of the solution that the Anonymous user on the default security domain for the managed website can read; restrict access rights for this user to control what unauthenticated users can access.

The log-in page can use ASP.NET log-in controls to authenticate users as described in The Sitecore Security API Cookbook (`http://bit.ly/tRRxiU`). Ensure that the security domain appears in the user name before attempting authentication. If authentication succeeds, redirect to a page to which the context user has read access. Alternatively, you can use the `Login()` method of the `Sitecore.Security.Authentication.AuthenticationManager` static class to authenticate a user. The first argument to this method specifies the user name, the second the password for that user, and the third whether to require authentication on future visits.

The Sitecore Security API Cookbook describes how you can configure the default authentication provider. This information includes how to configure password reset and retrieval, how to limit the number of invalid login attempts before Sitecore locks a user out of the system, and how to establish password complexity requirements. In addition to a sample login form and an explanation of how to use ASP.NET log-in controls, the cookbook contains a sample user self-registration form and an example that achieves the same result using the ASP.NET CreateUserWizard web control, as well as a sample password recovery form.

INTEGRATING AND EXTENDING SITECORE SECURITY

You can extend the Sitecore security infrastructure and integrate external security systems using ASP.NET membership, profile, and role providers. Sitecore uses these ASP.NET security providers for the following three purposes:

- **Membership Providers —** Verify users' credentials, including user names, passwords, and answers to the security questions used to validate authentication in specific scenarios
- **Profile Providers —** Represent users' profile information, such as e-mail addresses and custom profile properties
- **Role Providers —** Abstract role management and the relationships between users and roles

In short, membership providers support authentication, such as by abstracting the details of a system that manages usernames and passwords. Profile providers abstract systems that manage information about users, such as their preferences on the system. Role providers abstract systems that manage information about collections of users so that you can grant or deny access to those collections en-masse.

www.it-ebooks.info

Configuring ASP.NET Membership, Profile, and Role Providers

The default security providers supplied by Sitecore store information in tables in a relational database. To configure the default authentication, profile, and role providers you can update attributes of the `/configuration/system.web/membership/providers/add`, `/configuration/system.web/membership/profile/add`, and `/configuration/system.web/membership/roleManager/add` elements in the `/web.config` file.



Because they exist outside the `/configuration/sitecore` *element in the* `/web.config` *file, you cannot use a Web.config include file to configure these elements.*

For example, to store information in a database other than the Core database, set the `connectionStringName` attributes of these elements to the name of that connection string.

You can implement ASP.NET security providers that use external systems to manage authentication, profiles, and roles. Security providers consist of classes that inherit from abstract classes and implement various methods required to fulfill the contracts defined by those base classes. You can use the default providers included with Sitecore, you can use optional providers available as modules from Sitecore, or you can implement your own providers. Optional providers available from Sitecore include those in the Active Directory module `(http://bit.ly/rMAcPB)` further described in Chapter 10 and the CRM Security Provider `(http://bit.ly/twp3Z8)` module. Typically, you use the CRM provider to authenticate visitors on the published websites, while you use the Active Directory provider to authenticate CMS users. For instructions to implement custom authentication providers, see Sitecore Membership Providers Guide `(http://bit.ly/vWR0nG)`.

Building Virtual Users

You can implement virtual users to simulate Sitecore users after authenticating those users against an external security system. In some cases, virtual users are easier to implement than authentication providers. To implement virtual users, after successfully authenticating a user against your existing user management system, such as by using a log-in page, invoke APIs to create a virtual user in Sitecore, and then assign actual roles to that virtual user to control access rights to items in Sitecore databases.

Sitecore does not provide virtual roles; you can use the native roles provider or a custom roles provider, and assign virtual users to those roles. Typically you create roles in Sitecore based on roles defined in the external system, and associate virtual users with those roles when they log in. The difference between virtual users and actual roles is that between client sessions, Sitecore does not persist information about virtual users; they exist only for the duration of the browsing session. Sitecore persists information about roles that you create using the default role provider to a database, which is important because those roles need to appear in the user interface in order for you to apply access rights.

You can create a new unique identifier each time you create a virtual user, or store the unique identifier of the virtual user in the external system. To build a virtual user you can use the static

www.it-ebooks.info

`Sitecore.Security.Authentication.AuthenticationManager.BuildVirtualUser()` method, which returns an instance of the `Sitecore.Security.Accounts.User` class to which you can assign roles and profile properties. The first argument to the `BuildVirtualUser()` method is the user name, including the domain. The second parameter indicates whether to update the internal record of the user's last access date and time. For more information about and an example of creating virtual users, see `http://bit.ly/zCdEwG`.

Employing Federated Authentication

You can manage Sitecore users that authenticate using *federated authentication systems,* in which multiple systems share a common identification credential and user profile store. For example, the free Sitecore Social Connected (`http://bit.ly/tyQt6c`) module uses this technique to support authentication using Facebook, Twitter, LinkedIn, and Google+.

SWITCHING PROVIDERS

To use different authentication, user profile, and role management providers for different security domains, you can configure switching providers. The default providers use the same technical implementations for all security domains. The switching providers replace these defaults with providers that apply different actual implementations for different security domains. For example, you can use switching providers to use different security implementations for different managed websites, in which case each site specifies a security domain that the switching provider maps to an actual provider.

You can configure switching providers for individual functions, such as to use different providers for authentication for some security domains but a single provider for role management for all domains. You might authenticate CMS users against an internal LDAP using the Sitecore Active Directory authentication provider, but authenticate users of published websites using the default authentication provider so that you do not have to maintain those external users in your internal LDAP. Further, you might want to use the Active Directory provider for CMS user authentication, but use the default role provider even for the Sitecore security domain so that CMS users without administrative rights in Active Directory can manage roles specific to the CMS system through the CMS user interfaces.

For each provider implementation that you create, you should add an element to the Web.config file as described in this section. To use the provider, you must update a separate section of the actual `/web.config` file as described previously in the section, "Configuring ASP.NET Membership, Profile, and Role Providers."

The *authentication switching provider* allows different authentication providers for different security domains. To enable the authentication switching provider, set the `defaultProvider` attribute of the `/configuration/sitecore/membership` element in the Web.config file to `switcher`. This change instructs Sitecore to use the authentication switching provider, which evaluates the `/configuration/sitecore/switchingProviders/membership/provider` elements in the Web.config file to determine which actual authentication provider implementation to use for each security domain. To make additional authentication providers available to the authentication switching provider, insert `<add>` elements within the `<provider>` element.

www.it-ebooks.info

The *role management switching provider* allows different role management providers for different security domains. To enable the role management switching provider, set the `defaultProvider` attribute of the `/configuration/sitecore/roleManager` element in the Web.config file to `switcher`. This change instructs Sitecore to use the role management switching provider, which evaluates the `/configuration/sitecore/switchingProviders/roleManagement/provider` elements in the Web.config file to determine which actual role management provider implementation to use for each security domain. To make additional role management providers available to the role management switching provider, insert `<add>` elements within the `<provider>` element.

The *user profile management switching provider* allows different profile management providers for users in different security domains. To enable the user profile management switching provider, set the `defaultProvider` attribute of the `/configuration/sitecore/profile` element in the Web.config file to `switcher`. This change instructs Sitecore to use the user profile management switching provider, which evaluates the `/configuration/sitecore/switchingProviders/profile/provider` elements in the Web.config file to determine which actual user profile management provider to use for each security domain. To make additional user profile management provider implementations available to the user profile management switching provider, insert `<add>` elements within the `<provider>` element.

To specify which authentication provider Sitecore should use for a security domain, do the following:

1. Enable the authentication switching provider as described earlier in this section.
2. Add a `/configuration/sitecore/switchingProviders/membership/provider` element to the Web.config file.
3. Set the `domains` attribute of the new `<provider>` element to the name of the security domain.
4. Set the `providerName` attribute of the new `<provider>` element to the value of the `name` attribute of the `/configuration/sitecore/membership/providers/add` element in the `Web.config` file to use for that domain.

To specify which role management provider Sitecore should use for a security domain, follow these steps:

1. Enable the role management switching provider as described earlier in this section.
2. Add a `/configuration/sitecore/switchingProviders/roleManager/provider` element to the Web.config file.
3. Set the `domains` attribute of the new `<provider>` element to the name of the security domain.
4. Set the `providerName` attribute of the new `<provider>` element to the value of the `name` attribute of the `/configuration/sitecore/roleManager/providers/add` element in the `Web.config` file to use for that domain.

www.it-ebooks.info

To specify which role management provider Sitecore should use for a security domain, do the following:

1. Enable the user profile management switching provider as described earlier in this section.
2. Add a `/configuration/sitecore/switchingProviders/ profile/provider` element to the Web.config file.
3. Set the `domains` attribute of the new `<provider>` element to the name of the security domain.
4. Set the `providerName` attribute of the new `<provider>` element to the value of the `name` attribute of the `/configuration/sitecore/profile/providers/add` element in the `Web.config` file to use for that domain.

TAKE HOME POINTS

Even if your Sitecore solution does not require authentication for users of the managed websites, you should consider Sitecore security when designing your information architecture. Specifically, very few users should have rights beyond read for the home item for a managed website, especially the right to delete this item.

If you do need to secure your published websites, the Sitecore security model is quite comprehensive, including capabilities that exceed most requirements. You can restrict access to entire managed websites, branches within those sites, individual items within those branches, and individual fields within those items.

www.it-ebooks.info

www.it-ebooks.info

5

Error Management

WHAT'S IN THIS CHAPTER?

- Trapping exceptions with try...catch blocks
- Managing exceptions in XSL renderings, sublayouts, and web controls
- Handling exceptions at the application level with the global.asax file
- Redirecting to friendly error pages

This chapter describes error-management techniques that you can use with Sitecore solutions, including some provided by ASP.NET itself. You can trap errors at a number of layers, including the code, component, page, and application levels. This chapter covers error-management facilities from the lowest layer up to the highest.

Sitecore error management applies only to HTTP requests handled by Sitecore, which handles most of the requests that IIS handles with ASP.NET. As described in this chapter, Sitecore error management varies according to the configuration defined under the `/configuration/system.web/customErrors` element in the Web.config file.

Sitecore attempts to log every exception that it encounters. Such log entries include stack traces. An optimized Sitecore solution should log few or no exceptions. You should attempt to identify all potential errors in development and test environments before deploying that code to production. For more information about Sitecore logging, see Chapter 6. For more information about error management with the Sitecore ASP.NET content management system (CMS), see The Sitecore Presentation Component API Cookbook (`http://bit.ly/r76c6K`). In fact, much of the information and code in this chapter derive directly from that work.

www.it-ebooks.info

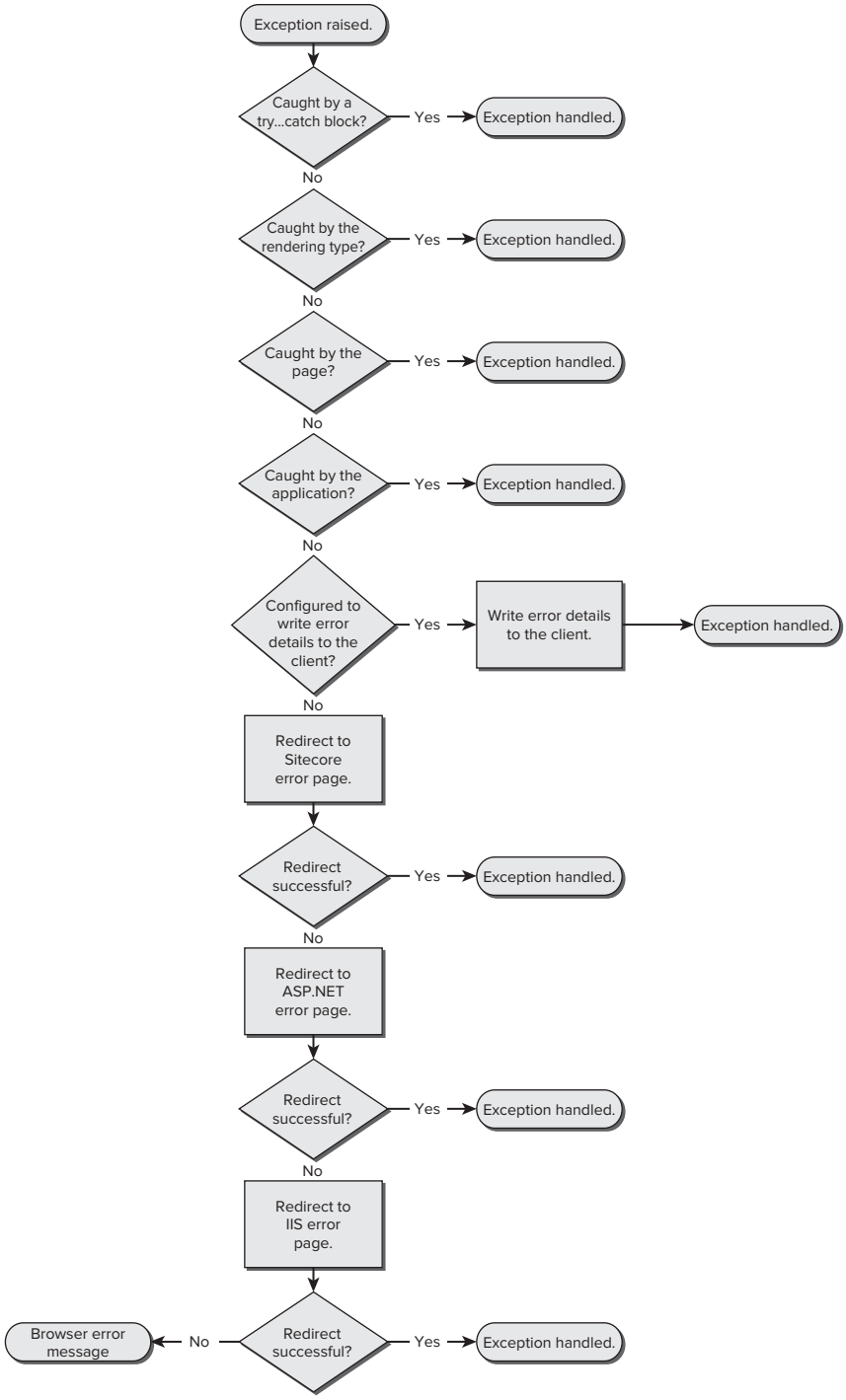


FIGURE 5-1

www.it-ebooks.info

EXCEPTION LEVELS

With Internet Information Services (IIS), ASP.NET, and Sitecore, you can handle errors and exceptions at a number of levels. The diagram shown in Figure 5-1 summarizes your options. While the diagram appears top-down, exceptions that you do not trap at a lower level actually bubble up through the call stack, from the individual line of code that generated the exception through the presentation control, the page, the application, Sitecore, ASP.NET, IIS, and potentially even to the browser (not shown in Figure 5-1), which conceptually handles errors in IIS.

DESIGNING AN ERROR HELPER CLASS

You will likely want to handle exceptions at a number of levels and to apply the same exception management logic for every type of presentation component, including layouts, eXtensible Stylesheet Language (XSL) renderings, sublayouts, and web controls. Your application may benefit from a class that encapsulates error management logic, such as access to the `/configuration/system.web/customErrors` element in the `/web.config` file to determine whether to render information about an error or to redirect to a friendly error page instead. For more information about this element in the `/web.config` file, see the ASP.NET Error Management Pages section of this chapter. You can create an error helper class based on the prototype shown in Listing 5-1. Subsequent examples in this chapter depend on this prototype rather than repeatedly duplicating its functionality.



Available for download on Wrox.com

LISTING 5-1: Implementing an ErrorHelper class

```
namespace SitecoreBook.Web.UI
{
  using System;
  using System.Web.Configuration;
  using System.Web.UI;

  public static class ErrorHelper
  {
    // represents the /configuration/system.web/customErrors section of web.config
    private static CustomErrorsSection _config = null;

    // URL to handle server errors
    private static string _error500Url = null;

    // lazy-loading representation of /configuration/system.web/customErrors
    public static CustomErrorsSection Config
    {
      get
      {
        if (_config == null)
        {
          System.Configuration.Configuration config =
            WebConfigurationManager.OpenWebConfiguration("/");
          _config =
            (CustomErrorsSection)config.GetSection("system.web/customErrors");
```

continues

www.it-ebooks.info

LISTING 5-1 *(continued)*

```
        Sitecore.Diagnostics.Assert.IsNotNull(_config, "customErrors");
      }

      return _config;
    }
  }

  // lazy-loading URL to handle server errors
  public static string Error500Url
  {
    get
    {
      // if this application has not attempted to determine the error page
      // to handle the HTTP 500 condition, but the
      // /configuration/system.web/customErrors/error element exists in web.config,
      // check for a contained <error> element with a statusCode attribute of 500
      if (_error500Url == null && Config != null)
      {
        // check for a /configuration/system.web/customErrors/error 500 element
        CustomError error500 = Config.Errors.Get(
          500.ToString(System.Globalization.CultureInfo.InvariantCulture));

        if (error500 != null && !String.IsNullOrEmpty(error500.Redirect))
        {
          _error500Url = error500.Redirect;
        }

        // if no such element exists, use the defaultRedirect attribute
        // of the /configuration/system.web/customErrors/error element
        if (String.IsNullOrEmpty(_error500Url))
        {
          _error500Url = Config.DefaultRedirect;
        }

        // default to using an empty string if the defaultRedirect attribute is null
        if (_error500Url == null)
        {
          _error500Url = String.Empty;
        }
      }

      return _error500Url;
    }
  }

  // Determine whether to redirect after an error.
  public static bool ShouldRedirect()
  {
    // if the user is debugging, they have authenticated; do not redirect
    if (Sitecore.Context.PageMode.IsDebugging)
    {
      return false;
```

www.it-ebooks.info

```
  }

  // if the /configuration/system.web/customErrors element is absent
  if (Config == null)
  {
    return true;
  }

  // if redirection to friendly error pages is completely disabled
  if (Config.Mode == System.Web.Configuration.CustomErrorsMode.Off)
  {
    return false;
  }

  // if configured to redirect only remote clients and this client is local
  if (Config.Mode == System.Web.Configuration.CustomErrorsMode.RemoteOnly
    && System.Web.HttpContext.Current.Request.IsLocal)
  {
    return false;
  }

  // otherwise, redirect by default
  return true;
}

// implementation of redirection logic
public static void Redirect()
{
  if (!String.IsNullOrEmpty(Error500Url))
  {
    string url = Error500Url
      + "?aspxerrorpath="
      + Sitecore.Web.WebUtil.GetLocalPath(Sitecore.Context.RawUrl);
    Sitecore.Web.WebUtil.Redirect(url);
  }
  else
  {
    Sitecore.Web.WebUtil.RedirectToErrorPage(
      "replace this text with your friendly error message");
  }
}

// log an exception
public static void LogException(string message, Exception ex, object owner)
{
  Sitecore.Diagnostics.Log.Error(message, ex, owner);
}

// render an error to the client browser
public static void RenderError(
  string message,
  string details,
  HtmlTextWriter output)
{
  Sitecore.Web.UI.WebControls.ErrorControl errorControl;
```

continues

www.it-ebooks.info

LISTING 5-1 *(continued)*

```
            errorControl = Sitecore.Web.UI.WebControl.GetErrorControl(
                message,
                details) as Sitecore.Web.UI.WebControls.ErrorControl;
            Sitecore.Diagnostics.Assert.IsNotNull(errorControl, "ErrorControl");
            errorControl.RenderControl(output);
            string error = String.Format(
              "A rendering error occurred: {0} (details: {1})",
              message,
              details);
            Sitecore.Diagnostics.Log.Error(error, errorControl);
          }
        }
      }
```

In the `SitecoreBook.Web.UI.ErrorHelper` class shown in Listing 5-1:

- The `Config` property exposes the `/configuration/system.web/customErrors` section of the `/web.config` file.
- The `Error500Url` property exposes the URL of a generic error management page.
- The `ShouldRedirect()` method returns `false` if any of the following are true:
  - The user is in the browser-based debugger.
  - The value of the `mode` attribute of the `/configuration/system.web/customErrors` element in the `/web.config` file is `Off`.
  - The value of the `mode` attribute of the `/configuration/system.web/customErrors` element in the `/web.config` is `RemoteOnly` and the current HTTTP request originated from the local system.

  The `ShouldRedirect()` method returns `true` by default or if any of the following are true:
  - The `/configuration/system.web/customErrors` element does not exist in the `/web.config` file and the use is not debugging.
  - The value of the `mode` attribute of the `/configuration/system.web/customErrors` element in the `/web.config` file is `On` or is `RemoteOnly`, the current HTTP request is not from the local system, and the user is not debugging.
- The `Redirect()` method redirects the client to the URL specified by the `redirect` attribute of the `/configuration/system.web/customErrors/error` element in the `/web.config` file with a value of `500` for the `statusCode` attribute.
- If no such `/configuration/system.web/customErrors/error` element exists, the `Redirect()` method redirects the client to the URL specified by the `defaultRedirect` attribute of the `/configuration/system.web/customErrors` element in the `/web.config` file.
- If the `/configuration/system.web/customErrors` element does not exist or its `mode` attribute does not exist or does not have a value, the `Redirect()` method redirects the client to the URL specified by the `ErrorPage` setting in the Web.config file.
- The `RenderError()` method invokes the error control as described in the further sections of this chapter, and then writes the error message to the Sitecore log.

www.it-ebooks.info

IMPLEMENTING YOUR OWN EXCEPTION CLASSES

You should implement your own classes to represent the types of exceptions that can occur in your code. Your exception classes can derive from the `System.Exception` class or another class that inherits from `System.Exception`. You may want to create an abstract base class from which to derive all of your classes that represent specific types of exceptions, meaning that all those classes derive from that same abstract base class. For an example of this type of abstract base class for exceptions, see Listing 5-2.



Available for download on Wrox.com

LISTING 5-2: Accessing the URL associated with an exception

```
namespace SitecoreBook.Exceptions
{
  using System;
  using System.Web;

  public abstract class CustomExceptionBase : System.Exception
  {
    // constructor accepts exception message
    public CustomExceptionBase(string message) : base(message)
    {
    }

    // return the results of calling ToString(), plus when possible, the original
    // requested URL and current URL (Sitecore may have modified the requested URL)
    public string GetLogMessage()
    {
      string result = this.ToString();

      if (!String.IsNullOrEmpty(Sitecore.Context.RawUrl))
      {
        result += System.Environment.NewLine
          + this.GetType().ToString()
          + " : Original URL : "
          + Sitecore.Context.RawUrl;
      }

      if (HttpContext.Current != null
        && HttpContext.Current.Request != null
        && !String.IsNullOrEmpty(HttpContext.Current.Request.RawUrl))
      {
        result += System.Environment.NewLine
          + this.GetType().ToString()
          + " : Current URL : "
          + HttpContext.Current.Request.RawUrl;
      }

      return result;
    }
  }
}
```

www.it-ebooks.info

Listing 5-2 defines the `SitecoreBook.Exceptions.CustomExceptionBase` abstract class from which you can derive any number of classes that represent specific types of exceptions. Each such deriving class can invoke the `GetLogMessage()` method in this abstract base class, which attempts to add the type of the exception and the URL associated with the current HTTP request to the result of calling the `ToString()` method in the `System.Exception` class from which the `SitecoreBook.Exceptions.CustomExceptionBase` class derives. Where you trap such exceptions, you can write the value returned by the `GetLogMessage()` method to the Sitecore log or elsewhere to include the exception type and the requested URL in error messages whenever possible.

You can create any number of classes that derive from the `SitecoreBook.Exceptions.CustomExceptionBase` class to represent specific types of exceptions. For example:

```
namespace SitecoreBook.Exceptions
{
  public class ActualException : SitecoreBook.Exceptions.CustomExceptionBase
  {
    public ActualException(string message) : base(message)
    {
    }
  }
}
```

When you experience an error condition that you cannot manage or would like to handle at a higher level, throw an exception as shown in the following example:

```
throw new SitecoreBook.Exceptions.ActualException("Error message");
```

TRAPPING EXCEPTIONS WITH TRY...CATCH...FINALLY BLOCKS

Follow best practices for trapping exceptions with `try...catch...finally` blocks (also called `try...catch` blocks) throughout your code. If you are not familiar with exception management practices, research this subject on the Internet. Implement and use your own exception classes as described in the previous section. Trap only those exceptions that you expect and from which you can recover. Trap exceptions as close as possible to where they occur.

You can use `try...catch` blocks anywhere in .NET code, and you can nest `try...catch` blocks within `try...catch` blocks. Other than in XSL extension methods and controls, you cannot use `try...catch` blocks in XSL renderings.

To log an exception, you can pass that exception as the second parameter to the `Error()` method in the `Sitecore.Diagnostics.Log` static class using the signature of that method that accepts three parameters, as shown in the last call to that `Error()` method in the following example:

```
try
{
  // if some error condition is true
  throw new SitecoreBook.Exceptions.ActualException("Error message");
}
catch (SitecoreBook.Exceptions.ActualException ex)
{
  Sitecore.Diagnostics.Log.Error(ex.GetLogMessage(), this);
```

www.it-ebooks.info

```
}
catch (SitecoreBook.Exceptions.CustomExceptionBase ex)
{
  // this catch block may appear redundant with the previous catch block,
  // but you may want to handle specific types of exceptions differently than others
  Sitecore.Diagnostics.Log.Error(ex.GetLogMessage(), this);
}
catch (Exception ex)
{
  Sitecore.Diagnostics.Log.Error("unexpected exception", ex, this);
  throw (ex);
}
```

The first two calls to the `Error()` method in the `Sitecore.Diagnostics.Log` static class shown in this example use a signature of that method that accepts only two parameters: a string containing a message to log, and an object to associate with that message. The third call to the `Error()` method accepts three parameters: a message to log, an exception to log, and the object to associate with that message. In either case, the logging infrastructure can use the type of the object passed as the last argument to route the message to different logging sinks, though such differentiation is uncommon in practice.

HANDING ERRORS IN PRESENTATION CONTROLS

You can configure how the layout engine handles the following error conditions in presentation components:

- Syntax errors in XSL renderings
- Runtime errors in XSL renderings
- Exceptions in web controls
- Exceptions in sublayouts

For example, you can determine whether to render information about the error in-line with the rest of the page, render the rest of the page without providing any output for the presentation component that encountered the error, redirect the browser to a page containing a friendly error message, or present the error information in some other way. You can base your decision on the page mode (such as whether the user is debugging or using the Page Editor), configuration in the Web.config file or elsewhere, or other logic.

Handing Syntax and Runtime Errors in XSL Renderings

XSL renderings can result in two types of errors:

- Syntax errors in the XSL code
- Exceptions raised during XSL processing

XSL renderings do not support compile-time error detection. When you change an XSL rendering, you do not need to compile it before the system can attempt to use that code. You might not know an XSL rendering contains a syntax error until you use it (or worse, someone else tries to use it).

www.it-ebooks.info

Whether an XSL rendering contains a syntax error or throws an exception, the layout engine uses the same approach to handle that error. The `Sitecore.Web.UI.WebControls.XslFile` class that Sitecore uses to process XSL renderings inherits from the `Sitecore.Web.UI.WebControl` base class for web controls. The layout engine calls the `Render()` method of the `Sitecore.Web.UI.WebControl` abstract base class. If output from the rendering under the same processing conditions does not exist in the output cache for the context site, the `Render()` method invokes the `DoRender()` method defined in the `Sitecore.Web.UI.WebControls.XslFile` class. For more information about caching the output of renderings, see Chapter 6. The `DoRender()` method contains a `try...catch` block that handles both syntax errors and exceptions raised by XSL extension methods and controls as well as exceptions thrown by the XSL transformation engine and the underlying platform.

If an error occurs while processing an XSL rendering, the layout engine (specifically, the `catch` block in the `DoRender()` method of the `Sitecore.Web.UI.WebControls.XslFile` class) invokes the `RenderError()` method of the `Sitecore.Web.UI.WebControl` abstract base class. The `RenderError()` method prevents the layout engine from caching the output of the control and invokes the `RenderControl()` method of the error control specified by the `type` attribute of the `/configuration/ErrorControl` element in the `/App_Config/Prototypes.config` file.

The `Sitecore.Web.UI.WebControl` abstract base class maintains a flag that indicates whether an error occurred during rendering, which controls whether the layout engine caches the output of the control. The `RenderError()` method of the `Sitecore.Web.UI.WebControl` abstract base class sets this flag. If you override error management and you want to cache the fact that an error occurred, do not call the `RenderError()` method. Otherwise, call the `RenderError()` method to set the flag that prevents the layout engine from caching the output of the XSL rendering. Alternatively, you can set your own flag if an exception occurs, and implement the `GetCachingID()` method to return an empty string if that flag is set, effectively preventing Sitecore from caching the output of that control in that condition.

The default error control renders an icon and a message that indicates that an error occurred. Depending on the value of the `mode` attribute of the `/configuration/system.web/customErrors` element in the Web.config file, the error control may output information about the error. You can override the type specified by the `/configuration/ErrorControl` element in the `/App_Config/Prototypes.config` file with one that manages exceptions differently, such as to render no output when a rendering encounters an error.

You cannot differentiate an XSL syntax error from an XSL runtime error without overriding the `DoRender()` method of the `Sitecore.Web.UI.WebControls.XslFile` class. Additionally, an exception can occur after an XSL rendering has generated some output, in which case both that output and the information about the exception can appear in the page. You can avoid this condition by replacing the `Sitecore.Web.UI.WebControls.XslFile` class with a version that writes to a temporary object using the `System.Web.UI.HtmlTextWriter` to prevent output in case of error, and then writes the content of that object to the output stream if no error occurs. If you replace the `Sitecore.Web.UI.WebControls.XslFile` class, you must update the `type` attribute of the `/configuration/sitecore/renderingControls/control` element in the Web.config file with a value of `xsl rendering` for the `template` attribute, similar to the technique described for sublayouts in the following section.

As described in Chapter 3, you can bind XSL files statically in layouts and sublayouts using code in those files, or dynamically to placeholders using layout details. To maximize consistency and minimize administration of layout details when all uses of a layout or sublayout invoke the same rendering in the same location, bind the rendering statically to the layout or sublayout. To apply custom

www.it-ebooks.info

logic to handle errors in XSL renderings bound either statically and dynamically, you must override methods in two classes:

- Override the `DoRender()` method of the `Sitecore.Web.UI.WebControls.XslFile` class with error management logic.
- Override the `GetControl()` method of the `Sitecore.Web.UI.XslControlRenderingType` class to return instances of the class that overrides the `DoRender()` method rather than the default `Sitecore.Web.UI.WebControls.XslFile` class.

The prototype shown in Listing 5-3 for a replacement to the `Sitecore.Web.UI.WebControls.XslFile` class used to process XSL renderings prevents XSL renderings from writing to the output stream before an exception occurs.



LISTING 5-3: Defining the XslFile class replacement

```
namespace SitecoreBook.Web.UI.WebControls
{
  using System;
  using System.IO;
  using System.Text;
  using System.Web.UI;

  public class XslFile : Sitecore.Web.UI.WebControls.XslFile
  {
    // use default Sitecore error control instead of custom error control
    public bool DisableErrorControl
    {
      get;
      set;
    }

    // cache the output of this control even if an exception occurs
    private bool CacheAfterException
    {
      get;
      set;
    }

    // render the XSL transformation
    protected override void DoRender(System.Web.UI.HtmlTextWriter output)
    {
      // if error control is disabled, call the default implementation and exit
      if (this.DisableErrorControl)
      {
        base.DoRender(output);
        return;
      }

      StringBuilder sb = new StringBuilder();
      HtmlTextWriter writer = new HtmlTextWriter(new StringWriter(sb));

      try
```

continues

www.it-ebooks.info

LISTING 5-3 *(continued)*

```
        {
          // invoke the XSL transformation and write the output to the output stream
          base.DoRender(writer);
          writer.Close();

          // no exception occurred; write the output to the output stream
          output.Write(sb.ToString());
        }
        catch (Exception ex)
        {
          // in case of exception, log an error message containing the and actual URLs
          string message = this + " : " + ex.GetType().ToString();

          if (!String.IsNullOrEmpty(this.Path))
          {
            message += " (" + this.Path + ")";
          }

          message += " processing " + Sitecore.Context.RawUrl;
          string details = ex.ToString();
          ErrorHelper.LogException(message, ex, this);

          // redirect if configured to redirect under current processing conditions
          if (ErrorHelper.ShouldRedirect())
          {
            ErrorHelper.Redirect();
            return;
          }

          // if configured to cache in case of exception, invoke custom RenderError()
          if (this.CacheAfterException)
          {
            ErrorHelper.RenderError(message, details, output);
            return;
          }

          // if not configured to cache after an exception, invoke default RenderError()
          this.RenderError(message, this.Path, details, output);
        }
      }
    }
  }
```

In the `SitecoreBook.Web.UI.WebControls.XslFile` class shown in Listing 5-3:

- The `DisableErrorControl` property controls whether to apply the custom error management logic.
- The `CacheAfterException` property controls whether the layout engine caches output for this instance of the control if it throws an exception.
- If the `DisableErrorControl` property is `true`, the `DoRender()` method calls the corresponding method in the `Sitecore.Web.UI.WebControls.XslFile` base class and then exits.

www.it-ebooks.info

- If the `DisableErrorControl` property is `false`, the `DoRender()` method creates a temporary `System.Web.UI.HtmlTextWriter` object, and passes that object to the corresponding method in the `Sitecore.Web.UI.WebControls.XslFile` base class.
- If the `DoRender()` method in the `Sitecore.Web.UI.WebControls.XslFile` base class does not throw an exception, the `DoRender()` method writes the content of the temporary `System.Web.UI.HtmlTextWriter` object to the output stream.

If the `DoRender()` method in the `Sitecore.Web.UI.WebControls.XslFile` base class throws an exception, the `DoRender()` method in the `SitecoreBook.Web.UI.WebControls.XslFile` class shown in Listing 5-3 logs the exception, and then uses the `SitecoreBook.Web.UI.ErrorHelper` class shown in Listing 5-1 to determine whether to redirect and whether to render information about the error to the browser.

The approach you use to replace the default `Sitecore.Web.UI.WebControls.XslFile` web control Sitecore uses by default to invoke XSL renderings differs depending on whether you bind XSL renderings statically or dynamically. To use the `Sitecore.Web.UI.WebControls.XslFile` class where you bind XSL renderings statically:

1. Add a `/configuration/system.web/pages/controls/add` element to the `/web.config` file.
2. Set the `tagPrefix` attribute in the new element to a unique prefix that identifies your class library.
3. Set the `namespace` and `assembly` attributes of the new element to the signature of your class. For example:

```
<add tagPrefix="sb"
  namespace="SitecoreBook.Web.UI.WebControls"
  assembly="SitecoreBook"/>
```

4. Replace the `sc` tag prefix in `<sc:xslfile>` elements in your layouts and sublayouts with your tag prefix. For example:

```
<sb:xslfile runat="server" ... />
```

To use your class where you bind XSL renderings dynamically using layout details, update the `type` attribute of the `/configuration/sitecore/renderingControls/control` element in the Web .config file with a value of `xsl rendering` for the `template` attribute to specify the signature of your class based on the prototype shown in Listing 5-4. This `type` attribute specifies a class that serves as a factory to create objects that implement XSL transformations bound to placeholders.



LISTING 5-4: Defining a GetControl() prototype

```
namespace SitecoreBook.Web.UI
{
  using System.Collections.Specialized;
  using System.Web.UI;

  public class XslControlRenderingType : Sitecore.Web.UI.XslControlRenderingType
```

continues

www.it-ebooks.info

LISTING 5-4 *(continued)*

```
    {
      // retrieve an object that renders an XSL transformation
      public override Control GetControl(
        NameValueCollection parameters, bool assert)
      {
        // create a control using the custom XslFile implementation
        SitecoreBook.Web.UI.WebControls.XslFile xslFile =
          new SitecoreBook.Web.UI.WebControls.XslFile();

        // pass all defined parameters to the control
        foreach (string str in parameters.Keys)
        {
          Sitecore.Reflection.ReflectionUtil.SetProperty(xslFile, str, parameters[str]);
        }

        // return the control
        return xslFile;
      }
    }
  }
```

The `SitecoreBook.Web.UI.XslControlRenderingType` class shown in Listing 5-4 inherits from the `Sitecore.Web.UI.XslControlRenderingType` class to provide a replacement for that type. This implementation overrides the `GetControl()` method in that class to return an instance of the `SitecoreBook.Web.UI.WebControls.XslFile` class defined in Listing 5-3 instead of the `Sitecore.Web.UI.WebControls.XslFile` class used by the default implementation. Use of the `SitecoreBook.Web.UI.WebControls.XslFile` class prevents the layout engine from writing anything to the output stream when an XSL rendering encounters an exception, even when you bind XSL renderings dynamically using placeholders, assuming that you implement and configure the `SitecoreBook.Web.UI.WebControls.XslControlRenderingType` class shown in Listing 5-4 as described previously.



The `GetControl()` *method in the default* `Sitecore.Web.UI.XslControlRenderingType` *class does not use the Boolean assert argument passed as the second argument to the method, so neither does the corresponding method in the* `SitecoreBook.Web.UI.XslControlRenderingType` *class shown in Listing 5-4 that overrides that class.*

Handling Exceptions in Sublayouts

By default, Sitecore does not handle exceptions thrown by sublayouts — they bubble up to the page and potentially to the application or even IIS. Technically, Sitecore uses the `Sitecore.Web.UI.WebControls.Sublayout` web control class to invoke sublayouts. That class inherits from the `Sitecore.Web.UI.WebControl` class. As with XSL renderings, the `Render()` method of `Sitecore.Web.UI.WebControl` calls the `DoRender()` method of `Sitecore.Web.UI.WebControls.Sublayout` web control class, which does not include a `try...catch` block. Just like with XSL renderings, you can override the type that returns the `Sitecore.Web.UI.WebControls.Sublayout` objects that the

www.it-ebooks.info

layout engine uses to invoke sublayouts with a custom version that returns instances of a derivative class that adds exception management logic. You can even apply the same error control logic used for XSL renderings.



This exception management logic applies only to the rendering phase in the ASP.NET page lifecycle. Use standard ASP.NET techniques, such as `try... catch` *blocks, to trap exceptions at other phases.*

To handle errors in sublayouts bound either statically or dynamically, you must override methods in two classes:

- Override the `DoRender()` method of the `Sitecore.Web.UI.WebControls.Sublayout` class with error management logic.
- Override the `GetControl()` method of the `Sitecore.Web.UI.SublayoutRenderingType` class to return instances of the class that overrides the `DoRender()` method rather than the default `Sitecore.Web.UI.WebControls.Sublayout` class.

In some cases, an exception can occur in a presentation component after that component has generated output to the page. To prevent this possibility, this solution creates a temporary object using the `System.Web.UI.HtmlTextWriter` class for use within the `try...catch` block, and writes the contents of that writer to the actual output stream only if no exception occurs.

In addition to exceptions thrown during rendering of a sublayout, just like XSL renderings, the `.ascx` file that implements the sublayout can contain syntax errors. You can override the `CreateChildControls()` method in the `Sitecore.Web.UI.WebControls.Sublayout` class to handle some types of syntax errors in sublayouts. Simply wrap the call to the corresponding method in the base class with a `try...catch` block. Over time, you may find additional methods that require error control, including methods for other types of presentation components such as web controls. The code shown in Listing 5-5 provides a replacement for the `Sitecore.Web.UI.WebControls.Sublayout` class that implements the features described previously in this section.



Available for download on Wrox.com

LISTING 5-5: Defining a Sublayout class prototype

```
namespace SitecoreBook.Web.UI.WebControls
{
  using System;
  using System.IO;
  using System.Text;
  using System.Web.UI;

  // custom implementation of web control to apply exception management to sublayouts
  public class Sublayout : Sitecore.Web.UI.WebControls.Sublayout
  {
    // disable custom exception management
    public bool DisableErrorManagement
```

continues

www.it-ebooks.info

LISTING 5-5 *(continued)*

```
{
  get;
  set;
}

// cache the output of this sublayout even if an exception occurs
private bool CacheAfterException
{
  get;
  set;
}

// last exception known to have occurred
private Exception _lastException;

// override CreateChildControls() to add exception management
protected override void CreateChildControls()
{
  // if instructed not to use custom exception management,
  // call the corresponding method in the base class and exit
  if (this.DisableErrorManagement)
  {
    base.CreateChildControls();
    return;
  }

  // add exception management to the corresponding method in the base class
  try
  {
    base.CreateChildControls();
  }
  catch (Exception ex)
  {
    // log the exception
    ErrorHelper.LogException("Error in CreateChildControls()", ex, this);

    // store the exception
    this._lastException = ex;
  }
}

// override Render() to add exception management
protected override void DoRender(System.Web.UI.HtmlTextWriter output)
{
  // if instructed not to use custom exception management,
  // call the corresponding method in the base class and exit
  if (this.DisableErrorManagement)
  {
    base.DoRender(output);
    return;
  }

  // if an exception has already occurred, such as in CreateChildControls()
```

www.it-ebooks.info

```
if (this._lastException != null)
{
  // if configured to redirect under current conditions, redirect and exit
  if (ErrorHelper.ShouldRedirect())
  {
    ErrorHelper.Redirect();
    return;
  }

  // exception message
  string message = this._lastException.Message;

  // details about the exception
  string details = this._lastException.ToString();

  // if configured to cache after an exception, call custom RenderError()
  if (this.CacheAfterException)
  {
    ErrorHelper.RenderError(message, details, output);
    return;
  }

  // if configured not to cache after an exception, call default RenderError()
  this.RenderError(message, this.Path, details, output);
  return;
}

// the output of the sublayout
StringBuilder sb = new StringBuilder();
HtmlTextWriter writer = new HtmlTextWriter(new StringWriter(sb));

// add error control around DoRender() in the base class
try
{
  base.DoRender(writer);
  writer.Close();

  // no exception occurred; write the output to the output stream
  output.Write(sb.ToString());
}
catch (Exception ex)
{
  // exception message
  string message = this
    + " : "
    + ex.GetType().ToString()
    + " in sublayout "
    + this.Path
    + " processing "
    + Sitecore.Context.RawUrl;

  // details about the exception
  string details = ex.ToString();

  // log the exception
```

continues

www.it-ebooks.info

LISTING 5-5 *(continued)*

```
            ErrorHelper.LogException(message, ex, this);

            // if configured to redirect under current conditions, redirect and exit
            if (ErrorHelper.ShouldRedirect())
            {
              ErrorHelper.Redirect();
              return;
            }

            // if configured to cache after exception, call custom RenderError() and exit
            if (this.CacheAfterException)
            {
              ErrorHelper.RenderError(message, details, output);
              return;
            }

            // if configured not to cache after an exception, call default RenderError().
            this.RenderError(message, this.Path, details, output);
          }
        }
      }
    }
```

In the `SitecoreBook.Web.UI.WebControls.Sublayout` web control class shown in Listing 5-5:

- The `CacheAfterException` property designates whether the control should attempt to cache its output after experiencing an exception.
- The `DisableErrorManagement` property enables or disables custom error management. If the `DisableErrorManagement` property is `true`, the `CreateChildControls()` method calls the corresponding method in the base `Sitecore.Web.UI.WebControls.Sublayout` class, and then exits.
- If the `DisableErrorManagement` property is `false`, the `CreateChildControls()` method invokes the corresponding method in the base `Sitecore.Web.UI.WebControls.Sublayout` class, and sets the `_lastException` variable to any exception thrown by that method call.
- If the `DisableErrorManagement` property is `true`, the `DoRender()` method invokes the corresponding method in the `Sitecore.Web.UI.WebControls.Sublayout` base class, and then exits.
- If the `DisableErrorManagement` property is `false` and the `CreateChildControls()` method generates an exception, the `DoRender()` method renders the error control and then exits.
- If the `DisableErrorManagement` property is `false` and the `CreateChildControls()` method does not generate an exception, the `DoRender()` method creates a temporary `System.Web.UI.HtmlTextWriter` and passes that object to the `DoRender()` method of the `Sitecore.Web.UI.WebControls.Sublayout` base class.

www.it-ebooks.info

- If the call to the `DoRender()` method in the base class does not generate an exception, the `DoRender()` method writes the data in the temporary `System.Web.UI.HtmlTextWriter` to the output stream.
- If the call to the `DoRender()` method in the base class generates an exception, the `DoRender()` method logs the exception and uses the error helper class to determine whether to redirect.
- Depending on configuration, the `DoRender()` method may render information about the error.

Just like XSL renderings as described in the previous section, you can bind sublayouts statically to layouts and sublayouts using code in those files, or dynamically to placeholders using layout details. To use the `SitecoreBook.Web.UI.WebControls.Sublayout` class shown in Listing 5-5 where you bind sublayouts statically:

1. Add a `/configuration/system.web/pages/controls/add` element to the `/web.config` file.
2. Set the `tagPrefix` attribute in the new element to a unique prefix that identifies your class library.
3. Set the `namespace` and `assembly` attributes of the new element to the signature of your class. For example:

```
<add tagPrefix="sb"
  namespace="SitecoreBook.Web.UI.WebControls" assembly="SitecoreBook"/>
```

4. Replace the `sc` tag prefix in `<sc:sublayout>` elements of your layouts and sublayouts with your tag prefix. For example:

```
<sb:sublayout runat="server" ... />
```

To use your class where you bind sublayouts dynamically, update the `type` attribute of the `/configuration/sitecore/renderingControls/control` element in the Web.config file with a value of `sublayout` for the `template` attribute to specify the signature of your class based on the prototype shown in Listing 5-6.



LISTING 5-6: Defining a SublayoutRenderingType class prototype

```
namespace SitecoreBook.Web.UI
{
  using System.Collections.Specialized;
  using System.Web.UI;

  // works as a factory for creating controls to implement sublayout rendering
  public class SublayoutRenderingType : Sitecore.Web.UI.SublayoutRenderingType
  {
    // retrieve a control that renders a sublayout
    public override Control GetControl(
      NameValueCollection parameters, bool assert)
    {
      // create an instance of the custom Sublayout control
```

continues

www.it-ebooks.info

LISTING 5-6 *(continued)*

```
        SitecoreBook.Web.UI.WebControls.Sublayout sublayout =
          new SitecoreBook.Web.UI.WebControls.Sublayout();

        // pass all defined parameters to the control
        foreach (string str in parameters.Keys)
        {
          Sitecore.Reflection.ReflectionUtil.SetProperty(
            sublayout,
            str,
            parameters[str]);
        }

        // return the control
        return sublayout;
      }
    }
  }
```

The `SitecoreBook.Web.UI.SublayoutRenderingType` class shown in Listing 5-6 inherits from the `Sitecore.Web.UI.SublayoutRenderingType` class to provide a replacement for that type. This implementation overrides the `GetControl()` method in that class to return an instance of the `SitecoreBook.Web.UI.WebControls.Sublayout` class defined in Listing 5-5 instead of the `Sitecore.Web.UI.WebControls.Sublayout` class used by the default implementation. Use of the `SitecoreBook.Web.UI.WebControls.Sublayout` class prevents the layout engine from writing anything to the output stream when a sublayout encounters an exception, even when you bind sublayouts dynamically using placeholders, assuming that you implement and configure the `SitecoreBook.Web.UI.SublayoutRenderingType` class shown in Listing 5-6 as described previously.

Unlike the solution for XSL renderings described in the previous section, which handles syntax errors in the `.xslt` files that implement XSL renderings, the solution described in this section does not trap every possible error that could occur while processing a sublayout, such as when the `Inherits` attribute of the `Control` directive in a sublayout file (`.ascx` file) specifies a class that does not exist. You should identify such cases in test environments before deploying them to production. For more information about managing exceptions in sublayouts, see my blog posts at `http://bit.ly/olEH9B` and `http://bit.ly/olEH9B`.

Handling Exceptions in Web Controls

By default, the layout engine does not handle exceptions in web controls — they bubble up to the page and potentially to the application or even IIS. You can configure Sitecore to handle exceptions in web controls in a manner similar to those previously described for XSL renderings and sublayouts.

All web controls that you use with Sitecore should inherit directly or indirectly from the `Sitecore.Web.UI.WebControl` abstract base class, which allows the layout engine to bind those controls to placeholders and cache their output. ASP.NET invokes the `Render()` method defined in the `Sitecore.Web.UI.WebControl` abstract base class, which determines whether to retrieve output from cache or invoke the `DoRender()` method of that individual web control. You can add `try...`

www.it-ebooks.info

`catch` blocks within the `DoRender()` methods of your web controls to trap specific types of exceptions, and you can apply an error control to manage exceptions not trapped elsewhere as described previously for XSL renderings and sublayouts.

You can follow this procedure to manage all exceptions thrown by the `DoRender()` method of all web controls:

1. Create an abstract base class that inherits from the `Sitecore.Web.UI.WebControl` abstract base class in your Visual Studio project.
2. Implement the `Render()` method in the new class to handle exceptions thrown by calling the `Render()` method in the `Sitecore.Web.UI.WebControl` abstract base class.
3. Update your web controls to inherit from your new base class.

The code shown in Listing 5-7 provides a prototype for such a base class.



LISTING 5-7: Defining a SublayoutRenderingType class prototype

```
namespace SitecoreBook.Web.UI
{
  using System;
  using System.IO;
  using System.Text;
  using System.Web.UI;

  public abstract class WebControl : Sitecore.Web.UI.WebControl
  {
    // disables custom error management for this control
    bool DisableErrorManagement
    {
      get;
      set;
    }

    // cache the output of this control even if an exception occurs
    private bool CacheAfterException
    {
      get;
      set;
    }

    // add exception management to the Render() method in the base class
    protected override void Render(System.Web.UI.HtmlTextWriter output)
    {
      // if instructed to disable error management,
      // call the corresponding method in the base class and exit
      if (this.DisableErrorManagement)
      {
        base.Render(output);
        return;
      }
```

continues

www.it-ebooks.info

LISTING 5-7 *(continued)*

```
    // the output of the web control
    StringBuilder sb = new StringBuilder();
    HtmlTextWriter writer = new HtmlTextWriter(new StringWriter(sb));

    // add exception management around the Render() method in the base class
    try
    {
      base.Render(writer);
      writer.Close();

      // no exception occurred; write the output to the response stream
      output.Write(sb.ToString());
    }
    catch (Exception ex)
    {
      // an exception occurred
      if (ErrorHelper.ShouldRedirect())
      {
        ErrorHelper.Redirect();
        return;
      }

      this.RenderError(ex.Message, ex.GetType().ToString(), ex.ToString(), output);
    }
  }

  public override void RenderError(
    string message,
    string itemPath,
    string details,
    HtmlTextWriter output)
  {
    if (this.CacheAfterException)
    {
      ErrorHelper.RenderError(message, details, output);
    }
    else
    {
      base.RenderError(message, itemPath, details, output);
    }
  }

  protected override string GetCachingID()
  {
    string id = base.GetCachingID();

    if (String.IsNullOrEmpty(id))
    {
      id = this.GetType().ToString();
    }

    return id;
```

www.it-ebooks.info

```
        }
      }
    }
```

In the `SitecoreBook.Web.UI.WebControl` prototype abstract base class for web controls shown in Listing 5-7:

- The `CacheAfterExpiration` property controls whether the layout engine can cache the output generated by this instance of the control.
- The `DisableErrorManagement` property controls whether to apply the custom error management logic to this instance of the control.
- The overridden `Render()` method calls the `Render()` method in the `Sitecore.Web.UI.WebControl` abstract base class and exits if `DisableErrorManagement` is `true`.
- If `DisableErrorManagement` is not `true`, the `Render()` method constructs a temporary `System.Web.UI.HtmlTextWriter`, opens a `try...catch` block, and passes that temporary writer to the existing `Render()` method in the `Sitecore.Web.UI.WebControl` abstract base class, preventing that method from writing directly to the output stream.
- If no exception occurs, the `Render()` method writes the content of the temporary `System.Web.UI.HtmlTextWriter` object to the actual output stream and exits.
- In the case of an exception, the `Render()` method uses the `Sitecore.Web.UI.ErrorHelper` static class shown in Listing 5-1 to interrogate the `mode` attribute of the `/configuration/system.web/customErrors` element in the Web.config file to determine whether to redirect to a friendly error management page.
- If the error helper does not redirect the client, the `Render()` method calls the `RenderError()` method in the `Sitecore.Web.UI.WebControl` abstract base class, which determines whether and how to render information about the exception.
- If the `CacheAfterException` property is `true`, the overridden `RenderError()` method replicates the functionality of the `RenderError()` method in the `Sitecore.Web.UI.WebControl` abstract base class without setting the flag that indicates an error occurred, which would otherwise prevent Sitecore from caching the output of this instance of the control.
- If the `CacheAfterException` property is `false`, the overridden `RenderError()` method calls the `RenderError()` method in the `Sitecore.Web.UI.WebControl` abstract base class, signaling that an error occurred and preventing the layout engine from caching the output of this instance of the control.
- The `GetCachingID()` method returns the value returned by the `GetCachingID()` method in the `Sitecore.Web.UI.WebControl` abstract base class, or the namespace and name of the class if that value is `null` or an empty string. While this method does not relate to the purpose described in this section, I assume that you might want to cache the output of any web control, so you might as well include this implementation in the base class for all web controls. If you know you do not want to cache the output of a web control, do not configure caching for that web control as described in Chapter 6.

www.it-ebooks.info

Depending on which additional methods of the `Sitecore.Web.UI.WebControl` class and its base classes your web controls implement and where you experience or expect to experience exceptions in your solution, you can follow a similar pattern to provide error control at other phases in the ASP.NET lifecycle. For example, similar to what you can do for sublayouts, you can override the `CreateChildControls()` method of the `System.Web.UI.Control` abstract base class for all controls. The `System.Web.UI.Control` class is an eventual base class for the `Sitecore.Web.UI.WebControl` class. The `CreateChildControls()` method is just an example; this method handles only one phase in the ASP.NET page lifecycle that might concern you. Your override can add exception management to any phase(s) of the ASP.NET page lifecycle in all web controls that do not implement methods to handle those phases. When your control experiences an exception, you can set something similar to the `_lastException` variable used in the sublayout example to avoid calling the `DoRender()` method of your web control if an exception occurs in that control earlier in the page lifecycle. You would need to use `try...catch` blocks to trap exceptions in all methods of any web controls that actually implement methods to handle such lifecycle phases.

Because web controls inherit from a base class that contains all the required features, you do not have to add a line to update any layouts or sublayouts, add a line to any configuration file, or update any line to the `/App_Config/Prototypes.config` file. This solution for handling errors in the render phase of the ASP.NET lifecycle applies automatically to all web controls that inherit from the `SitecoreBook.Web.UI.WebControl` abstract base class shown in Listing 5-7, whether you bind those controls statically in code or dynamically using layout details.

Hiding Presentation Control Errors

You can implement an error control based on the prototype shown in Listing 5-8 that hides information about all exceptions unless the user is in the Sitecore browser-based debugger.



Available for download on Wrox.com

LISTING 5-8: Hiding exception information

```
namespace SitecoreBook.Web.UI.WebControls
{
  using System.Web.UI;

  // error control to display exception details only if debugging
  public class HiddenErrorControl : Sitecore.Web.UI.WebControls.ErrorControl
  {
    // clone this error control (implementation required by the contract)
    public override Sitecore.Web.UI.WebControls.ErrorControl Clone()
    {
      return new HiddenErrorControl();
    }

    // render the error
    protected override void DoRender(HtmlTextWriter output)
    {
      // if the user accessed the Sitecore debugger, render error details
      if (Sitecore.Context.PageMode.IsDebugging)
      {
        base.DoRender(output);
```

www.it-ebooks.info

```
        }

        // if the user did not access the debugger, don't render any output
        return;
      }
    }
  }
```

The `SitecoreBook.Web.UI.WebControls.HiddenErrorControl` web control class shown in Listing 5-8 inherits from the `Sitecore.Web.UI.WebControls.ErrorControl` web control class that manages the presentation of error information by default. The `Clone()` method of the `SitecoreBook.Web.UI.WebControls.HiddenErrorControl` class returns an instance of that type. The `DoRender()` method calls the corresponding method in the base class if the debugger is active, or generates no output otherwise.

To use this control, set the `type` attribute of the `/configuration/ErrorControl` element in the `/App_Config/Prototypes.config` file to the signature of your class.

HANDLING EXCEPTIONS AT THE PAGE LEVEL

To implement exception management at the page level, you can use an event handler for error events. If the `AutoEventWireup` attribute in the `Page` directive in the layout (`.aspx` file) is `true`, then all you have to do is name your event handler `Page_Error`. If the `AutoEventWireup` property does not evaluate to `true`, you may have to add the error handler programmatically using a line such as the following in the constructor for the layout or early in the page lifecycle:

```
this.Error += new EventHandler(Page_Error);
```

Within this event handler, you can use the `System.Web.HttpContext.Current.Server.GetLastError()` method to retrieve the last exception, and the `System.Web.HttpContext.Current.Server.ClearLastError()` method to clear the exception, preventing it from reaching higher exception management levels. The example shown in Listing 5-9 does exactly that, logging and clearing all exceptions that inherit from the `SitecoreBook.Exceptions.CustomExceptionBase` base class for exceptions shown in Listing 5-2. This example depends on classes in the `System` and `System.Web` namespaces not shown in the code.



LISTING 5-9: Logging and clearing CustomExceptionBase exceptions

```
// page-level exception handler
public void Page_Error(object sender, EventArgs e)
{
  Sitecore.Diagnostics.Assert.IsNotNull(HttpContext.Current, "context");

  // retrieve the last exception
  Exception exception = HttpContext.Current.Server.GetLastError();
  Sitecore.Diagnostics.Assert.ArgumentNotNull(exception, "exception");

  // handle only exceptions based on CustomExceptionBase
  SitecoreBook.Exceptions.CustomExceptionBase customException =
    exception as SitecoreBook.Exceptions.CustomExceptionBase;
```

continues

www.it-ebooks.info

LISTING 5-9 *(continued)*

```
    // if this exception is based on CustomExceptionBase
    if (customException != null)
    {
      // log and clear the exception
      Sitecore.Diagnostics.Log.Error(customException.GetLogMessage(), this);
      HttpContext.Current.Server.ClearError();
    }
  }
```

The code for the `Page_Error()` method shown in Listing 5-9, intended for use in the code-behind of layouts, traps all exceptions that reach the page level due to no trapping at any lower level. This `Page_Load()` method implementation uses the `System.Web.HttpContext.Current.Server.GetLastError()` method to retrieve the most recent exception. If the exception derives from the `SitecoreBook.Exceptions.CustomExceptionBase` abstract base class for exceptions shown in Listing 5-2, this `Page_Load()` method implementation calls the `GetLogMessage()` method of that exception, writes the result of that method call to the Sitecore log, and then invokes the `System.Web.HttpContext.Current.Server.ClearError()` method to clear that exception. Otherwise, it allows the exception to bubble up to the next layer that can handle exceptions.

This solution will not trap every possible error that could occur while processing a layout, such as if the `Inherits`, `CodeBehind`, or `CodeFile` attributes of the `Page` directive of a layout (`.aspx` file) specify classes that do not exist.

HANDLING EXCEPTIONS AT THE APPLICATION LEVEL

You can use the `global.asax` file in the document root subdirectory of the document root of your Sitecore solution to handle exceptions at the application level, which means all exceptions that you do not handle at a lower level. You can either implement the `Application_Error()` method inline or implement this method in a class that inherits from the `Sitecore.Web.Application` class and update the `Inherits` attribute in the `Application` directive in the default `global.asax` file to specify your class.



For compile-time error detection, avoid inline code in the `global.asax` *file by creating and compiling a class and updating the default* `global.asax` *file to use that class. For instructions, see the "How to Trap Application Exceptions" section of The Sitecore Presentation Component API Cookbook (*`http://bit.ly/r76c6K`*).*

Application exception handlers are similar to page-level exception handlers in the following ways:

- In both cases, you implement an event handler.
- In both cases, you can call the `System.Web.HttpContext.Current.Server.GetLastError()` method to access the exception.
- In both cases, you can call `System.Web.HttpContext.Current.Server.ClearLastError()` method to clear the exception.

www.it-ebooks.info

The primary difference between page-level exception handlers and application-level exception handlers is that page-level exception handlers apply only to requests that activate the `.aspx` file that contains the error handler, whereas application-level exception handlers apply to all requests that IIS processes with ASP.NET.

ERROR MANAGEMENT PAGES

By default, Sitecore redirects the web client to the pages specified by the following settings in the Web.config file under the conditions described:

- `ErrorPage` — General error other than any of those listed here, such as server error.
- `ItemNotFoundUrl` — The item specified in the URL does not exist in the context database.
- `LayoutNotFoundUrl` — The context item contains invalid layout details for the context device.
- `NoAccessUrl` — The context user cannot access the context site or context item.
- `NoLicenseUrl` — The Sitecore license is missing or invalid.

The `RequestErrors.UseServerSideRedirect` setting in the Web.config file controls whether Sitecore invokes the `System.Web.HttpContext.Current.Server.Transfer()` method to implement a server-side redirect rather than using the `System.Web.HttpContext.Current.Response.Redirect()` method that issues a client-side redirect (the default). If you use a client-side redirect, the value in the address bar in the browser changes from the requested URL to the redirect target URL. If you use a server-side redirect, the address bar shows the original requested URL, rather than the target of the redirect.

If application-level exception handlers do not handle an exception, then Sitecore redirects to the page specified by the `ErrorPage` setting in the Web.config file. For most error conditions that do not involve an exception thrown by a presentation component, the `ExecuteRequest` processor in the `httpRequestBegin` pipeline redirects the request to the error page. For more information about pipelines, see Chapter 7. The media request handler specified by the `type` attributes of the `/configuration/system.WebServer/handlers/add` and `/configuration/system.web/handlers/add` elements in the `/web.config` file with `sitecore_media.ashx` for the value of their `path` attributes handles error conditions for media requests separately.

In addition to the settings described previously in this section, when the value of a data template field of type Rich Text contains a link to an item that no longer exists or which the context user cannot access, Sitecore renders that link using the URL specified by the `LinkItemNotFound` setting in the Web.config file.

ASP.NET Error Management Pages

If you do not handle an exception at a lower level or redirect to an error page, ASP.NET applies the configuration defined under the `/configuration/system.web/customErrors` element in the `/web.config` file.

www.it-ebooks.info



Because the `/configuration/system.web/customErrors` *element is not within the* `/configuration/sitecore` *element in the* `/web.config` *file, you must update the actual* `/web.config` *file within the document root of the IIS website rather than using a Web.config include file.*

Sitecore error management features follow the conventions defined by ASP.NET and the configuration defined by this section of the `/web.config` file, and your error management constructs can take this configuration into account as well.

When an exception occurs, depending on system configuration and whether the request originated from the local system (`localhost`) or a remote system, ASP.NET can render detailed information about the exception, including the error message, the exception type, and a stack trace to the client. Although this information can be useful when you are debugging an ASP.NET solution, it could also reveal information that could be used in attempts to compromise the site. The `mode` attribute of the `/configuration/system.web/customErrors` element in the `/web.config` file controls if and when ASP.NET renders such detailed information about exceptions to client browsers on the server and on remote systems.

There are three possible values for the `mode` attribute of the `/configuration/system.web/customErrors` element in the `/web.config` file are as follows:

- `Off` — Always display detailed error information; never redirect to a friendly error page.
- `On` — Always display a friendly error page; never display detailed error information.
- `RemoteOnly` — Display detailed error information if the request originated from the local system (`localhost`); otherwise, display a friendly error page.



To increase system security, never allow a value of `Off` *for the* `mode` *attribute of the* `/configuration/system.web/customErrors` *element in the* `/web.config` *file. Otherwise, when an error occurs, you could expose information that attackers could use in attempts to compromise the system.*

The `/configuration/system.web/customErrors` element in the `/web.config` file can contain any number of `<error>` elements with values for the `statusCode` attribute that match HTTP error code numbers. For example, if a `/configuration/system.web/customErrors/error` element exists with a value of `404` for the `statusCode` attribute, ASP.NET uses that element to handle the HTTP 404 (not found) condition. Specifically, ASP.NET redirects to the URL specified by the `redirect` attribute of that `/configuration/system.web/customErrors/error` element. If no such element exists, ASP.NET redirects to the URL specified by the `defaultRedirect` attribute of the `/configuration/system.web/customErrors` element. In either case, before redirection, ASP.NET adds the `aspxerrorpath` query string parameter to the new URL to indicate the URL of the page that experienced the error.

www.it-ebooks.info

In some cases, Sitecore uses the `mode` attribute of the `/configuration/system.web/customErrors` element defined in the `/web.config` file to determine how to handle errors, mimicking ASP.NET itself. For example, if the `mode` attribute is `Off` or `RemoteOnly`, Sitecore does not handle an exception by redirecting; instead it allows ASP.NET to render information about the exception to the client.

IIS Error Management Pages

If ASP.NET does not handle an exception that neither Sitecore nor any lower-level component handles, IIS handles the exception by redirecting to an error management page. The instructions to configure IIS error pages vary depending on the version of Windows and hence IIS in use. To configure IIS error management pages such as HTTP 404 (not found) and HTTP 500 (server error) conditions in IIS Manager (`inetmgr`) on Windows 7, select a server or a site, and then double-click Error Pages.

HTTP 404 Not Found

When a web server receives a request for a resource that does not exist, it returns the HTTP 404 (not found) status code. In Sitecore, the 404 condition occurs when there is no content item, media item, file, or other handler that matches the requested URL. If no item beneath the context site corresponds to the path in the requested URL but a corresponding file exists beneath the document root, Sitecore returns control to ASP.NET. If neither a corresponding item nor a corresponding file exists beneath the document root, the context item (`Sitecore.Context.Item`) remains `null` and Sitecore handles the 404 condition as described in this section. If Sitecore does not handle an HTTP request and no corresponding file exists beneath the document root, then ASP.NET redirects as specified by the `/configuration/system.web/customErrors` element in the `/web.config` file. If IIS does not handle a request with ASP.NET and no corresponding file exists beneath the document root, then IIS activates its own 404 handler as defined in IIS Manager (`inetmgr`).

The `ItemNotFoundUrl` and `RequestErrors.UseServerSideRedirect` settings in the Web.config file control how the application handles the 404 condition. Under the default configuration, the `RequestErrors.UseServerSideRedirect` setting is `false`, and the `ExecuteRequest` processor at the end of the `httpRequestBegin` pipeline redirects the browser to the URL specified by the `ItemNotFoundUrl` setting in the Web.config file. Under this configuration, the `ItemNotFoundUrl` setting can contain the following:

- The path to a file relative to the document root subdirectory, such as an `.html` or an `.aspx` file
- An external URL including a protocol and domain, such as `http://domain.tld/notfound.html`
- The URL of a Sitecore item relative to the home item of the managed website

In all these cases, Sitecore passes the `item`, `user`, and `site` query string parameters to identify the item path, the context user, and the context site, respectively. Also in all cases, the target page for the redirect should contain a friendly error message or logic to handle the condition, and return the HTTP 404 status code. Be sure to apply this status code change late in the page lifecycle so nothing else overrides this value, and so that ASP.NET and IIS do not detect the status code and handle the 404 condition using their own facilities.

www.it-ebooks.info

By default, the `ItemNotFoundUrl` setting in the Web.config file specifies the path to a file relative to the document root that contains a friendly error message and applies the HTTP 404 (not found) status code to the response. Because all managed sites share a document root, if the `ItemNotFoundUrl` setting contains the path to a file, that error page applies to all managed websites. This is the default configuration.

If the `ItemNotFoundUrl` setting contains the URL of an item relative to the home item of the managed website, then the item that handles the 404 condition depends on the context site. For example, if the `ItemNotFoundUrl` setting is `/notfound.aspx`, then under the default configuration, the `/sitecore/content/home/notfound` item handles the 404 condition for the default managed site named `website`. You would need to create additional items named `notfound` under the home items of any additional managed sites.

Under the default configuration, the `RequestErrors.UseServerSideRedirect` setting is `false`, and Sitecore issues an HTTP 302 (found) response code and redirects the browser to the URL specified by the `ItemNotFoundUrl` setting in the Web.config file. For search engine optimization (SEO) or other purposes, you may prefer to avoid this redirect. Your options include these two approaches:

- Set `RequestErrors.UseServerSideRedirect` to `true`.
- Implement a pipeline processor to set the context item to an item that handles the HTTP 404 (not found) condition when the `context` item is `null`.

If you set `RequestErrors.UseServerSideRedirect` setting in the Web.config file to `false`, Sitecore invokes the `System.Web.HttpContext.Current.Response.Redirect()` method to perform a client-side redirect, which results in a subsequent HTTP request. If you set `RequestErrors.UseServerSideRedirect` to `true`, Sitecore instead invokes the `System.Web.HttpContext.Current.Transfer()` method to perform a server-side redirect, which renders the content of the page specified by the `ItemNotFoundUrl` setting in the Web.config file without a subsequent HTTP request.

By the time the request reaches the `ExecuteRequest` processor in the `httpRequestBegin` pipeline, it is too late to use the `System.Web.HttpContext.Current.Transfer()` method to transfer to the URL of a Sitecore item. If you set `RequestErrors.UseServerSideRedirect` to `true`, then all error pages, including those specified by the `ItemNotFoundUrl`, `ErrorPage`, `LayoutNotFoundUrl`, `NoAccessUrl`, and `NoLicenseUrl` settings in the Web.config file should specify paths to files relative to the document root subdirectory of the Sitecore solution, and Sitecore uses the same error pages for all managed websites. To use Sitecore items to handle errors, allowing different error items for different managed websites, or to use an external URL to handle errors, the `RequestErrors.UseServerSideRedirect` setting cannot be `true`.

To avoid the redirect and still use Sitecore items to handle the 404 condition, you can implement a processor in the `httpRequestBegin` pipeline defined in the Web.config file such that, if the context item is null, sets the context item to an item that handles the 404 condition for the context site. With this solution, you can use a different item to handle the 404 condition for each managed website.

To specify the item to handle the 404 condition for each managed site, you can add a custom attribute to the `/configuration/sitecore/sites/site` elements in the Web.config file — for example, `notFound`. Then you can extend the `Sitecore.Sites.SiteContext` class with a method that

www.it-ebooks.info

returns the item specified by that attribute. You can retrieve the item that handles the 404 condition for the context site easily by calling that method of the `Sitecore.Context.Site` static property that exposes an instance of the `Sitecore.Sites.SiteContext` class that represents the context site. Listing 5-10 defines such an extension method.



LISTING 5-10: Extending the Sitecore.Sites.SiteContext class

```
namespace SitecoreBook.Sites
{
  using System;

  // extension method
  public static class SiteContext
  {
    // the name of the notFound attribute of <site> elements in the web.config file
    public const string NotFoundAttribute = "notFound";

    // retrieve the not found item based on the notFound attribute for the site
    public static Sitecore.Data.Items.Item GetNotFoundItem(
      this Sitecore.Sites.SiteContext me)
    {
      Sitecore.Diagnostics.Assert.IsNotNull(me, "me");
      Sitecore.Diagnostics.Assert.IsNotNull(
        Sitecore.Context.Database,
        "context database");

      // if the context database or the notFound attribute is null for the site, exit
      if (Sitecore.Context.Database == null
        || String.IsNullOrEmpty(me.Properties[NotFoundAttribute]))
      {
        return null;
      }

      // full path to notFound item
      string path = me.StartPath + me.Properties[NotFoundAttribute];

      // retrieve and return the item at that path from the context database
      Sitecore.Data.Database db = Sitecore.Context.Database;
      Sitecore.Data.Items.Item notFound = db.GetItem(path);
      Sitecore.Diagnostics.Assert.IsNotNull(notFound, path);
      bool allowed =
        Sitecore.Security.AccessControl.AuthorizationManager.IsAllowed(
          notFound,
          Sitecore.Security.AccessControl.AccessRight.ItemRead,
          Sitecore.Context.User);
      Sitecore.Diagnostics.Assert.IsTrue(
        allowed,
        "context user access to 404 item");
      return notFound;
    }
  }
}
```

www.it-ebooks.info

The `SitecoreBook.Sites.SiteContext` static class shown in Listing 5-10 defines the `GetNotFoundItem()` extension method for the `Sitecore.Sites.SiteContext` class provided by Sitecore. For more information about extending classes provided by Sitecore with additional methods, see my blog post at `http://bit.ly/ntowNQ`. In this class shown in Listing 5-10:

- The `NotFoundAttribute` constant defines the name of the attribute of `/configuration/sitecore/sites/site` elements in the Web.config file that can contain the path to the item that handles the 404 condition for each managed site.
- The `GetNotFoundItem()` extension method retrieves the attribute named by the `NotFoundAttribute` constant from the `Sitecore.Sites.SiteContext` argument passed to the method implicitly using the first argument (`me`). That argument abstracts the `/configuration/sitecore/sites/site` element in the Web.config file that defines the context site. If no such attribute exists in that `<site>` element, the `GetNotFoundItem()` method returns `null` to indicate that the site does not override Sitecore's default logic for handling the 404 condition.
- If the attribute named by the `NotFoundAttribute` constant exists in that `/configuration/sitecore/sites/site` element, the `GetNotFoundItem()` method constructs the full path to that item. To do so, it uses the `StartPath` property of that `Sitecore.Sites.SiteContext` object, which combines the `rootPath` and `startItem` attributes of that `<site>` element to specify the home item for that managed site.
- The `GetNotFoundItem()` method throws an exception if the item at that path does not exist in the context database or if the context user does not have read access to that item. Otherwise, it returns that item.



You may have noticed that the code in Listing 5-10 explicitly calls the `Sitecore.Security.AccessControl.AuthorizationManager.IsAllowed()` *static method to determine if the context user can access the 404 item for the context site. While the* `GetItem()` *method of the* `Sitecore.Data.Database` *class returns* `null` *if the context user cannot access the item specified as the first argument to this method, because the* `UserResolver` *processor in the* `httpRequestBegin` *pipeline determines the context user, Sitecore cannot establish a proper security context before it invokes that pipeline. Therefore, the security context for the* `httpRequestBegin` *pipeline is undefined, and you can assume it runs in the context of a user with administrative rights. Processors in this pipeline perform explicit security checks as done in Listing 5-10, and since you will call the method in that listing from an* `httpRequestBegin` *pipeline processor or in any methods that you invoke from such processors.*

Add a new pipeline processor to apply the site-specific 404 item after the default `ItemResolver` in the `httpRequestBegin` pipeline defined in the Web.config file. Listing 5-11 provides sample code for such a processor.

www.it-ebooks.info



LISTING 5-11: Defining the NotFoundResolver class

```
namespace SitecoreBook.Pipelines.HttpRequest
{
  using System;
  using System.Web;
  using SitecoreBook.Sites;

  // httpRequestBegin pipeline processor to set the context item to the notFound item
  // associated with the context item if the context item is null after the default
  // ItemResolver
  public class NotFoundResolver : Sitecore.Pipelines.HttpRequest.HttpRequestProcessor
  {
    // httpBeginRequest processor implementation
    public override void Process(Sitecore.Pipelines.HttpRequest.HttpRequestArgs args)
    {
      // if the context item is not null, this is not the 404 condition; exit.
      // if the context site is not known, it is not possible to determine the 404
      // page associated with the context site; exit.
      if (Sitecore.Context.Item != null
        || Sitecore.Context.Site == null)
      {
        return;
      }

      // if the site specifies a virtual folder, exit
      if ((!String.IsNullOrEmpty(Sitecore.Context.Site.VirtualFolder))
        && Sitecore.Context.Site.VirtualFolder != "/")
      {
        return;
      }

      // determine if the request maps to a file path rather than an item
      string filePath = Sitecore.Context.Request.FilePath.ToLower();

      // if the request maps to a file path rather than an item, exit
      if (String.IsNullOrEmpty(filePath)
        || Sitecore.Web.WebUtil.IsExternalUrl(filePath)
        || System.IO.File.Exists(HttpContext.Current.Server.MapPath(filePath)))
      {
        return;
      }

      // set the context item to the notFound item associated with the context site
      Sitecore.Context.Item = Sitecore.Context.Site.GetNotFoundItem();
    }
  }
}
```

The `SitecoreBook.Pipelines.HttpRequest.NotFoundResolver` class shown in Listing 5-11 defines the `Process()` method containing the following logic:

- If the context item is not null, the current request did not trigger the 404 condition. If the context site is null, the processor cannot determine the 404 item for the context site. In

www.it-ebooks.info

either of these cases, `Process()` method exits immediately and the `httpRequestBegin` pipeline continues.



For optimal performance, exit processors as soon as you determine that the processing context is irrelevant.

- This processor does not handle managed sites that define the `virtualFolder` attribute as anything other than slash. In such cases, the `Process()` method exits immediately and the `httpRequestBegin` pipeline continues.
- If the requested URL is empty, indicating a request for the home page of the context site, or if that URL is an external page, or maps to a file on the file system, the processor cannot handle that condition and the `Process()` method exits immediately and the `httpRequestBegin` pipeline continues.
- If none of the conditions described previously apply, the processor sets the context item to the item that handles the 404 condition for the context site using the `GetNotFoundItem()` extension method for the `Sitecore.Sites.SiteContext` class defined in Listing 5-10. If the definition for the context site does not define the `notFound` attribute evaluated by that method, then the context item continues to be null, and the `httpRequestBegin` pipeline applies its default logic for handling the 404 condition by redirecting to the URL specified by the `ItemNotFoundUrl` setting in the Web.config file.

For items that handle the 404 condition, you can statically or dynamically bind a presentation control to a placeholder in the layout or a nested sublayout to set the HTTP status code for the response to 404. The example in Listing 5-12 provides a web control that generates no output, but sets the HTTP status code of the response to 404.



While it would be convenient to set the status code in the `httpRequestBegin` pipeline processor shown in Listing 5-11, that pipeline runs too early in the ASP.NET page lifecycle for this to work. If you set the response status code in that processor, IIS or ASP.NET handles the 404 condition with their own logic, preventing the layout engine from applying layout details defined in the item you specify to handle the 404 condition.



Available for download on Wrox.com

LISTING 5-12: A presentation control to set the HTTP status in the response to 404

```
namespace SitecoreBook.Web.UI.WebControls
{
  using System.Web.UI;

  // web control to set the HTTP status code to 404 (not found)
  public class SetNotFoundStatus : Sitecore.Web.UI.WebControl
```

www.it-ebooks.info

```
    {
      protected override void OnInit(System.EventArgs e)
      {
        this.Page.Response.StatusCode = 404;
        base.OnInit(e);
      }

      // required by the Sitecore.Web.UI.WebControl contract
      protected override void DoRender(HtmlTextWriter output)
      {
        // this method intentionally generates no output
      }
    }
  }
```

The `SitecoreBook.Web.UI.WebControls.SetNotFoundStatus` web control class shown in Listing 5-12 implements the `Init()` method to set the HTTP response status code to 404 to indicate that no resource exists at the requested URL. While the `DoRender()` method implementation shown in Listing 5-12 generates no output, if you implement a control based on this class that does generate output, you can cache the output of that control as described in Chapter 6. In your control, implement the `GetCachingID()` method required to support output caching or inherit from a class based on the `SitecoreBook.Web.UI.WebControl` abstract base class shown in Listing 5-3 that implements the `GetCachingID()` method for you. When you cache the output of a presentation control, ASP.NET invokes all page lifecycle events for the control. If the layout engine has cached the output of your control, the `Render()` method `Sitecore.Web.UI.WebControl` abstract base class from which your control eventually derives, which handles the rendering lifecycle event, does not call the `DoRender()` method of your control, but the ASP.NET framework invokes other lifecycle methods including `Init()`.



Although you should not cache the output of this web control and this control is unlikely to experience exceptions at any phase in the ASP.NET page lifecycle, the `SitecoreBook.Web.UI.WebControls.SetNotFoundStatus` *web control could inherit from the* `SitecoreBook.Web.UI.WebControl` *prototype abstract base class shown in Listing 5-7 instead of the* `Sitecore.Web.UI.WebControl` *class as shown in Listing 5-12.*

Finally, create items to handle the 404 condition under the home item of each managed site for which you want to apply this custom 404 management logic, and set the `notFound` attribute of the corresponding `/configuration/sitecore/sites/site` elements in the Web.config file to the paths to those items relative to the home items of those managed sites. For example, set the `notFound` attribute to `/notfound` (with the leading slash but without the `.aspx` extension) to trigger the child of the home item named `notfound`. If you do not set the `notFound` attribute for one of the managed sites, Sitecore redirects to the URL specified by the `ItemNotFoundUrl` setting in the Web.config file for 404 requests associated with that managed site, which results in a client or server redirect depending on the value of the `RequestErrors.UseServerSideRedirect` setting in the Web.config file.

www.it-ebooks.info

Some apparent error conditions, specifically the HTTP 404 (not found) condition, do not always indicate errors. You can sometimes use the 404 management facilities of IIS and ASP.NET to your advantage. For example, IIS versions prior to 7 that added the integrated mode for application pools, IIS did not use ASP.NET to handle all HTTP requests, but used ASP.NET only for URLs with certain extensions such as `.aspx`. You could set the IIS 404 page to a URL such as `/default.aspx` to trick IIS into triggering ASP.NET for all requests that did not map to files within the document root. For another example `httpRequestBegin` processor, see the `Sitecore.Sharedsource.Pipelines.HttpRequest.NotFoundSearchResolver` processor in the PageNotFound (`http://bit.ly/yyGqrh`) Sitecore Shared Source project, which attempts to determine search keywords from the requested URL.



For more information about handling the HTTP 404 condition with Sitecore, see The Sitecore Guide to Handling HTTP 404 (`http://bit.ly/nFMNTd`*).*

SYSTEM OUTAGES

Sitecore depends on the underlying platform, which includes Microsoft Windows, IIS, ASP.NET, supporting relational databases, and of course the underlying hardware, including network facilities. If any of these elements does not function properly, Sitecore cannot function. If the databases are unavailable, Sitecore is unavailable. Although certain Sitecore features, such as agents configured in the Web.config file, continue to run while ASP.NET is running, they fail if they attempt to access a Sitecore database or various Sitecore APIs. Research and implement best practices and tools for handling scheduled and unscheduled application outages on the ASP.NET platform, including external monitoring software and services.

TAKE HOME POINTS

Error management is critical to the quality of a Sitecore solution. Follow best practices for managing exceptions with ASP.NET, including implementation of your own exception classes. Use similar logic and facilities to trap exceptions at all levels and in all presentation component types, including XSL renderings, web controls, layouts, and sublayouts. To make this easier, implement a helper class that you can use from any component and configure error controls for all types of components. Consider different exception management approaches depending on whether the user accesses the published site, the Sitecore browser-based debugger, or another CMS user interface. Evaluate the `/configuration/system.web/customErrors` section of the Web.config file to determine how to handle errors.

Begin by implementing your own exception types and trapping exceptions that you expect using `try...catch...finally` blocks. Then, configure error management for XSL renderings, sublayouts, and web controls. You can also trap exceptions at the application level; and, finally, you should consider the error pages that Sitecore, ASP.NET, and IIS apply under various conditions. You can implement custom logic to determine the item to handle the HTTP 404 condition for each managed site, and you can otherwise handle the HTTP 404 condition to your advantage or the visitor's.

www.it-ebooks.info

6

Optimizing, Scaling, and Troubleshooting

WHAT'S IN THIS CHAPTER?

- Capitalizing on caching
- Scaling your solution
- Virtualizing your instances
- Edge caching and content delivery networks
- Architecting for high availability
- Planning hardware capacity
- Monitoring and improving performance
- Diagnosing problems and debugging
- Understanding the web.config file
- Disassembling .NET assemblies

This chapter contains invaluable information that you can use to scale, troubleshoot, and optimize the performance of your Sitecore solution. Beginning with a detailed description of the layered Sitecore caching architecture, it continues to explain how you can monitor and tune the size of each cache.

The next section introduces concepts that you can use to scale your Sitecore solution to meet high visitor traffic loads in the content delivery environment and high concurrent user loads in the content management environment, which also increases the reliability of that solution. This section touches on virtualization, edge caching, and content delivery networks (CDNs).

www.it-ebooks.info

Next, this chapter provides techniques that you can use to troubleshoot your Sitecore solution, such as monitoring the contents of the Sitecore log files, configuring performance thresholds, and debugging. The final section contains information about options for furthering your understanding of the product, such as by investigating the root `/web.config` file, Web.config include files, stack traces, and how to reflect upon and disassemble .NET assemblies (`.dll` files) that ship with the product to review an approximation of the source code used to compile those assemblies.

OPTIMIZING SITECORE PERFORMANCE

This section explains how to maximize the performance of a Sitecore solution and covers a variety of subjects to help you wring every microsecond of processing time out of your Sitecore servers. Optimal performance directly improves viewers' impressions of a website, and indirectly their impressions of the organization hosting that website. Just as important, optimal performance provides a given hardware configuration with greater capacity to service requests, especially under heavy loads. While some of the techniques described here are specific to Sitecore, others apply to the underlying technology stack, including the Microsoft Windows operating system, the Internet Information Services (IIS) web server, and the ASP.NET web application layer.



While this book covers the most important topics on the subject, space does not permit a comprehensive guide to optimizing the performance of the technology platform. For information about optimizing these technologies, see Improving .NET Application Performance and Scalability *(`http://bit.ly/xMTm7r`; ISBN 978-8120326989) by Microsoft Press.*

Leveraging Sitecore Caching

One of the most important aspects affecting the performance of a Sitecore solution involves the layers of caching inherently provided by the platform. As with any high-performance application, caching strategies and cache tuning have a significant impact on the performance of a website. In fact, one of the major advantages of using Sitecore relative to alternative solutions such as building your own content management system is the implicit caching mechanisms provided by the platform.

Sitecore engineering has designed its caching architecture carefully and tested its caching strategies and default configuration thoroughly to provide the greatest benefit for the most common types of solutions without excessive memory consumption. At the same time, you can increase, decrease, and disable the various caches to tune the platform for your specific requirements. Regardless of whether you use the default configuration or refine Sitecore caching precisely, when it comes to tuning performance, knowledge of the Sitecore caches and their interactions can be very important. For example, if you know that Sitecore caches a specific type of data, and how Sitecore clears that cache, to avoid redundant caching that can only adversely affect memory consumption, avoid caching that data in your application.

www.it-ebooks.info

For more information about caching with Sitecore beyond that provided in the following sections, see The Sitecore Cache Configuration Reference (`http://bit.ly/qYIAia`).

Utility Caches

Sitecore provides a number of caches for different purposes. In general, other than defining appropriate size limits as described later in this chapter, you can ignore the utility caches included here only for informative purposes. Sitecore provides most of these caches to improve the performance of Sitecore CMS user interfaces such as the Content Editor. Such utility caches include the following:

- **Client Data Store cache —** This cache stores information about authenticated users, including user names and properties.
- **Security caches —** The IsUserInRole, UserProfile, and AccessResult (security) caches store information about whether users are members of roles, user profiles, and access rights for items.
- **Viewstate cache —** This cache stores information about the state of ASP.NET controls.
- **Registry cache —** Sitecore user interfaces use the registry cache for various purposes, such as to store the ID of the item selected by each user in the Content Editor.
- **XSL caches —** Sitecore associates an eXtensible Stylesheet Language (XSL) cache with each managed site to store the transformation .NET objects used to implement XSL renderings.



The XSL cache does not contain the output of XSL transformations, but the objects used to implement those transformations. The output cache for each managed site described in a subsequent section of this chapter can contain cached output from XSL renderings. If your solution depends on XSL renderings, and especially if it uses XSL renderings extensively, then you should not ignore the XSL cache. In that case, the XSL cache can significantly improve performance by minimizing the number of load and process operations to build transformation objects for XSL renderings.

This is not a comprehensive list of all utility caches — only those that may interest typical Sitecore developers. You can generally ignore any cache not specifically mentioned in this chapter. This book does not touch upon these caches beyond the information provided here. For a complete list of the caches in use on a Sitecore instance, access the cache administration page on that instance (a subsequent section of this chapter provides more information about this tool).

Database Caches

To minimize round trips to the database, Sitecore caches data about items in each database at a number of layers. These database caches can drastically affect the performance of a Sitecore solution, both at startup and throughout the lifecycle of the application. Sitecore automatically maintains these database caches for each database throughout the lifetime of the application.

www.it-ebooks.info



Restarting the application, such as by intentionally or unintentionally cycling the application pool, clears all Sitecore caches other than the media caches described in a subsequent section of this chapter.

Sitecore data providers abstract implementation details about underlying data stores to provide a universal architecture and application programming interface (API) for the data retrieved from each. Database caches technically cache data from data providers, not directly from the associated databases.

Prefetch Caches

Prefetch caches contain information about items that the system typically accesses during or immediately subsequent to starting the CMS. Sitecore loads the prefetch caches during application initialization or immediately thereafter. Prefetch caches optimize performance immediately subsequent to application startup, but loading too much data into a prefetch cache can adversely affect the time required for the application to initialize. Prefetch caches provide the first caching layer between the application and the database. Each prefetch cache contains data from a single data provider, such as the data provider associated with each Sitecore database.

Each entry in a prefetch cache contains data for a single item returned by the data provider associated with the cache. Each such entry contains field values for all versions of the item in all languages available from that provider. Because the Master database can contain multiple versions for each language, the size of the prefetch cache typically exceeds that of the corresponding caches for each publishing target database, which contain only one version in each language for each item. Entries in these caches can contain information about items related to that item, such as its parent and children.

The `.config` files in the `/App_Config/Prefetch` subdirectory of the document root of the IIS website control the prefetch caches. The names of these files match the names of the corresponding database; for example, the `/App_Config/Prefetch/Core.config` file controls the prefetch cache for the Core database. Sitecore merges the information in these files with global prefetch cache configuration definitions in the `/App_Config/Prefetch/Common.config` file. Together, these configuration files limit the size of the prefetch caches, control which items to load into each of these caches, and dictate whether to include information about the children of items in each of these caches.

The default `/App_Config/Prefetch/Web.config` file, which defines prefetch cache settings for the default publishing target database named Web, contains the following XML by default:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <cacheSize>10MB</cacheSize>
  <template desc="alias">{54BCFFB7-8F46-4948-AE74-DA5B6B5AFA86}</template>
  <template desc="layout">{3A45A723-64EE-4919-9D41-02FD40FD1466}</template>
  <template
    desc="xsl rendering">{F1F1D639-4F54-40C2-8BE0-81266B392CEB}</template>
  <item desc="home">{110D559F-DEA5-42EA-9C1C-8A5DF7E70EF9}</item>
  <children desc="main items">{110D559F-DEA5-42EA-9C1C-8A5DF7E70EF9}</children>
</configuration>
```

www.it-ebooks.info



The `/App_Config/Prefetch/Web.config` *file is unrelated to the* `/web.config` *file in the document root of the IIS website hosting the solution. Sitecore does not apply files in the* `/App_Config/Prefetch` *subdirectory as Web.config include files automatically;* `<sc.include>` *elements in the* `/web.config` *file include these files explicitly for each database. If you add publishing target databases, you should configure prefetch caching for that database, either directly in the* `/web.config` *file or a Web.config include file, or using one or more* `<sc.include>` *elements in that database definition to include specific prefetch cache configuration files.*

The fundamental elements in a prefetch cache configuration file are as follows:

- `<cacheSize>` — The `<cacheSize>` element defines the memory size limit for the prefetch cache for this database. If you change configuration of a prefetch cache, you should almost certainly tune the cache size accordingly. A prefetch cache size of 100 MB to 300 MB is not uncommon for a publishing target database.
- `<template>` — Each `<template>` element specifies the ID of a data template. At application initialization, Sitecore loads all items based on this data template into the prefetch cache for this database. The prefetch caches should typically include the data templates for the types of pages accessed most frequently in your solution, such as the data template for top-level sections of the managed sites, without causing Sitecore to load a very large number of items into the cache. The default `<template>` elements shown previously for the Web database contain the IDs of the data templates for alias definition items and layout definition items.
- `<item>` — Each `<item>` element specifies the ID of specific items to load into the prefetch cache for this database at application initialization. The only `<item>` element in the default `/App_Config/Prefetch/Web.config` file specifies the ID of the `/sitecore/content/home` item, which is the home page of the default managed site named website. If you change the data template associated with the `/sitecore/content/home` item, or update configuration to use a home item based on a different data template for any of your managed sites, you should update the default `<item>` element or add a corresponding `<item>` element accordingly.
- `<children>` — Each `<children>` element specifies the ID of an item whose children should be loaded into the prefetch cache at application initialization for this database. The default `<children>` element contains the ID of the `/sitecore/home/item`; Sitecore loads all children of that item into the prefetch cache at application initialization.

To increase performance, you can tune the prefetch caches to include the items most commonly requested for your solution. For example, if a number of functional components depend on some specific items to determine their intended behavior, you can update your configuration to include those items in the prefetch caches. You may also benefit from storing structured data that you represent as items, such as metadata hierarchies and lists of keywords, in the prefetch caches.

www.it-ebooks.info



If your CMS users complain of slow performance, you may want to adjust configuration of the prefetch cache for the Master database (in the `/App_Config/Prefetch/Master.config` *file) to include the items accessed most frequently in the content management environment. You can use the cache statistics page described in a subsequent section of this chapter to tune caches.*

Data Caches

Data caches contain information from all the data providers defined for a database. Under the default configuration, each representation of a database depends on a single data provider that connects to an actual database, but it is possible for such a database representation to retrieve data from multiple providers. For example, you could implement a data provider that retrieves information from an external product information management (PIM) or order management system and represents that data as if it were native to the Master database. The data cache could contain data retrieved from all the providers defined for a representation of a database.

Data caches retrieve data from database prefetch caches where such caches exist, and therefore contain the same information but in a different format. When Sitecore purges data from a prefetch cache, it also purges corresponding data from the data cache that depends on that prefetch cache. Not all data providers implement prefetch caching; database data caches are especially important for such providers.

Similar to prefetch caches, entries in database data caches represent individual items, and each entry contains field values for all versions in all languages available for an item, as well as the relations between that item and other items. Unlike prefetch caches, Sitecore does not prepopulate data caches at application initialization, but rather as the application (specifically, the item caching layer) demands data from the data provider. Data caches use .NET classes that provide extremely fast access to maximize performance while protecting data providers, which typically abstract database servers, from constant requests.

After your solution is up and running, performance can often benefit from data cache tuning as described later in this chapter. Specifically, when monitoring cache utilization indicates that a data cache has reached its size limit, assuming sufficient available memory, you should increase the size limit for that cache.

To specify a default size limit for all database data caches, set the `Caching.DefaultDataCacheSize` setting in the Web.config file. To specify a size limit for the data cache for an individual database, set the value of the `/configuration/sitecore/database/data/cacheSizes/data` element in the Web.config file, where the value of the `id` attribute of the `/configuration/sitecore/database` element containing that `<data>` element matches the logical name of the database.

Item Caches

Each entry in a database item cache represents an individual version of an item in a single language using the well-known `Sitecore.Data.Items.Item` class. Most Sitecore developers access items using this class, implicitly accessing an item cache rather than accessing a database directly. Therefore, automatic caching of objects using the `Sitecore.Data.Items.Item` type with a well-populated item cache can really make a Sitecore solution fly.

www.it-ebooks.info

Item caches retrieve data from data caches. When Sitecore purges data from a data cache, it also purges corresponding data from the item cache that depends on that data cache. Sitecore does not populate item caches at application initialization, but rather as the application demands data.

To specify a default size limit for all database item caches, set the `Caching.DefaultDataCacheSize` setting in the Web.config file. To specify a size limit for the item cache for an individual database, set the value of the `/configuration/sitecore/database/data/cacheSizes/item` element in the Web.config file, where the value of the `id` attribute of the `/configuration/sitecore/database` element containing that `<item>` element matches the logical name of the database. To disable all database item caches, perhaps because you suspect item caching as the cause of an issue you are troubleshooting, you can set the `Caching.ItemCachingEnabled` setting in the Web.config file to `false`.



Disabling item caching has a significant negative impact on performance.

Sitecore does not track the exact size of item caches; it approximates the amount of memory consumed to determine whether each cache has reached its size limit. To estimate the size of each item cache, Sitecore multiplies factors, including the number of entries in the cache, the value specified by the `Caching.AverageItemSize` setting in the Web.config file, and the load factor initially defined by the `Caching.LoadFactor` setting in the Web.config file. The `Caching.AverageItemSize` setting in the Web.config provides an estimate of the average size of each entry in the item cache. The `Caching.LoadFactor` setting in the Web.config file is simply a factor that Sitecore uses to adjust cache size estimates. Before the number of entries in the cache exceeds the configured size limit for the cache divided by the product of `Caching.AverageItemSize` and the load factor, Sitecore begins to evict entries from the cache. Increasing the value of either of these two settings reduces the number of items Sitecore can store in the cache before reaching its size limit. Your solution may benefit from an accurate `Caching.AverageItemSize` estimate, which depends on your unique solution. For more information about these settings, see The Sitecore Cache Configuration Reference (`http://bit.ly/qYIAia`) and the comments above these settings in the Web.config file.



The `Caching.LoadFactor` *setting affects Sitecore's size estimates for all caches stored in memory, but the* `Caching.AverageItemSize` *setting affects only the item caches.*

Item Path Caches

Sitecore uses database item path caches to map item paths to the IDs of the corresponding items in each database. Item paths often appear in URLs, and sometimes in code, configuration, and elsewhere, but you can always achieve a slight performance advantage by accessing items using IDs rather than paths. Sitecore does not populate database path caches at application initialization, but as the application demands data.

To specify a default size limit for all database path caches, set the `Caching.DefaultPathCacheSize` setting in the Web.config file. To specify a size limit for the item path cache for an individual database, set the value of the `/configuration/sitecore/database/data/cacheSizes/path` element

www.it-ebooks.info

in the Web.config file, where the value of the `id` attribute of the `/configuration/sitecore/database` element containing that `<path>` element matches the logical name of the database.

When you use the Sitecore API to request an item using its path, Sitecore uses an item path cache to map that path to an ID and then uses that ID to retrieve data from the underlying cache layers. While you always gain some slight performance edge by using the ID of an item rather than its path, your solution can also benefit from using the path cache when you do not know the ID of an item. For example, to access a known child of an item, you can use an approach such as the following:

```
private void Sitecore.Data.Items.Item GetKnownChild(
  Sitecore.Data.Database database,
  string childName)
{
  Sitecore.Data.Items.Item knownItem =
    database.GetItem("/sitecore/content/knownitem"];

  if (knownItem != null)
  {
    return knownItem.Children[childName];
  }

  return null;
}
```

A slightly more efficient, and arguably more maintainable, implementation would leverage the path cache as shown here:

```
private void Sitecore.Data.Items.Item GetKnownChild(
  Sitecore.Data.Database database,
  string childName)
{
  return database.GetItem("/sitecore/content/knownitem/" + childName]);
}
```

These two examples achieve the exact same result, but the former accesses the children of the known item, whereas the latter accesses an individual child item directly using its path. The second version can therefore utilize the path cache during subsequent reads of children under the same known item, which is more efficient than accessing its children. While some readers may intuit this relative advantage, others might assume a performance advantage from using the known item that Sitecore caches automatically, and potentially its cached children as well (in reality, Sitecore may confirm the children each time you access them). Together, minor optimizations like this can add up to dramatically increased performance.

Site Media Caches

Sitecore maintains a media cache for each managed site. Media caches contain metadata about media items accessed in the context of that site, as well as a cached copy of the binary media itself. Sitecore caches the binary data after media manipulation, such as server-side image resizing. If you access an image media item in multiple dimensions, the media cache contains multiple cache entries for that image. Unlike most Sitecore caches that store data in random access memory (RAM), due to the volume and persistence of media data, Sitecore manages media caches within a filesystem.

By default, Sitecore manages site media caches in the subdirectory specified by the `Media.CacheFolder` setting in the Web.config file. For each managed site, Sitecore creates a subdirectory

www.it-ebooks.info

named after the site within the subdirectory indicated by this setting. Under the default configuration, the `Media.CacheFolder` setting is `/App_Data/MediaCache`. Under this configuration, Sitecore caches media for the default site named `website` to the `/App_Data/MediaCache/website` subdirectory of the document root of the IIS website. Sitecore creates these subdirectories automatically when needed.



Storing site media caches under `/App_Data` *prevents IIS from serving cached media files directly. While it is unlikely that an attacker could determine the URL of a cached media file, moving the media cache to a subdirectory that IIS does not protect could allow attackers to access cached media directly, increasing the attack surface and thereby reducing security.*

To specify the location of the media cache subdirectory for an individual managed site, you can set the `mediaCachePath` attribute of the `/configuration/sitecore/sites/site` element in the Web.config file that defines properties of that site to the path to that subdirectory. To disable media caching for a managed site, you can set the `cacheMedia` attribute of that `/configuration/sitecore/sites/site` element in the Web.config file to `false`. To disable media caching entirely, set the `Media.CachingEnabled` setting in the Web.config file to `false`.

Sitecore does not populate media caches at application initialization, but as clients request media resources. Unlike other Sitecore caches, which store data in memory, because the media cache uses a filesystem, it can contain entries before the application initializes. Sitecore does not limit the size of media caches. Sitecore uses a scheduled agent to expire old entries from media caches.



If you specify an alternate subdirectory for a media cache, update elements within the `Sitecore.Tasks.CleanupAgent /configuration/sitecore/scheduling/agent` *element in the Web.config file to ensure that the process to remove old entries from the media cache includes that subdirectory.*

Site Output Caches

Sitecore maintains a site output cache for each managed site. These output caches (also called HTML caches) contain the results of invoking renderings (typically markup fragments) under various conditions as described later in this section. Unlike all Sitecore caches described previously, which Sitecore populates automatically, you must explicitly configure renderings to populate output caches. Site output caching is typically the most important aspect of caching to monitor, as this cache can greatly increase solution performance by reducing central processing unit (CPU) load.

By default, Sitecore does not cache the output of entire pages. Instead, Sitecore caches the output of individual renderings. You can reuse the cached output of a rendering on any number of content items (pages), or Sitecore can invoke that rendering each time it encounters differing processing conditions as described further in this section. Unless you direct Sitecore to cache the output of a rendering (or sublayout), Sitecore invokes that rendering each time it applies layout details to an item that include that rendering, which is always more resource-intensive than retrieving cached output for that component.

www.it-ebooks.info



In addition to application initialization, under the default and recommended configuration, publishing clears the entire output caches for the managed sites.

For the published site to benefit from output caching, you should configure output caching for each presentation component. Sitecore does not cache the output of a rendering unless you instruct it to do so and optionally define the conditions under which to generate multiple entries in the output cache for that rendering. To minimize both the number of times Sitecore invokes each rendering and the amount of memory used by the output cache, cache the output of each rendering by the fewest possible criteria described in the following paragraphs.

While you can configure default (implicit) caching options for a rendering in the Caching section of the definition item associated with the rendering, I prefer to explicitly define caching options for each use of each rendering. To configure caching options in the layout details of the standard values for a data template or an individual item, follow these steps:

1. Select the item in the Content Editor. For instructions on how to access the Content Editor, see Chapter 1.
2. Select the Presentation tab, and then click Details in the Layout group. The Layout Details dialog appears as shown in Figure 6-1.

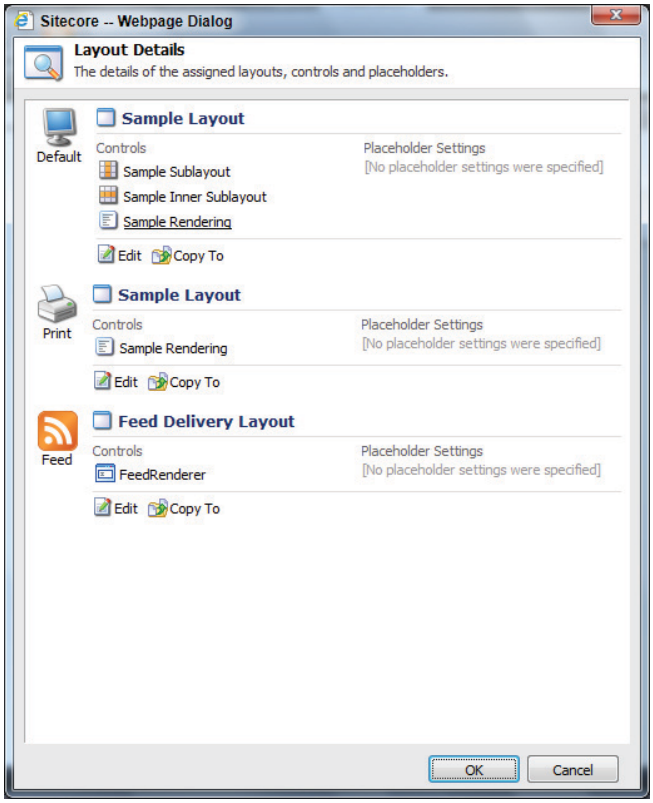


FIGURE 6-1

www.it-ebooks.info

3. Click the rendering for which to configure caching options. The Control Properties dialog appears as shown in Figure 6-2.

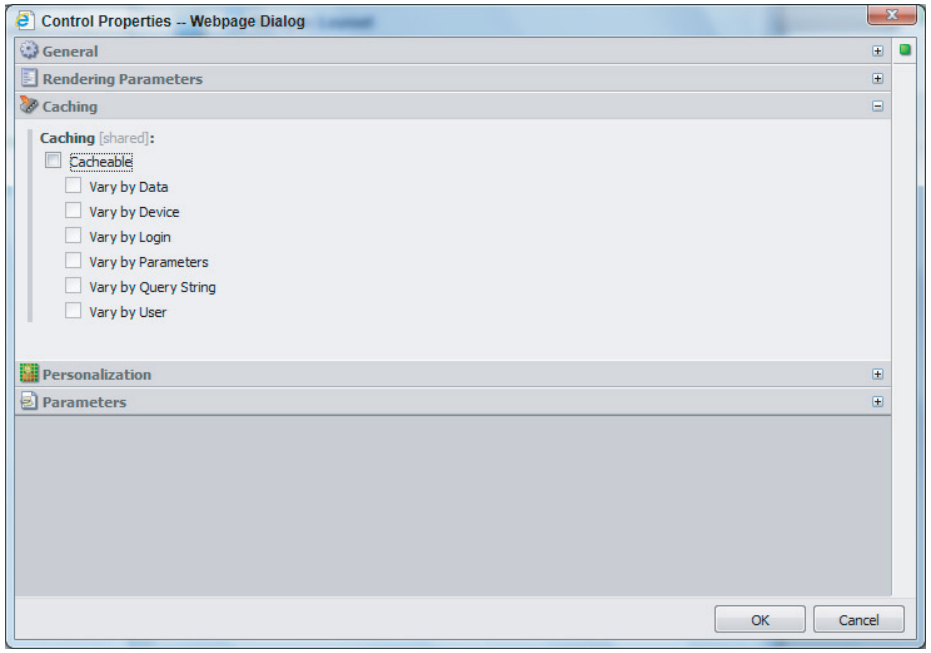


FIGURE 6-2

4. To enable output caching for the rendering, select the Cacheable checkbox. Until you select the Cacheable checkbox, Sitecore will not cache the output of the rendering, and you cannot select additional caching options (the Vary By options).
5. Click OK. The Control Properties dialog disappears and you return to the Layout Details dialog.
6. Click OK again. The Layout Details dialog disappears and you return to the Content Editor.



You should also define caching options where you bind presentation controls statically and programmatically. To define caching options for presentation controls that you bind statically, set the `Cacheable` *attribute and the appropriate* `VaryBy` *attributes of the control to* `true`*. To define caching options for presentation controls that you bind programmatically, set the* `Cacheable` *property and the* `VaryBy` *properties of the control to* `true` *before reaching the rendering phase of the ASP.NET lifecycle.*

Because Sitecore maintains an output cache for each managed site, output caching automatically varies by site. In other words, if you select the Cacheable checkbox and then invoke the rendering in

www.it-ebooks.info

the context of two different managed sites, Sitecore invokes the rendering twice and generates two cache entries, one in the output cache of each site.

To add the output generated by a rendering to an output cache, Sitecore generates a cache key that uniquely identifies the criteria under which it generated that output. If you have enabled output caching for a rendering, before invoking that rendering, Sitecore constructs a cache key for that rendering based on the Vary By options selected for the rendering and the current processing conditions. If that key already exists in the output cache, the layout engine retrieves that output and embeds it in the current page; otherwise, the layout engine invokes the rendering, embeds that output in the HTTP response stream, and adds that output to the output cache for the context site using that cache key.

Each output cache works much like a large hash table, where the cache key is the key and the output generated by the rendering is the value. The cache key automatically includes a code that identifies the rendering, which is the value returned by the `GetCachingID()` method of the web control used to implement that rendering. For XSL renderings and sublayouts, the `GetCachingID()` method returns the path to the `.xslt` or `.ascx` file relative to the document root of the IIS website hosting the Sitecore solution. Because the output of a rendering usually depends on the context language, keys for entries in the output cache automatically contain the name of the context language, so Sitecore automatically varies by the language.



Note the distinction between the cache ID, which identifies the presentation control, and the cache key, which identifies the presentation control, the Vary By caching options selected for that control, and current values of those options.

After you select the Cacheable checkbox in the Layout Details dialog as described previously, you can select additional checkboxes to vary by additional criteria, which effectively adds those criteria to the cache key. Because the output of most renderings depends on the data source of the rendering, you are most likely to select the Vary by Data checkbox, which causes the key to contain the ID of the data source item for the rendering. You can select the Vary by Device checkbox to include the name of the device in the cache key. Selecting Vary by Login causes the cache key to contain a flag that indicates whether the user authenticated; you can use this option for renderings that generate different output based on whether the user is anonymous or known. If you select Vary by User, the cache key includes the login name of the context user, which means that the rendering can generate and cache different output depending on the specific user who has authenticated.



Because it results in cache entries specific to each user, selecting the Vary by User option can increase memory utilization, especially on systems with large numbers of concurrent users. Apply this option with caution.

The Vary by Parameters option results in a cache key that contains any key/value parameters passed to the rendering. The Vary by Query String option causes the cache key to contain any query string parameters in the URL for which Sitecore invoked the rendering.

www.it-ebooks.info

When you publish changes from the Master database to a publishing target database, on the local instance Sitecore clears the output cache for the managed sites listed under the `/configuration/sitecore/events/event/handler` element in the Web.config file for the `<event>` element named `publish:end`. On remote content delivery instances, publishing clears the output caches for the sites listed for the `publish:end:remote` event. For more information about remote content delivery instances, see The Sitecore Scaling Guide (`http://bit.ly/oKvYuN`).



Publishing clears all entries from the output caches. To make the best use of entries in output caches, minimize publishing frequency.

The Sitecore browser-based debugger disables output caching by default, causing Sitecore to invoke each rendering on each page view regardless of processing conditions. This can help in diagnosing issues with renderings, but it can prevent you from diagnosing issues with output caching. To enable output caching in the Sitecore debugger, clear the Information checkbox in the Rendering group on the Ribbon. Alternatively, add the `sc_ri` query string parameter with a value of `0`.

Caching the output of a rendering prevents Sitecore from invoking rendering operations in that rendering, but does not prevent Sitecore from invoking operations during other phases in the ASP.NET page lifecycle. In general, activity during those other phases has no impact if the system has already cached the output of the rendering, but this functionality may be valuable under certain circumstances. As a contrived example, consider a page hit counter that does not display the hit count. After the first invocation, this rendering always renders cached output under equivalent processing conditions, but it can increment the counter at a step in the page lifecycle not related to rendering. Consider this when optimizing your rendering; at other stages in the page lifecycle, you may wish to exit the method if Sitecore has already cached the output of that rendering under equivalent processing conditions. The `SitecoreBook.Web.UI.WebControls.PerformsExpensiveInitialization` web control shown in Listing 6-1 implements this technique for the initialization phase of the lifecycle.



Available for download on Wrox.com

LISTING 6-1: PerformsExpensiveInitialization.cs

```
namespace SitecoreBook.Web.UI.WebControls
{
  using System;
  using System.Web.UI;

  public class PerformsExpensiveInitialization : Sitecore.Web.UI.WebControl
  {
    public bool HasBeenCached()
    {
      // used to identify an entry in the output cache
      // for this control under equivalent processing conditions
      string cacheKey = this.GetCacheKey();

      // if the layout engine will not cache the output
      // of this instance of the control
      if (String.IsNullOrEmpty(cacheKey))
```

www.it-ebooks.info

```
        {
          return false;
        }

        // if the context site defines an output cache
        // and that cache already contains the output of this control
        // under equivalent processing conditions, return true;
        // otherwise, return false
        Sitecore.Caching.HtmlCache htmlCache =
          Sitecore.Caching.CacheManager.GetHtmlCache(
            Sitecore.Context.Site);
        return htmlCache != null && htmlCache.GetHtml(cacheKey) != null;
      }

      protected override string GetCachingID()
      {
        // define a unique identifier for this control
        string id = base.GetCachingID();

        if (String.IsNullOrEmpty(id))
        {
          id = this.GetType().ToString();
        }

        return id;
      }

      protected override void OnInit(System.EventArgs e)
      {
        // if the layout engine has already cached the output of this control
        // under equivalent processing conditions
        // in the output cache associated with the context site,
        // do nothing
        if (this.HasBeenCached())
        {
          return;
        }

        // expensive initialization logic
      }

      protected override void DoRender(HtmlTextWriter output)
      {
        // rendering logic
      }
    }
  }
```

The source code for the `SitecoreBook.Web.UI.WebControls.PerformsExpensiveInitialization` web control class shown in Listing 6-1 defines the `HasBeenCached()` method to determine whether the output of this control under equivalent processing conditions already. The `HasBeenCached()` method calls the `GetCacheKey()` method defined in the `Sitecore.Web.UI.WebControl` abstract base class, and then checks whether the output cache contains an entry with that key exists in the

www.it-ebooks.info

output cache for the context site. The `GetCacheKey()` method in the `Sitecore.Web.UI.WebControl` abstract base class from which the `SitecoreBook.Web.UI.WebControls.PerformsExpensiveInitialization` web control class derives adds the value returned by calling the `GetCachingID()` method of the control to strings that identify the Vary By caching criteria defined for each use of each control. The resulting cache key identifies the output of those controls under equivalent processing conditions. If the `GetCacheKey()` method returns null or an empty string, Sitecore does not cache the output of the control; in this case the `HasBeenCached()` method returns false. If the `GetCacheKey()` method returns a string that is not empty, and an output cache exists for the context site, and that cache contains an entry for this control under equivalent processing conditions as defined by the Vary By caching options specified for this instance of the control, the `HasBeenCached()` method returns true.

Like any web control that supports output caching, the `SitecoreBook.Web.UI.WebControls.PerformsExpensiveInitialization` web control class shown in Listing 6-1 implements the `GetCachingID()` method to identify the individual presentation control. If your web controls inherit from a custom abstract base class that inherits from the `Sitecore.Web.UI.WebControl` class and implements the `GetCachingID()` method, such as the `SitecoreBook.Web.UI.WebControl` abstract base class shown in Listing 5-7, you can move the `HasBeenCached()` method to that abstract base class.

The `SitecoreBook.Web.UI.WebControls.PerformsExpensiveInitialization` web control class shown in Listing 6-1 defines the `GetCachingID()` method to return a string that uniquely identifies this presentation control. If your web controls inherit from a custom abstract base class that inherits from the `Sitecore.Web.UI.WebControl` class and implements the `GetCachingID()` method, such as the `SitecoreBook.Web.UI.WebControl` abstract base class shown in Listing 5-7, you do not need to implement the `GetCachingID()` method in each concrete web control as shown in this example.

The `OnInit()` method of the `SitecoreBook.Web.UI.WebControls.PerformsExpensiveInitialization` web control shown in Listing 6-1 calls the `HasBeenCached()` method to check whether an entry generated by the presentation control previously under equivalent processing conditions exists in the output cache for the context site. If the call to the `HasBeenCached()` method returns `true`, the `OnInit()` exits, avoiding unnecessary initialization logic. Otherwise, the `OnInit()` method continues to initialize the control.

For more information about output caching, see The Sitecore Presentation Component Reference (`http://bit.ly/o2TdvV`) and my blog posts at `http://bit.ly/ndR0Bz` and `http://bit.ly/x399nO`. For information about adding custom Vary By options for output caching, see my blog post at `http://bit.ly/oTMB4l`. For information about scheduling expiration of entries from output caches, see my blog post at `http://bit.ly/qWNPjq`.

Tuning Cache Sizes

You can specify a maximum size for all but media caches. When Sitecore determines that a cache is close to reaching its size limit, it evicts one or more entries from the cache or clears the entire cache before adding any additional entries to it. For instructions about how to configure the size of each cache, see preceding sections and The Sitecore Cache Configuration Reference (`http://bit.ly/qYIAia`).

www.it-ebooks.info



Sitecore does not allocate memory for caches at initialization; it creates new entries in each cache until it reaches the limit for each.

To help determine an appropriate size limit for each cache, as well as to monitor the current utilization of each cache and its growth over time, access the cache administration page at `/sitecore/admin/cache.aspx` on your Sitecore instance, such as `http://sitecorebook/sitecore/admin/cache.aspx`.



In load-balanced environments, the cache administration page shows information for the individual instance on which the page runs rather than all instances in the balance.

Generate some load against your solution before evaluating the cache administration page, and then click Refresh in the page to populate the table. You can click Clear All in the cache administration page to clear all of the Sitecore caches in memory completely.

Use the cache administration page to tune the size limits for each cache. When you determine that the utilization of a cache has approached or exceeded its size limit, consider increasing that size limit if available RAM permits. You can use the CacheTuner Sitecore Shared Project (`http://bit.ly/xCHKCH`) to help identify caches that could benefit from larger size limits.

The table displayed by the cache administration page contains the following columns as shown in Figure 6-3.

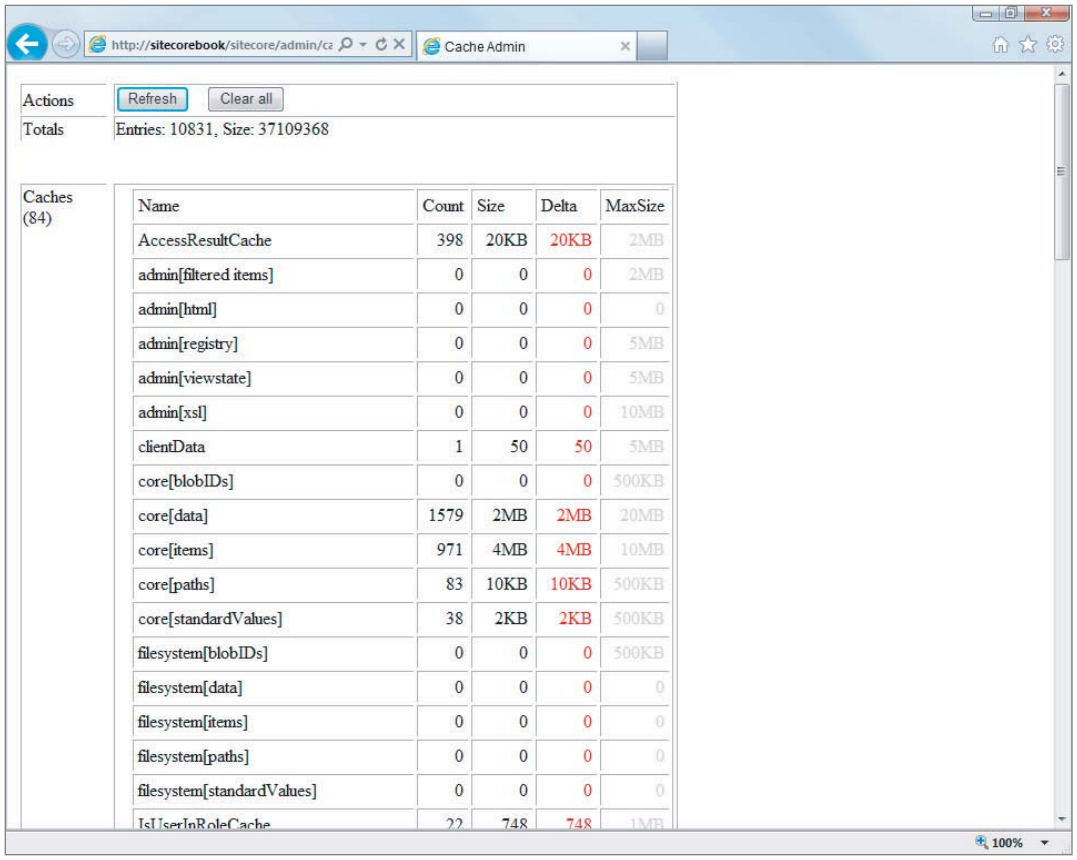
- **Name** — Name of the cache, which can indicate the database or managed site associated with the cache in addition to the type of data it contains
- **Count** — Number of entries currently in the cache
- **Size** — Approximation of total memory consumed by the cache
- **Delta** — Approximation of the amount of memory consumed by the cache since you last clicked Refresh



Because this page uses an ASP.NET postback to compute cache size deltas, always click Refresh instead of using the page reload feature of your browser.

- **MaxSize** — Size limit for the cache

www.it-ebooks.info



Name	Count	Size	Delta	MaxSize
AccessResultCache	398	20KB	20KB	2MB
admin[filtered items]	0	0	0	2MB
admin[html]	0	0	0	0
admin[registry]	0	0	0	5MB
admin[viewstate]	0	0	0	5MB
admin[xsl]	0	0	0	10MB
clientData	1	50	50	5MB
core[blobIDs]	0	0	0	500KB
core[data]	1579	2MB	2MB	20MB
core[items]	971	4MB	4MB	10MB
core[paths]	83	10KB	10KB	500KB
core[standardValues]	38	2KB	2KB	500KB
filesystem[blobIDs]	0	0	0	500KB
filesystem[data]	0	0	0	0
filesystem[items]	0	0	0	0
filesystem[paths]	0	0	0	0
filesystem[standardValues]	0	0	0	0
IsUserInRoleCache	22	748	748	1MB

FIGURE 6-3

When monitoring caches with the cache administration page, you should not see negative numbers in the Delta column unless you have restarted the application pool, cleared or removed entries from the cache with code, or otherwise knowingly reduced the number of items in the cache since you last clicked Refresh in that page. Except in such cases, a negative number in this column generally indicates that Sitecore removed items from the cache after it reached its size limit, in which case you should seriously consider increasing the size limit for that cache. At worst, caches should reach their size limits very infrequently. Requiring Sitecore to remove items from the cache in order to add them back afterward can significantly reduce solution performance by consuming CPU time simply for cache management.

For more information about the cache administration page, see The Sitecore Cache Configuration Reference (`http://bit.ly/qYIAia`). For a cache administration page replacement with an improved UI and more features, see the Caching Manager (`http://bit.ly/A5vk3Z`) Sitecore Shared Source Project.

www.it-ebooks.info

Disabling Cache Size Limits

Sitecore CMS 6.4.1 rev. 110928 introduced the `Caching.DisableCacheSizeLimits` setting to the Web.config file, which defaults to `false`. Sitecore CMS 6.5.0 rev. 111123 introduced the `Caching.DisableCacheSizeLimits` setting to the 6.5 line. Setting the `Caching.DisableCacheSizeLimits` setting to `true` prevents Sitecore from enforcing cache size limits, allowing each cache to grow indefinitely. You can set the `Caching.DisableCacheSizeLimits` setting to `true` during load testing to evaluate how large caches would grow without size limit constraints, which can help you determine appropriate cache size limits. If you leave `Caching.DisableCacheSizeLimits` set to `true`, you should have no need to tune cache sizes. However, you may need to worry about running out of memory, as you cannot control how Sitecore uses the caches with this setting in place. You should leave `Caching.DisableCacheSizeLimits` set to `true` only on 64-bit platforms with sufficient available RAM.



Whether you choose to set the `Caching.DisableCacheSizeLimits` *setting to* `true` *may depend on your hosting environment. If you use an Internet cloud that provides capacity on demand, the cost savings of reducing memory consumption may outweigh the performance advantage of extreme caching. Additionally, virtualized environments can limit available RAM. Use this setting only on systems with sufficient available memory. In any case, tuning the content of prefetch caches (as opposed to the size limit of these caches) can always help to implement the best trade-off between startup time and efficient processing.*

Bringing the Most Bits Together

As the title of this section indicates, I simply cannot resist a bad pun, even if it results in a technical inaccuracy. The following list describes, and Figure 6-4 demonstrates, the dependencies between the caches described in this section:

- When the layout engine processes a rendering for which you have configured output caching, if output generated previously under equivalent processing conditions does not exist in the cache, Sitecore invokes the rendering. Each rendering typically accesses some number of items in the item cache for the context database. Theoretically, a rendering could access items in multiple databases. Sitecore manages separate output caches for each managed site; the content of these caches often depends on the data in the item cache for the database associated with the context site.

www.it-ebooks.info

- If required data does not exist in the item cache, Sitecore retrieves it from the data cache and populates the item cache with an item representation of that data.
- If required data does not exist in the data cache, Sitecore retrieves it from the prefetch cache and populates the data cache with that data.
- If required data does not exist in the prefetch cache, Sitecore retrieves it from the database and populates the prefetch cache with that data.

The importance of output caching varies according to the specifics of your solution. The performance of extremely efficient renderings that access a very small number of items may not improve significantly with output caching. Renderings that access external resources, such as databases and RSS feeds, may benefit more from output caching.

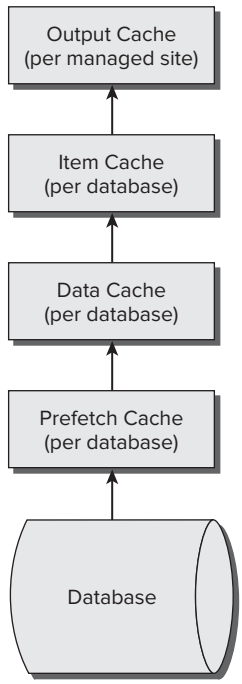


FIGURE 6-4

Excluding output caches, the database caches (prefetch, data, and item caches) are typically most important. Sitecore implements database caching in an attempt to optimize every aspect of the solution, including CMS user interface performance as well as that of the content delivery environment, and to minimize interaction with the underlying database.

Under the default and all correct configurations according to The Sitecore Scaling Guide (`http://bit.ly/oKvYuN`), Sitecore updates almost all caches in memory in both single-instance and multi-instance environments automatically when the underlying data changes — for example, when a user updates or publishes an item. The site output caches present a single exception to this rule. Because it is infeasible for Sitecore to track all the items on which the output of each rendering depends under all possible processing conditions, the system does not identify which entries in the site output caches to remove when those underlying items change. For this reason and as described previously, whenever you publish, Sitecore completely clears output caches for the managed sites. Specifically, on the local instance, Sitecore clears the output caches for the managed sites listed under the `/configuration/Sitecore/events/event` named `publish:end` in the Web .config file; on content delivery instances separate from the content management environment, Sitecore clears the output caches for the managed sites specified under the `publish:end:remote` event.



While efficient utilization of site output caches can dramatically increase performance, these caches should not provide the foundation of your optimization effort, especially for solutions that publish frequently.

www.it-ebooks.info

SCALING YOUR SOLUTION

This section focuses on the challenge of scaling Sitecore for larger installations, specifically those involving multiple content management and content delivery instances hosting a single Sitecore solution. Sitecore supports very large installations measured both in terms of traffic (the number of HTTP requests per second) and content volume (the number of items in the content databases). Scaling provides for higher capacity and helps to ensure availability in the case of hardware failure. Sitecore handles both objectives in the same manner, though the underlying infrastructure requirements may differ in some regards.



To minimize costs, optimize your solution (both code and caching) before determining scalability requirements.

Horizontal scaling, or scaling across multiple servers, is typically the best way to scale Sitecore solution as opposed to vertical scaling, or increasing the capacity of individual servers. Sitecore natively supports extreme scalability and largely relies on the hosting infrastructure to scale across servers. Therefore, other than enabling scalability features, Sitecore does not require much administration to scale across servers. As described in Chapter 9, Sitecore does present some unique challenges in deploying software and solution updates to multiple instances.



Adding capacity for Sitecore cannot help to scale that solution if the bottleneck is a dependency on a database or integrated external application.

Most of the work you will do to scale your Sitecore solutions involves scaling your hosting infrastructure, including the following:

- Hardware sizing, both in terms of individual machine capacity and the number of servers
- Firewalls and load balancing, which involves hardware and software to secure the solution and spread the load between those servers
- Database server scalability and optimization
- Network infrastructure capacity, both local area network (LAN) for connections to the database and wide area network (WAN) for the connection to the Internet

Many of the specific details involved in scaling a Sitecore solution depend on the unique performance profile of that solution. For example, a solution requires less hardware for an equivalent load if the code is efficient and you can cache the output of most renderings, as opposed to the same solution with inefficient code and without output caching. Additionally, a complex solution might require more hardware than a simple solution.

www.it-ebooks.info



Always consider scalability to support traffic in excess of anticipated load over a short time duration at your peak traffic periods. Other than cost (licensing, hardware, network bandwidth, administration, power, and so on), it cannot harm your solution to acquire more capacity than you expect to need under peak load.

In addition to the techniques described in this section, you can use Sitecore Azure to deploy your solution to the Microsoft Windows Azure Internet cloud, effectively resulting in unlimited scalability and geographic dispersion at minimal cost. For more information about Sitecore Azure, see Chapter 10.

Scaling Sitecore requires that you enable relevant features as described in The Sitecore Scaling Guide (`http://bit.ly/oKvYuN`). The `/App_Config/Include/ScalabilitySettings.config.example` Web.config include file lists the most common settings involved in enabling scalability. Of these, the most important may be the `EnableEventQueues` setting. When set to `true`, this setting enables Sitecore event queues, which Sitecore uses to pass messages between instances. Among other things, messages in event queues inform the various Sitecore instances about cache entries to remove and entire caches to clear after item updates, publishing events, and other operations.

Scaling Infrastructure

Before scaling Sitecore into multi-instance environments, you need to understand the infrastructure of a Sitecore environment. You can deploy Sitecore in a nearly unlimited number of configurations; this section describes some of the most common approaches.

As mentioned in Chapter 1, each Sitecore instance depends on an ASP.NET application, which consists of an Internet Information Services (IIS) website using ASP.NET (Application Server Pages with .NET) running on the Microsoft Windows operating system. A Sitecore environment consists of one or more such instances. A single Sitecore instance can host a content management environment for CMS users and any number of managed sites for visitors in the content delivery environment integrated by default into that instance. Each instance depends on one or more database servers, where multiple instances can share databases on those servers, and different instances in a single environment can use different databases. Regardless of which database vendor you use, the principles involved in scaling that infrastructure are the same, though Oracle and SQL Server name the corresponding technologies differently and use different tools for these purposes.

One of the first steps to scaling and securing your Sitecore solution is to separate the content management environment, typically inside your LAN, from the content delivery environment, typically in a periphery network. To scale further, add instances to one or both of these environments. Most solutions begin by scaling the content delivery environment to meet site visitor load; you can scale the content management environment if you have a very large number of concurrent CMS users, to offload publishing overhead to a single content management instance (in the case of frequent publishing or very large content databases), and to provide failover in that environment.

www.it-ebooks.info

Load Balancing

After separating the content management environment from the content delivery environment, the most common technique to scale Sitecore solutions involves load balancing. Load balancing uses two or more Sitecore instances to serve a common solution. Sitecore allows you to load balance the content management environment, the content delivery environment, or both. To support load balancing, enable a scalable infrastructure as described previously, and follow best practices for scalability of ASP.NET solutions.

When you decide to load balance a solution, you face a number of choices. Primarily, you can balance using Microsoft Network Load Balancing (MS NLB) built into the Microsoft operation system, or you can use external load-balancing hardware from a number of vendors such as BIG-IP, Citrix, Cisco, and Barracuda. The Sitecore CMS does not limit your options for load-balancing software and hardware that do not affect the software platform. That said, if you plan to immediately or eventually scale beyond two Sitecore instances, you should use load-balancing hardware. Such hardware provides greater application failure detection and failover management. MS NLB relies on the network stack to detect hardware failure in order to implement such transitions; it may not detect and remove an instance from the balance in case of application failure.

While Sitecore solutions can employ clustering technology, it does not require such solutions. You can scale your Sitecore solutions simply by adding servers and instances side-by-side to the balance in each environment, where the load-balancing solution routes HTTP requests to the most appropriate server determined from configuration of the balance and current system loads.

In addition to scalability, load balancing supports high availability; if one of the instances in the balance fails, the load balancer routes traffic to any of the remaining instances. For many Sitecore customers, scalability and high availability in the content delivery environment are more important than the same features in the content management environment. Hence, the most common environment to balance is content delivery. The basic principles in scaling either environment are similar, but also consider publishing from the content management environment. The following sections describe aspects of load balancing specific to each environment.

Balancing Content Management

A load-balanced content management environment consists of two or more Sitecore instances hosting CMS users managing one or more managed sites, typically supported by three or more Sitecore databases (the Master database, the Core database, one or more publishing target databases, potentially a separate database for security information, and possibly additional publishing target and other databases). By default, each content management environment includes a content delivery environment that CMS users can access to evaluate aspects of the solution, but the visiting public typically cannot access the content delivery environment available in the content management environment. Instead, visitors access a production content delivery environment separate from the production content management environment.

You can balance the content management environment to support a large number of CMS users, to place CMS instances in multiple geographic locations for improved local performance for CMS users, and to provide fault tolerance and high availability in that environment. After content

www.it-ebooks.info

delivery, organizations most often balance the production content management environment (as opposed to content management environments used for development and testing).



To test failover and other load balancing features, you need balanced test environments similar to your production environments.

Before balancing the content management environment, you need to either disable the capability to store media as files or configure software, such as Microsoft Distributed File System (DFS), to synchronize media files between the content management instances. If you choose to allow file media in a balanced content management environment separate from the content delivery environment, you must also use software, such as Microsoft WebDeploy, to deploy those files from the content management environment to the content delivery environment, which should also use software such as DFS to synchronize media files between the instances in that environment. The Sitecore Scaling Guide (`http://bit.ly/oKvYuN`) provides instructions on how to disable file media, configure DFS, and configure WebDeploy.



I strongly recommend that you disable support for storing the binary content of media items on the file system rather than in the database. If you decide that you must store media on a file system, however, I recommend that you consider integration with edge caching systems and CDNs as an alternative to file media. The `/App_Config/Include/ScalabilitySettings.config.example` *Web.config include file most commonly used to enable Sitecore scalability disables file media by default by setting the* `Media.DisableFileMedia` *setting to* `true`.

In addition to addressing file media, if you balance the content management environment you should also set the `Publishing.PublishingInstance` setting on all instances in that environment to specify an individual content management instance to perform publishing operations, as described in a subsequent section of this chapter. Otherwise, each instance in the balance may republish items previously published by other instances, which can only harm performance. You do not need to dedicate an entire instance to publishing; you can select any instance in the balance to perform these operations, and allow CMS users to access that instance like any other in the content management environment.

Finally, you should use the InProc mode for managing ASP.NET sessions on all instances in the content management environment. For information about ASP.NET session management options including InProc, see `http://bit.ly/zAGJyw`. In short, with InProc mode, ASP.NET manages session information in memory instead of using a dedicated process or database server to manage that information, which prevents sharing the ASP.NET session state between instances in the balance. For InProc to work, you must configure the load balancer for server affinity (also called sticky sessions), which causes that balancer to direct all traffic from an individual browser to one of the instances in the balance instead of spreading that traffic between all instances. InProc session management generally results in better application performance; but if a server fails, the CMS user loses their session.

www.it-ebooks.info



The same session management concerns can apply to the content delivery environment if you use session state there. Unlike the content delivery environment, Sitecore actually requires InProc in the content management environment, whereas InProc is simply an optimization in the content delivery environment. Wherever you use InProc, you should enable server affinity in the load balancer.

Balancing Content Delivery

A load-balanced content delivery environment consists of two or more Sitecore instances hosting one or more managed sites supported by at least one Sitecore database (at least one publishing target database, an optional Core database, potentially a database for security artifacts, and possibly more databases, such as to support optional modules). Most Sitecore customers that load balance Sitecore do so in the production content delivery environment, where performance and availability are often of critical importance. Content delivery includes primarily the layout engine and any custom applications that you develop, supported by one or more Sitecore databases. You should also apply information in The Sitecore Security Hardening Guide (`http://bit.ly/swFfRp`) to maximize security in the content delivery environment.

Specifically, in the content delivery environment you should remove all references to the Master database. Because content delivery depends on publishing target databases, and the Master database contains unpublished work in progress that is not appropriate for content delivery, you should not expose that database and that environment. To share security information without duplication and to use a common event queue to optimize cache clearing, you can share the Core database between the content management and content delivery environments.

Otherwise, configuring load balancing in the content delivery environment is very similar to configuring load balancing in the content management environment. For specific instructions on how to configure load balancing in either environment, see The Sitecore Scaling Guide (`http://bit.ly/oKvYuN`).

Scaling Publishing

Publishing involves the transfer of updated items from the Master database used in the content management environment to one or more content delivery databases that support one or more content delivery environments. Not only does publishing clear caches, but it can be a CPU-intensive process in the content management environment that can affect the performance of the publishing instance. For more information about publishing, see Chapter 9.

A number of factors can trigger publishing operations, such as when a CMS user chooses to publish through the graphical user interface, when a content item reaches a given state in a workflow, on a regular basis using a scheduled process, and if you write custom code that invokes APIs that publish. You should make an effort to minimize the number of publishing operations required to support your solution. Experience has shown that many organizations publish an excessive number of items more frequently than needed. Specifically, CMS users are often unaware of the preview functionality built into the CMS, and often publish changes, review them visually, continue editing, and publish again. Considering that publishing consumes resources and clears entire output caches,

www.it-ebooks.info

this approach is unnecessarily harmful to performance and hence the scalability of both the content management and content delivery environments.

You can take two steps to improve the performance of a Sitecore solution that involves frequent or large publishing operations. The first step is to work with your editorial staff to reduce the frequency of unnecessary publication. If CMS users use the edit-publish-check-edit process described previously, you will likely experience performance challenges regardless of your hosting infrastructure, the quality of your code, and anything else you attempt to improve performance, capacity, and scalability. Educate your CMS users to use the preview functionality of the CMS, and spend your development resources ensuring that the preview features closely mimic visual aspects of the published sites. In some cases, you can implement features available in previewing modes that exceed those available on the published sites, such as by improving CMS performance and usability by allowing CMS users choose whether to display page components that use Adobe Flash.



To get the most productivity in your development environments, you should take a similar approach when you update code. Every time you build your solution, Sitecore reinitializes, which is worse for performance than clearing all caches. Though it requires more engineering discipline than changing small amounts of code and building, you will benefit from changing significant amounts of code and then evaluating the impact of those combined changes.

In some cases, you may even want to limit the number of users with rights to publish, and instead use an incremental, scheduled publishing operation, and possibly a workflow action to publish when users have completed all changes. This approach strongly encourages CMS users to preview their changes instead of publishing them, reducing stress on the system. Because CMS users often want to know when their changes will publish, you can build visual indicators into the desktop, the Content Editor, and even the Page Editor, such as countdowns to indicate when the next publishing operation will occur.

The second step to improve the performance of the content management environment in a solution that involves frequent or large publishing operations is to dedicate a content management instance to publishing operations, and to remove that instance from the balance. With this approach, you offload expensive publishing operations to an instance committed only to publishing operations, so that publishing does not consume resources on the content management instances accessed by CMS users. A significant number of Sitecore installations, especially those with very large content repositories, use this approach to mitigate the effect of publishing operations in the content management environment. To set the publishing instance, in the content management environment, configure the `PublishingInstance` setting in the Web.config file, typically in the `/App_Config/Include/ScalabilitySettings.config.example` file after renaming that file to `ScalabilitySettings.config` to activate that configuration.

Scaling the Database Layer

Each instance in each environment must have the ability to connect to the databases on which it depends. Such connections may require you to open ports in one or more firewalls between the Sitecore instances and the database servers. The resiliency, scalability, and performance of any

www.it-ebooks.info

Sitecore instance depend on the underlying databases. To ensure these features, do your best to guarantee the corresponding features at the database layer.

Sitecore can connect to databases that use standard database server clustering technologies. Typically, you implement clustering for reliability and sometimes scalability, rather than performance (clustering the database can actually reduce performance). You can achieve some increase in reliability by implementing multiple publishing targets in favor of database clustering, which generally requires a more expensive database server license. You can use database replication features to synchronize multiple databases as described further in this chapter and in The Sitecore Scaling Guide (`http://bit.ly/oKvYuN`).

Virtualizing Sitecore

Virtualization software enables multiple virtual machines to run on a single physical machine, and it can enable a single virtual machine to span multiple physical machines. Numerous Sitecore customers use the CMS software in virtualized environments for both small-scale and large-capacity solutions. Virtualization provides a number of advantages for server administration, scalability, redundancy, and cost reduction by helping an organization utilize a greater percentage of available machine resources.

Sitecore has no requirements that you cannot address with virtualized hardware, though you should ensure that you allocate enough actual hardware to each virtual machine as required by Sitecore. For specific hardware requirements for each physical or virtual instance of your version of Sitecore, see The Sitecore Installation Guide at `http://bit.ly/pv7X3B`, noting that optional components such as the Digital Marketing System (DMS) can require additional resources. The choice of virtualization software can also affect performance of the virtualized environment.

Sitecore content delivery instances are often excellent candidates for hardware virtualization. While virtualization of the web server and the database server should not directly affect Sitecore itself, you should be aware that Sitecore solutions tend to implement many small, parallel transactions with the database. Sitecore content databases do not typically place an untoward level of stress on database servers. The features of Sitecore that you use, such as complex authorization rules in the content delivery environment and especially the use of DMS, can increase the number of database transactions. Therefore, the performance of storage subsystems on the database server can affect the performance of the Sitecore solution, so it is not always advantageous to virtualize the database server.

Edge Caching and Content Delivery Networks

You can use edge caching devices and content delivery networks (CDNs) to scale your Sitecore solutions. These technologies reduce the number of HTTP requests for the content delivery environment to service, and can place web servers closer to the clients that request resources, reducing network response time, especially for image-intensive sites and large, static, binary assets such as video. Both technologies effectively offload request processing from the Sitecore instances to the edge caching and CDN systems.

With edge caching, you deploy content to separate servers near the production content delivery environment. You can use a load balancer such as Citrix NetScaler (`http://bit.ly/yc7add`) to direct specific HTTP requests from Internet users to the caching servers instead of the Sitecore instances,

www.it-ebooks.info

or you can use dedicated caching technologies such as Varnish (`https://www.varnish-cache.org`) and Squid (`http://www.squid-cache.org`).

With CDNs, you cache resources using technologies that are not part of your infrastructure, instead using an external service that you purchase. CDNs often use specific URLs to differentiate resources that you manage as opposed to those that the CDN can cache, typically using separate Domain Name System (DNS) entries for each of these systems. On the first HTTP request for a resource, the CDN requests the asset from the Sitecore content delivery environment, and then caches it locally to service subsequent requests. CDNs are most valuable for audiences with a wide geographic spread, as larger networks synchronize cached content automatically across servers around the globe. With this approach, content consumers retrieve media from the caching servers closest to their individual locations, reducing network transit time for those assets.

You can use edge caching and CDNs for any static files, including those that you do not manage in the Sitecore media library, such as JavaScript (`.js` files), Cascading Style Sheets (`.css` files), and images referenced directly by developers in code and retrieved from the filesystem rather than referenced by managed content and retrieved from the media library. Some solutions enable "minification" of resources such as CSS and JavaScript, removing whitespace to reduce file size and hence network download time. You can also use CDN and edge caching technologies for binary material stored in the media library, which can dramatically improve performance of the overall solution.

Because edge caching systems and content delivery networks are unaware of Sitecore security, you should not use these technologies for resources that you secure. Additionally, if you use the Digital Marketing System (DMS) described in Chapter 10 to track the download of binary assets, you should not use edge caching or CDNs for those resources, nor should you cache any material externally for which the organization may need to measure access. For example, you may want to cache product images but not whitepapers and other resources that you wish to monitor.

Some Sitecore customers use edge caching for textual content managed in Sitecore. While this approach increases performance and hence scalability, especially for sites with massive traffic volumes, it reduces the potential for dynamic aspects of the solution such as per-user personalization. Before caching such content, consider the implications carefully. Certain edge caching technologies, such as NetScaler, can support personalization and similar features, but it can take some work to achieve your goals without adversely affecting Sitecore features.

Architecting High Availability

Many organizations strive for very high availability (HA) of systems, especially in the content delivery environment. Again, Sitecore capitalizes on the underlying technology stack to provide for HA at all levels, from network infrastructure to web server hardware and database servers. You should ensure redundancy in each of these layers individually, and apply HA solutions at all layers. The network layer includes network failover equipment such as switches, routers, firewalls, and load balancers not within the scope of this book, but critical to actually achieving high availability.

Considering only the CMS software, for high availability, you should configure least two load-balanced instances of Sitecore in the content delivery environment. If you need high availability in the content management environment, you must configure load balancing in that environment as well. If you employ virtualized hardware, the various instances should not depend on a single physical host, as a failure of that host could affect more than one virtualized instance.

www.it-ebooks.info

You should also configure at least two servers in the database tier, where you should use standard database replication technologies. In Microsoft SQL Server, you can implement merge replication, which is the most appropriate approach for Sitecore solutions and hence the approach supported by Sitecore. Some organizations use other replication modes in production environments, but Sitecore fully supports only merge replication in case the need for conflict resolutions between the databases arises. Reliability of the publishing target databases is critical to the content delivery environment, which you should isolate from the content management environment as described previously. You can use database replication to scale the publishing target database (and additional databases such as Core if needed) to reside on multiple hosts without the need to publish to each target database separately.

If you implement multiple content delivery environments in different geographies, you should configure at least one publishing target database to support each location. If publishing will likely present bottlenecks, it may be worthwhile to configure publishing to a single publishing target database in the content delivery environment closest to the content management environment, and then use database replication to copy content to the additional publishing target databases in different locations. For maximal performance and security, you can publish to a target database in the content management environment, and then use database replication to copy content from that publishing target database to all of the databases supporting the content delivery environments, potentially replicating the Core database as well.

Planning Hardware Capacity

For every solution, you must determine the amount of resources required, most importantly to service load in the content delivery environment. Questions to consider include:

- How many Sitecore instances does your solution require today (horizontal scaling)?
- How much server capacity should you dedicate to each instance (vertical scaling)?
- How can you determine when to add Sitecore instances?
- How much database capacity will you need (storage)?
- How can you determine when to add more publishing target and other databases?

While the answers to these questions are important, they depend on solution-specific factors including the amount of visitor traffic, the complexity of the solution, the quality of the implementation, the amount of caching that implementation allows, and page performance requirements, as well as the frequency and volume of publishing. Capacity requirements can vary significantly between solutions, and no general formulas to answer these questions apply to all implementations.

The most common processing bottleneck in a Sitecore solution is the time required to generate HTML. The amount of processing power needed depends significantly on the amount of caching the solution can leverage for rendered output, but also on how well you control application restarts, which can be expensive in terms of computing resources. While you cannot estimate hardware requirements without knowledge of the actual solution, the amount of RAM available for caches can significantly affect solution performance. The only rule to determine the amount of memory required to optimize performance to the utmost is that each instance should have enough memory to avoid reaching any limits defined for prefetch, data, item, and output caches. While some solutions may benefit most from 16GB of memory, smaller solutions may function well with smaller amounts, while other solutions could leverage more.

www.it-ebooks.info

As an example, major components of an actual eCommerce site built with Sitecore include the CMS, the Digital Marketing System (DMS), and a search engine. At peak volume, this solution experiences more than 300,000 page views per hour, equating to more than 150 page views per second. Even after optimizing the code, prefetch cache configuration, and general tuning of IIS including content expiration and compression, to support these requirements, the content delivery environment depends on three physical machines, each supported by eight processor cores and 16GB of memory, plus a dedicated database server with eight processor cores. Only very large Sitecore installations require multiple publishing target databases; some solutions involve more than ten Sitecore instances accessing a single publishing target database. Because the Digital Marketing System involves high transaction rates, you should consider a separate database server for the analytics database used by that optional module.

TROUBLESHOOTING YOUR SOLUTION

Inevitably, despite your best efforts, you will experience issues with your solution that require some troubleshooting, preferably in a development environment but sometimes in production. This section contains information to assist you in that effort.

The Sitecore Log Files

Sitecore uses Apache log4net (`http://bit.ly/wC4Ri0`) to log system activity and audit messages. Sitecore periodically creates a new log file in the subdirectory specified by the `LogFolder` setting in the Web.config file. The Sitecore log files contain a wealth of information that you can use to diagnose a Sitecore solution.



The Sitecore log files, which record Sitecore activity, are completely separate from the IIS log files, which record web client access to the solution.

Sitecore writes both system and audit messages to a single log. To configure Sitecore to write audit messages to a separate log file, which can simplify audit trail monitoring, update the `/web.config` file as described here:



Because the `/configuration/log4net` *section is not within the* `/configuration/sitecore` *section of the* `/web.config` *file, you cannot implement this change using a Web.config include file.*

1. Add the following elements after the `/configuration/log4net/appender` element named `LogFileAppender`:




```
<appender name="AuditFileAppender"
type="log4net.Appender.SitecoreLogFileAppender, Sitecore.Logging">
  <file value="$(dataFolder)/audit/audit.{date}.txt" />
  <filter type="log4net.Filter.StringMatchFilter">
```

www.it-ebooks.info

```
    <regexToMatch value="^AUDIT" />
  </filter>
  <filter type="log4net.Filter.DenyAllFilter" />
  <appendToFile value="true" />
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="%4t %d{ABSOLUTE} %-5p %m%n" />
  </layout>
</appender>
```

AuditFileAppender.config

2. Add the following elements before the `/configuration/log4net/root/appender-ref` with a value of `LogFileAppender` for the `ref` attribute:

```
<appender-ref ref="AuditFileAppender" />
```

To prevent specific messages from appearing in the system log, you can use the `<filter>` element within a `/configuration/log4net/appender` element in the `/web.config` file. For example, to prevent messages containing the text `Sitecore.Tasks.CounterDumpAgent` from appearing in the system log, add a section such as the following within that `/configuration/log4net/appender` element named `LogFileAppender`:

```
<filter type="log4net.Filter.StringMatchFilter">
  <stringToMatch value="Sitecore.Tasks.CounterDumpAgent" />
  <acceptOnMatch value="false" />
</filter>
```

To control the verbosity level in the Sitecore log, set the `value` attribute of the `/configuration/log4net/root/priority` element to `DEBUG`, `INFO`, `WARN`, `ERROR`, or `FATAL`, where `DEBUG` is the most verbose and `FATAL` records only critical log entries.



I recommend setting the value attribute of the `/configuration/log4net/root/priority` to `WARN` or a more verbose option. You should monitor the log for `WARN`, `ERROR`, and `FATAL` entries, and always try to resolve and eliminate as many as possible in all environments. To find such messages in a Sitecore log file, search for the tokens `WARN`, `ERROR`, and `FATAL`.

Depending on the configured logging verbosity level, you can use the `Audit()`, `Debug()`, `Error()`, `Fatal()`, `Info()`, and `Warn()` methods of the `Sitecore.Diagnostics.Log` static class to write messages to the system log. You can also use the `Error()` and `Fatal()` methods to log exceptions. All log entries contain at least the time, the verbosity level, and a message, and exceptions include a stack trace and information about any nested exceptions.

When writing a message to the Sitecore log, I like to include the name of the class writing the message to make it easier to find my own messages in the log. For methods that require an owner parameter, I generally pass `this`, which is the class that generated the error message. For example:

```
Sitecore.Diagnostics.Log.Info(this + " is a horrible way to debug.", this);
```

www.it-ebooks.info

For more information about Sitecore logging, see my blog post at `http://bit.ly/qYym7S`. That blog post includes links to additional posts that describe how write to your own separate log files, move old log files from the default subdirectory at application initialization, log to a SQL Server database instead of files, and integrate with the log4net dashboard, which provides a graphical user interface for investigating logs generated with log4net.

Rendering Statistics

You can use the rendering statistics page to identify underperforming renderings and renderings for which you may be able to improve output cache configuration, by enabling output caching or caching output by fewer Vary By options, for example. To access the rendering statistics page, navigate a browser to `/sitecore/admin/stats.aspx` on your Sitecore instance, such as `http://sitecore-book/sitecore/admin/stats.aspx`.



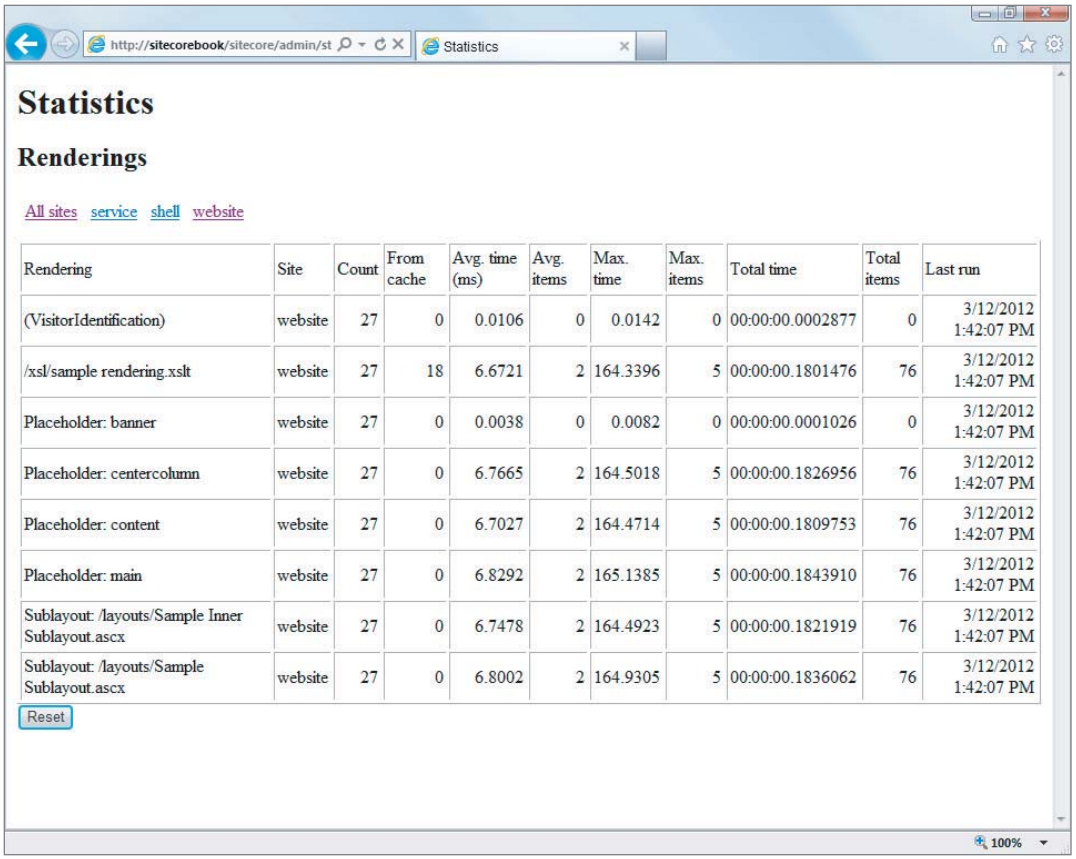
In load-balanced environments, the rendering statistics page shows information for the individual instance on which the page runs rather than all instances in the balance.

Generate load before reviewing the rendering statistics page. You can click links at the top of the rendering statistics page to filter the data by managed site. Figure 6-5 shows how the rendering statistics page appears after clicking the website link to exclude data for other managed sites.

The table displayed by the rendering statistics page contains the following columns as shown in Figure 6.5.

- **Rendering** — Identifier for an individual presentation control
- **Site** — Context site associated with data presented in this row
- **Count** — Total number of uses of this control by this site since application initialization
- **From cache** — Number of times the layout engine retrieved output for this control from cache instead of invoking the control
- **Avg. time (ms)** — Average duration of an invocation of the control
- **Avg. items** — Average number of items accessed during an invocation of the control
- **Max time** — Maximum duration of a single invocation of the control
- **Max. items** — Maximum number of items accessed during an individual invocation of the control
- **Total time** — Total duration of all invocations of the control
- **Total items** — Total number of items accessed while processing the control
- **Last run** — Date and time the layout engine last invoked the control

www.it-ebooks.info



Rendering	Site	Count	From cache	Avg. time (ms)	Avg. items	Max. time	Max. items	Total time	Total items	Last run
(VisitorIdentification)	website	27	0	0.0106	0	0.0142	0	00:00:00.0002877	0	3/12/2012 1:42:07 PM
/xsl/sample rendering.xslt	website	27	18	6.6721	2	164.3396	5	00:00:00.1801476	76	3/12/2012 1:42:07 PM
Placeholder: banner	website	27	0	0.0038	0	0.0082	0	00:00:00.0001026	0	3/12/2012 1:42:07 PM
Placeholder: centercolumn	website	27	0	6.7665	2	164.5018	5	00:00:00.1826956	76	3/12/2012 1:42:07 PM
Placeholder: content	website	27	0	6.7027	2	164.4714	5	00:00:00.1809753	76	3/12/2012 1:42:07 PM
Placeholder: main	website	27	0	6.8292	2	165.1385	5	00:00:00.1843910	76	3/12/2012 1:42:07 PM
Sublayout: /layouts/Sample Inner Sublayout.ascx	website	27	0	6.7478	2	164.4923	5	00:00:00.1821919	76	3/12/2012 1:42:07 PM
Sublayout: /layouts/Sample Sublayout.ascx	website	27	0	6.8002	2	164.9305	5	00:00:00.1836062	76	3/12/2012 1:42:07 PM

FIGURE 6-5

Because other processing can occur concurrently with the invocation of each control, Sitecore cannot accurately report exactly the average or maximum number of items accessed by each control or the average, maximum, or total actual processing time for each control. For example, the Max. items column in the image shown in Figure 6-5 shows the same value for several presentation components because layout details nest these components, so processing for the sublayouts actually shows the number of items accessed by all descendant controls.

Despite this limitation, information presented on the Rendering Statistics page can help you determine presentation components that consume significant processing time (high numbers in the Avg. Time, Max. Time, and Total Time columns) or access an inordinate amount of items (high numbers in the Avg. Items, Total Items, and Max. Items columns), which can affect processing time. This tool can also help you determine presentation controls for which you may be able to enable caching or reduce the number of Vary By caching options to increase the frequency of retrieving output from cache rather than invoking the control anew (a low number in the From Cache column relative to the value in the Count column).

www.it-ebooks.info

Tracking Performance Thresholds

You can configure Sitecore to log a warning if processing an HTTP request does not complete within a given period, appears to access more than a given number of items, or appears to consume more than a given amount of memory.



While it may appear that a single request exceeded one or more thresholds, in some cases, other system activity at the same time — including that required to service concurrent HTTP requests — could utilize processing time, access items, and consume memory that would appear to be associated with that HTTP request. Specifically, Sitecore initializes itself during the first HTTP request after the application pool restarts, which can consume time and memory, and access a large number of items. Threshold warnings immediately after an application pool restart may be invalid. After restarting an application pool, to initialize Sitecore load a page other than the suspect page, and then load the suspect page to see if it generates a threshold warning.

To configure Sitecore to log such threshold warnings, set the value of the `<ShowThresholdWarnings>` element of the `StopMeasurements` processor in the `httpRequestEnd` pipeline defined in the Web.config file to `true`, and optionally override the default values of the `<TimingThreshold>`, `<ItemThreshold>`, and `<MemoryThreshold>` elements of that processor to define the thresholds for these criteria that you deem appropriate for your solution.

Debugging in a Browser

You can use the Sitecore browser-based debugger remotely or locally to troubleshoot and diagnose issues with your solutions, as well as to find opportunities for further performance optimization, without installing any software. To access the Sitecore debugger, log into the Sitecore desktop, click the Sitecore button, and click Debug from the menu that appears.

The Sitecore debugger opens in a new browser window that resolves the Sitecore context as it would in the content delivery environment. Under the default configuration, the debugger accesses the `/sitecore/content/home` item in the Web database using the configuration defined for the `/configuration/sitecore/sites/site` element in the Web.config file named `website`. The icon at the far right of the gray bar that appears at the top of the debugger contains an icon to show or hide the debugging Ribbon. Figure 6-6 shows the debugger as it appears after clicking that icon.

Using the Modes group on the debugger Ribbon, you can enable or disable inline editing with the Edit command and switch between the Preview and Debug interfaces. Using the Profile group in the debugger Ribbon, you can enable or disable performance profiling and access a high-level performance profile for the page. The profile attempts to identify hot spots, which are the components that appear to perform worst. Using the Trace group in the debugger Ribbon, you can enable or disable low-level tracing and access a low-level trace for the page. When you enable profiling or tracing, the debugger renders the profile or page trace after the page body as shown in Figure 6-7.

www.it-ebooks.info

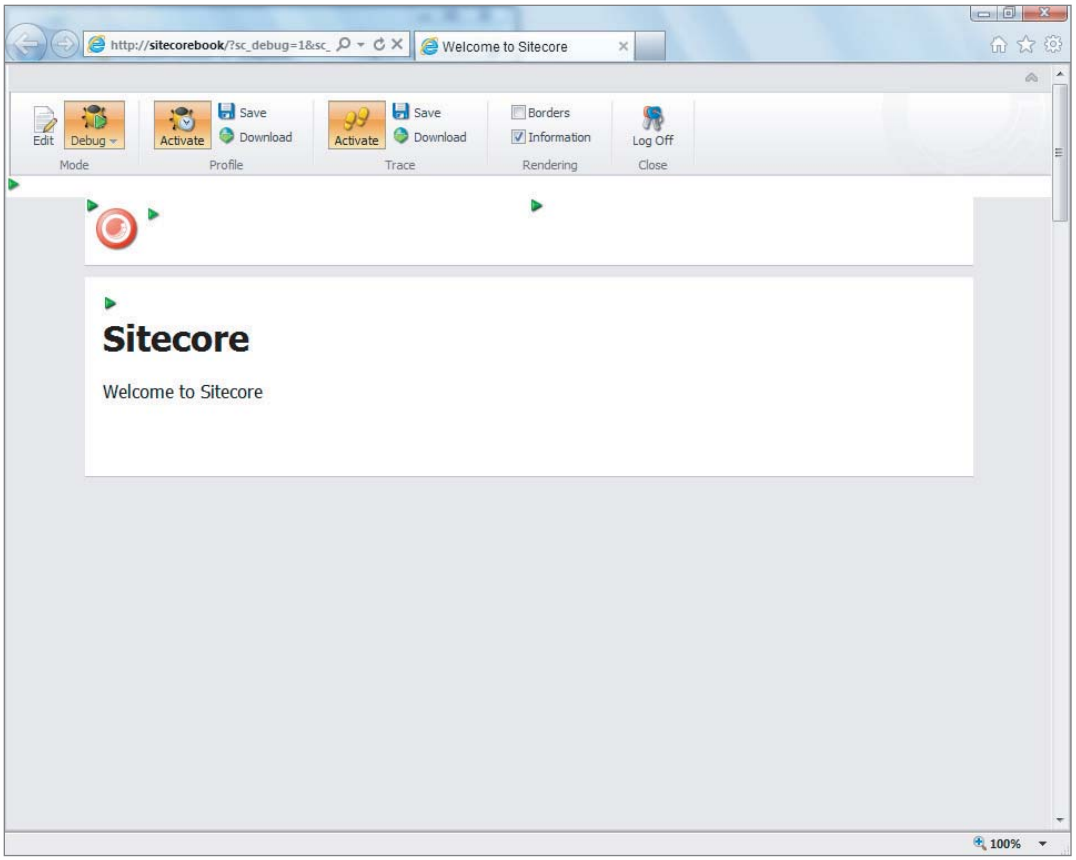


FIGURE 6-6



The profile and trace can contain error messages that do not otherwise appear on the page, such as if you attempt to bind a presentation component to a placeholder that does not exist.

Using the Open group in the debugger Ribbon, you can access the ASP.NET trace for the current page, which is separate from the trace that Sitecore maintains.

You can use the `Info()` method of the `Sitecore.Diagnostics.Tracer` static class to write to the trace from a .NET presentation component, as shown in the following example:

```
Sitecore.Diagnostics.Tracer.Info(this + " : debugging message");
```

You can use the equivalent `sc:trace()` XSL extension method to write to the trace from an XSL rendering, as shown in this example:

```
<xsl:value-of select="sc:trace('debugging message')" />
```

www.it-ebooks.info

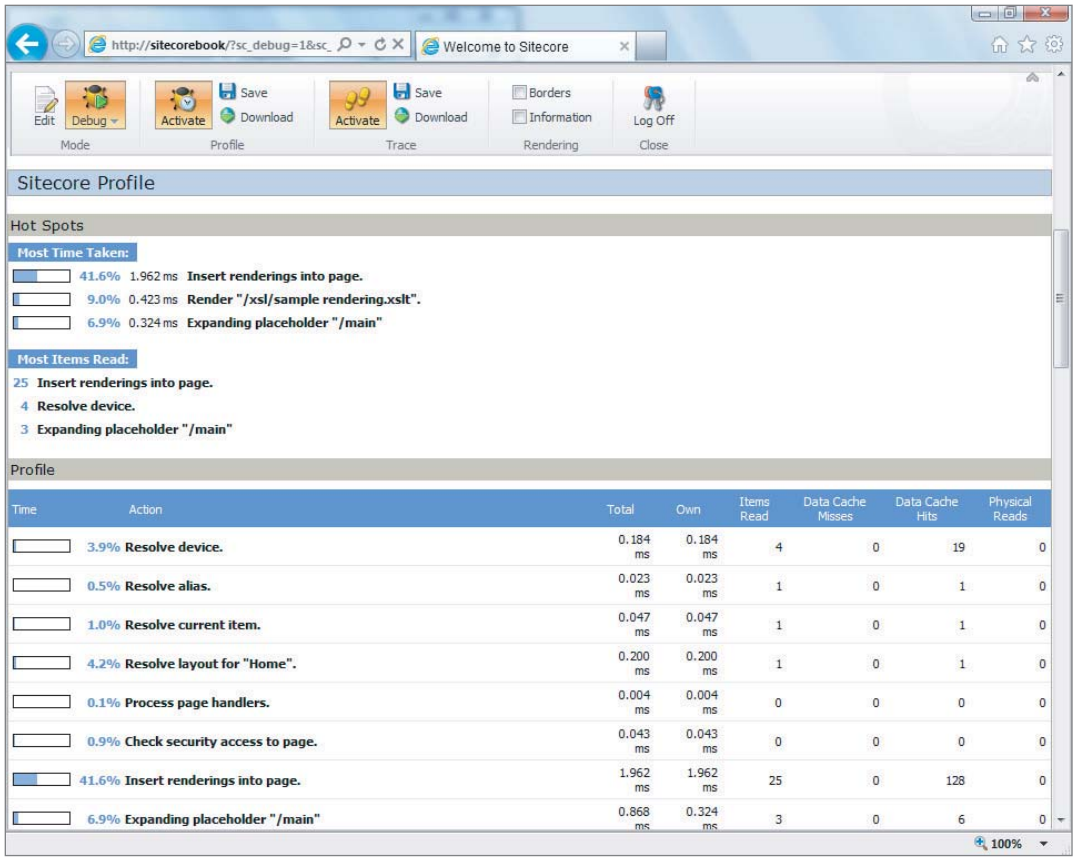


FIGURE 6-7

The trace indicates whether Sitecore invoked each presentation component or instead retrieved cached output generated previously by that component under equivalent processing conditions based on output caching settings (Vary By options) that you specify for that component.



Use health monitoring thresholds as described in the previous section to identify underperforming pages from the Sitecore logs. Use the rendering statistics pages described in a previous section and profiling features in the Sitecore debugger to identify underperforming components within those pages. Use tracing features in the Sitecore debugger to identify underperforming steps within those components. Use the Visual Studio debugger to debug your code as described in the following section.

Using the Rendering group in the debugger Ribbon, you can enable and disable Borders and Information. With Borders enabled, Sitecore adds visual borders around each presentation component. With Information enabled, Sitecore adds green triangle icons to each presentation component.

www.it-ebooks.info

Hover over these rendering information icons to see information about individual presentation components, which includes tabs to show performance profiles, cache settings, and output generated by individual controls. The example shown in Figure 6-8 shows the Details tab for the default Sample Rendering XSL.

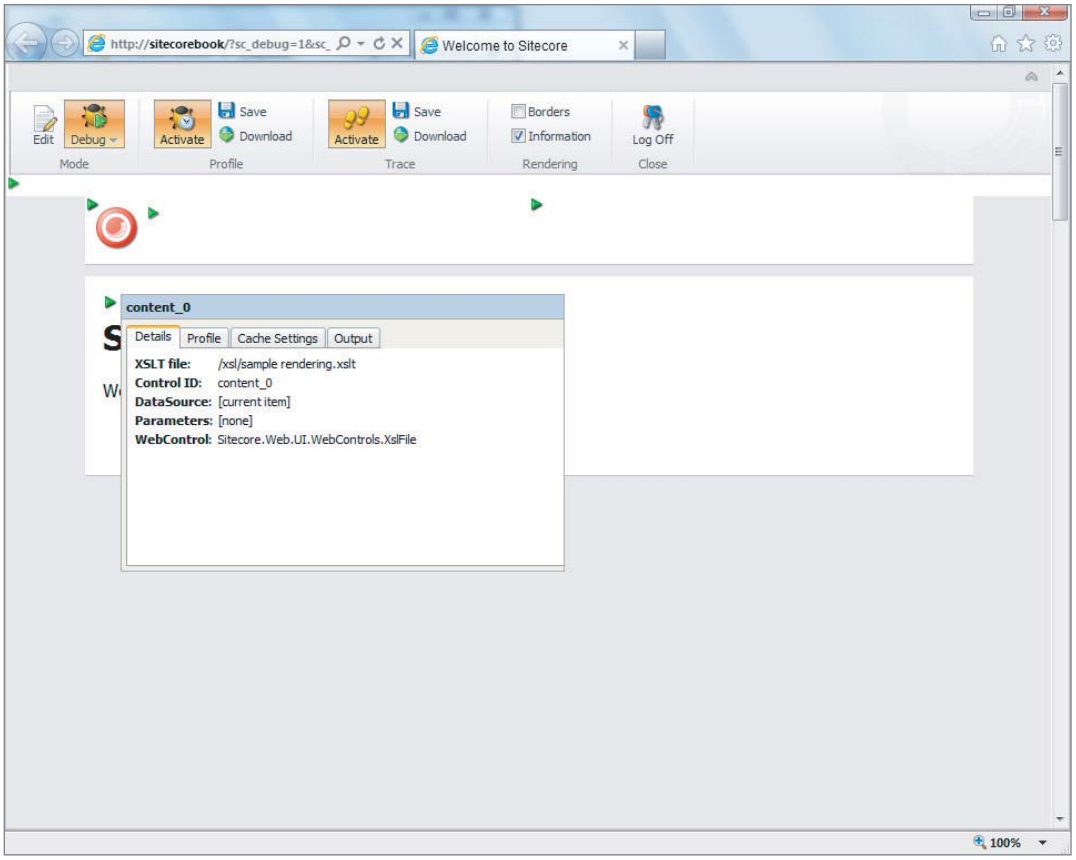


FIGURE 6-8



When showing rendering information, Sitecore never retrieves the output of a presentation component from the output cache, but instead executes each presentation component for each HTTP request. To see the impact of caching, you must clear the Information checkbox in the Rendering group.

If you use XSL renderings, note that Sitecore caches the objects used to apply the XSL transformation objects as well as the output of XSL transformations. A trace message such as the following indicates that the layout engine retrieved output for a control from cache:

```
Finished rendering "/xsl/sample rendering.xslt" (using cache).
```

www.it-ebooks.info

A trace message such as the following indicates reuse of a cached XSL transformation object:

```
Xslt file loaded from cache.
```

For more information about the Sitecore browser-based debugger, see my blog post at `http://bit.ly/n6x0QL`. For instructions to add a command to the Content Editor Ribbon to debug the selected item in the content database rather than always debug the home item in a publishing target database, see Chapter 7.

Debugging with Microsoft Visual Studio

You can use Microsoft Visual Studio to debug your Sitecore solution. You can even debug remote Sitecore solutions.



Follow the instructions in this section to debug with Internet Information Services rather than debug with the native web server built into Visual Studio.

To debug a Sitecore solution in Visual Studio, first request a page from the solution to ensure the ASP.NET application pool is active. Then follow this process:

1. Open the Sitecore project.
2. Clear the Show All Files button in Solution Explorer.



If you do not clear the Show All Files option before attempting to attach to the application pool process, Visual Studio may try to read all files under the document root when you start debugging, and either fail or seriously underperform. To address this issue if you happen to forget, restart Visual Studio and clear the Show All Files option.

3. Click the Debug menu, and then click Attach to Process. The Attach to Process dialog appears.
4. Select the Show Processes From All Users checkbox.
5. Select the Show Processes In All Sessions checkbox.
6. Select the appropriate instance of the `w3wp.exe` or `aspnet_wp.exe` process.

Alternatively, you can configure the start page for your Visual Studio project:

1. In Solution Explorer, expand your project and then double-click Properties. The project properties pane appears.
2. Click the Web tab.

www.it-ebooks.info

3. Under Servers, select Use Custom Web Server.
4. Under Use Custom Web Server, for Server Url, enter the URL of the home page of the Sitecore solution (such as `http://sitecorebook`), or the page to debug (such as `http://sitecorebook/mypage.aspx`).
5. Close the project Properties pane to save the changes.
6. To start debugging, press Ctrl+F5, or click the Debug menu and then click Start Debugging.

SPELUNKING SITECORE

This section describes various aspects of the Web.config file that you can investigate to understand, configure, extend, and troubleshoot Sitecore solutions. It then explains how you can disassemble compiled .NET assemblies (`.dll`) files to achieve an approximation of the source code used to compile the CMS and supporting products.

The Web.Config File

You can expand your knowledge of Sitecore and how you can extend the platform by reviewing the Web.config file, including both the root `/web.config` file and the example Web.config include files in the `/App_Config/Include` subdirectory of the document root of the IIS website hosting your Sitecore solution. Thanks to the configuration factory described in Chapter 7, you can substitute your own classes to replace the default implementations of numerous product features by simply updating the `/web.config` file, updating a Web.config include file, or best, adding your own Web.config include files.



The Sitecore developer community does not keep secret the fact that the default `/web.config` file is a monolithic monster, but in the end, you will likely appreciate the flexibility this solution allows. You should include at least the Web.config file and any include files that they modify or create in your Visual Studio project and in your source code management system (SCMS), although you may need to maintain multiple versions of these files for different environments. Always take a backup of the `/web.config` file and any Web.config include files before making any changes. Visual differencing tools such as WinMerge (`http://winmerge.org`), which you can use to compare your Web.config setup against the default, can help you troubleshoot issues that you may have introduced to your solution. Remember to use the `/sitecore/admin/ShowConfig.aspx` page (through a URL such as `http://sitecorebook/sitecore/admin/ShowConfig.aspx` on your solution) or the DetailedConfigReport (`http://bit.ly/wfMYhb`) Sitecore Shared Source project to evaluate your configuration after the application of Web.config include files.

www.it-ebooks.info

Web.Config Include Files

The contents of Web.config include files override (patch) the contents of the Web.config file itself. You can use Web.config include files:

- To override aspects of Sitecore's default configuration
- To simplify configuration, release management, system maintenance including upgrades, and other aspects of the system
- To separate configurations such as content management and content delivery or test and production
- To separate elements specific to certain solutions, modules, components, instances, and environments
- To otherwise to categorize configuration elements in separate files

Sitecore automatically applies Web.config include files (all files ending with the `.config` extension) in the `/App_Config/Include` subdirectory of the document root of the IIS website hosting the Sitecore solution, as well as those specified by `<sc.include>` elements in the actual `/web.config` file. The `/App_Config/Include` subdirectory contains examples (with the `.config.example` extension) for you to investigate and so that you can easily enable features. To enable the configuration in one of these example files, rename it with the `.config` extension. Web.config include files derive their structure from the `/configuration/sitecore` element of the root `/web.config` file. You can only patch elements within the `/configuration/sitecore` element of the Web.config file.

Each Web.config include file typically begins with a `<configuration>` element that uses the `xmlns:patch` attribute to map the `patch` namespace to support dynamic configuration. Elements within the `/configuration/sitecore` element such as a Web.config include file can then use the `patch:` prefix to indicate the position for their inclusion relative to existing elements in the actual `/web.config` file or to specify existing elements to replace, update, or delete.



Use of the `patch:` *prefix is just a convention. You can use any prefix that you map appropriately. If the location of elements defined in a Web.config include file relative to existing elements is unimportant, that Web.config include file does not need to define any such prefix.*

For example, to add a `/configuration/sitecore/sites/site` element before the default managed site named website:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <sites>
      <site patch:before="*[@name='website']" name="mysite" ... />
    </sites>
  </sitecore>
</configuration>
```

www.it-ebooks.info

To define a new managed site, the `patch:before` attribute in this example specifies an XPath expression that positions this new `<site>` element as a sibling before the existing `/configuration/sitecore/sites/site` with a value of `website` for the `name` attribute. For more information about managing multiple sites, see Chapter 9.



If you add a managed site as this example demonstrates, you typically need to update the `publish:end` *and* `publish:end:remote` *events defined in the Web .config file to clear the output cache for the new site after publishing. For more information about publishing and managing multiple sites, see Chapter 9. For a more comprehensive example Web.config include file for adding a managed site, see my blog post at* `http://bit.ly/rl9HcG`.

You can use the following tokens with the `patch:` prefix:

- `patch:before` — As shown above, insert a new element before the specified element
- `patch:after` — Insert an element after the element specified by the XPath expression
- `patch:instead` — Replace the element specified by the XPath expression
- `patch:delete` — Remove the element specified by the XPath expression
- `patch:attribute` — Define or replace the attribute specified by the XPath expression

For more information about Web.config include files, see my blog post at `http://bit.ly/qp0vps`.

The /configuration/sitecore Section

Because the Sitecore configuration factory uses the `/configuration/sitecore` section of the Web.config file to provide so much flexibility and extensibility, it is by far the largest section of the file; this book cannot describe every element it contains. I will only mention briefly those sections of the default root `/web.config` file that I think you are most likely to reconfigure or override.

- The `database` attribute of the `/configuration/sitecore` element specifies whether the instance uses SQL Server or Oracle.
- The `/configuration/sitecore/sc.variable` elements define variables used elsewhere in the Web.config file.
- The `/configuration/sitecore/prototypes` element configures a few specific types used by the configuration factory factored out to the `/App_Config/Prototypes.config` file.
- The `/configuration/sitecore/events` element defines Sitecore events and configures event handlers for those events.
- The `/configuration/sitecore/eventing` element configures management of events on remote instances.
- The `/configuration/sitecore/customHandlers` element maps handlers for URL prefixes used by Sitecore, such as the `~/media` prefix for Sitecore media, to tokens that match the path attributes of `/configuration/system.web/handlers/add` and `/configuration/`

www.it-ebooks.info

`system.webServer/httpHandlers/add` elements to specify the .NET classes to handle HTTP requests that begin with those prefixes. The `CustomHandlers` processor in the `httpRequestBegin` pipeline defined in the Web.config file aborts that pipeline if the URL matches any of these prefixes, minimizing server overhead for such requests.

- The `/configuration/sitecore/LinkDatabase` element specifies the class that implements the internal links management table and in which Sitecore database to maintain that table.
- The `/configuration/sitecore/TaskDatabase` element specifies the class that processes tasks scheduled in a Sitecore database and in which Sitecore database the instance maintains task information.
- The `/configuration/sitecore/IDTable` element configures the IDTable that you can use to map unique identifiers in external systems to Sitecore IDs, and vice-versa.
- The `/configuration/sitecore/pipelines` and `/configuration/sitecore/processors` elements configure pipelines. Sitecore user interfaces invoke pipelines defined within the `/configuration/sitecore/processors` element; pipelines defined within the `/configuration/sitecore/pipelines` manage application logic and HTTP requests.
- The `/configuration/sitecore/dataProviders` element configures Sitecore data providers, including those used for the file system and the Sitecore databases.
- The `/configuration/sitecore/workflowHistoryStores` element configures history management for the Sitecore workflow engine.
- The `/configuration/sitecore/databases` element configures the Sitecore databases, including features you might not expect, such as workflow providers and notification engines.
- The `/configuration/sitecore/archives` element configures Sitecore archive management.
- The `/configuration/sitecore/indexes` and `/configuration/sitecore/search` elements configure the Lucene search engine.
- The `/configuration/sitecore/authentication`, `/configuration/sitecore/httpAuthentication`, `/configuration/sitecore/authorization`, `/configuration/sitecore/rolesInRolesManager`, `/configuration/sitecore/configStores`, `/configuration/sitecore/domainManager`, and `/configuration/sitecore/switchingProviders` elements configure the Sitecore security infrastructure.
- The `/configuration/sitecore/accessRights` element configures access rights for the Sitecore security infrastructure. Each `/configuration/sitecore/accessRights/rights/add` element defines an access right. For information about adding custom access rights, see my blog post at `http://bit.ly/q9FQvl`.
- The `/configuration/sitecore/publishManager` element configures publishing operations, which invoke the `publish` pipeline defined in the Web.config file.
- The `/configuration/sitecore/itemManager` element configures a low-level item provider API.
- The `/configuration/sitecore/mediaPath` element configures the media path provider, which is responsible for determining the paths, names, and potentially other properties of new media items that users upload.

www.it-ebooks.info

- The `/configuration/sitecore/siteManager` element configures the site configuration provider, which abstracts information about the managed sites.
- The `/configuration/sitecore/linkManager` element configures link management, meaning dynamic URL generation.
- The `/configuration/sitecore/standardValues` element configures the standard values provider, which is responsible for retrieving standard values for data templates.
- The `/configuration/sitecore/fieldTypes` element configures internal link management features for the Sitecore data template field types.
- The `/configuration/sitecore/clientDataStore` element configures storage of data used by the Sitecore user interfaces.
- The `/configuration/sitecore/hooks` element configures initialization hooks triggered at application startup by the `LoadHooks` processor in the `initialize` pipeline defined in the Web.config file.
- The `/configuration/sitecore/scheduling` element of the Web.config file configures the Sitecore task scheduling engine, including the interval between polls for tasks to invoke as defined in a Sitecore database or as `<agent>` elements within this element. For more information about scheduling processes with Sitecore, see Chapter 7.
- The `/configuration/sitecore/settings` element contains a number of settings; review the comment above each to determine its purpose.



Rather than parsing Web.config directly, use methods in the `Sitecore.Configuration.Factory` *static class to access specific types configured in the Web.config file, and the* `Sitecore.Configuration.Settings` *class and descendant classes to access named settings.*

Reflecting on Sitecore

This section describes how you can use software products to disassemble Sitecore and other .NET assemblies (`.dll` files) to view an approximation of the source code compiled into those files. Disassembly can be invaluable to both troubleshooting and development, especially when extending Sitecore solutions. If you are not familiar with disassembly, this may be the most important section of this book for you to read.



Check your Sitecore end-user license before disassembling any Sitecore product. For any questions about your rights, contact your regional Sitecore representative. If you do not know whom to contact, see `http://bit.ly/AFaRil`*. In no case does any Sitecore license permit you to reverse engineer Sitecore software to develop competing products. Nothing in this section or elsewhere in this book authorizes you to reverse engineer Sitecore products except as indicated by Sitecore's end-user licenses.*

www.it-ebooks.info

Because the disassembler must guess at the original syntax from the IL (intermediate language) contained in the assembly, disassembly results in only an approximation of the original source code, not a line-by-line exact replica of that source. The disassembler cannot expose comments and other non-essential aspects of the original code removed by the compiler, which can also optimize code in ways you might not expect. For example, the compiler converts certain types of loops to `goto` statements with labels, and can convert method calls to inline code. Additionally, Sitecore intentionally obfuscates some assemblies to protect its intellectual property.

With Sitecore and ASP.NET, you can use a number of languages that can compile into .NET assemblies. Sitecore internally uses the C# language, and for a variety of reasons I generally recommend that customers use C#. Most example code for Sitecore, including everything in this book, uses C#. You may want to confirm that the software you choose to decompile can generate output in C#, and you should have at least a reading competency in C#. If necessary, you can compile C# samples to IL, and then decompile from IL to your preferred language.

The RedGate company (`www.red-gate.com`) distributes the .NET Reflector software product (`www.reflector.net`) for investigating .NET assemblies. This book describes RedGate's .NET Reflector (also called Reflector) because it is most familiar to me, but you can use any such tool that you prefer. Reflector uses reflection to interrogate the structure of compiled code, but much more important, it disassembles IL to source code that you could use to compile the interrogated assembly.

To investigate Sitecore assemblies, launch Reflector, and then use standard Windows features to open the assemblies (`.dll` files) that you wish to investigate from the `/bin` subdirectory of the document root of the IIS website hosting your Sitecore solution, starting with those that your Visual Studio project references. Most Sitecore developers primarily use APIs in the `Sitecore.Kernel.dll` assembly, which mostly contains back-end code. You could also use classes in `Sitecore.Client.dll`, which contains user interface code, and `Sitecore.Analytics.dll`, which contains code most relevant to the Digital Marketing System (DMS) described in Chapter 10.

When you open an assembly, Reflector may prompt you to open dependent assemblies as needed, or may open such assemblies automatically. For performance and usability, you may wish to limit the number of open assemblies. To close an assembly, select it in the tree and press the Delete key. For your convenience, when you close and restart Reflector, it reloads the assemblies that you had opened previously.

If your Sitecore solutions employ both .NET and XSL, in addition to reviewing The Sitecore XSL Presentation Component Reference (`http://bit.ly/qsVzvR`), you can learn more about XSL extension methods such as `sc:field()` by disassembling the `Sitecore.Xml.Xsl.XslHelper` class in the `Sitecore.Kernel.dll` assembly.



Sitecore typically translates XSL extension controls such as `<sc:text>` *in XSL renderings to call methods in the* `Sitecore.Xml.Xsl.XslHelper` *class.*

For convenience and your own education, you may wish to open additional assemblies, such as the `HtmlAgilityPack.dll` assembly, which contains the HTML Agility Pack (HAP) implementation described in Chapter 7. After you open assemblies, you can expand contained namespaces and

www.it-ebooks.info

types in the assembly tree frame on the left. Figure 6-9 shows Reflector after opening some common assemblies, navigating to the static constructor for the `Sitecore.Context` class, and double-clicking that member to disassemble (decompile) it.

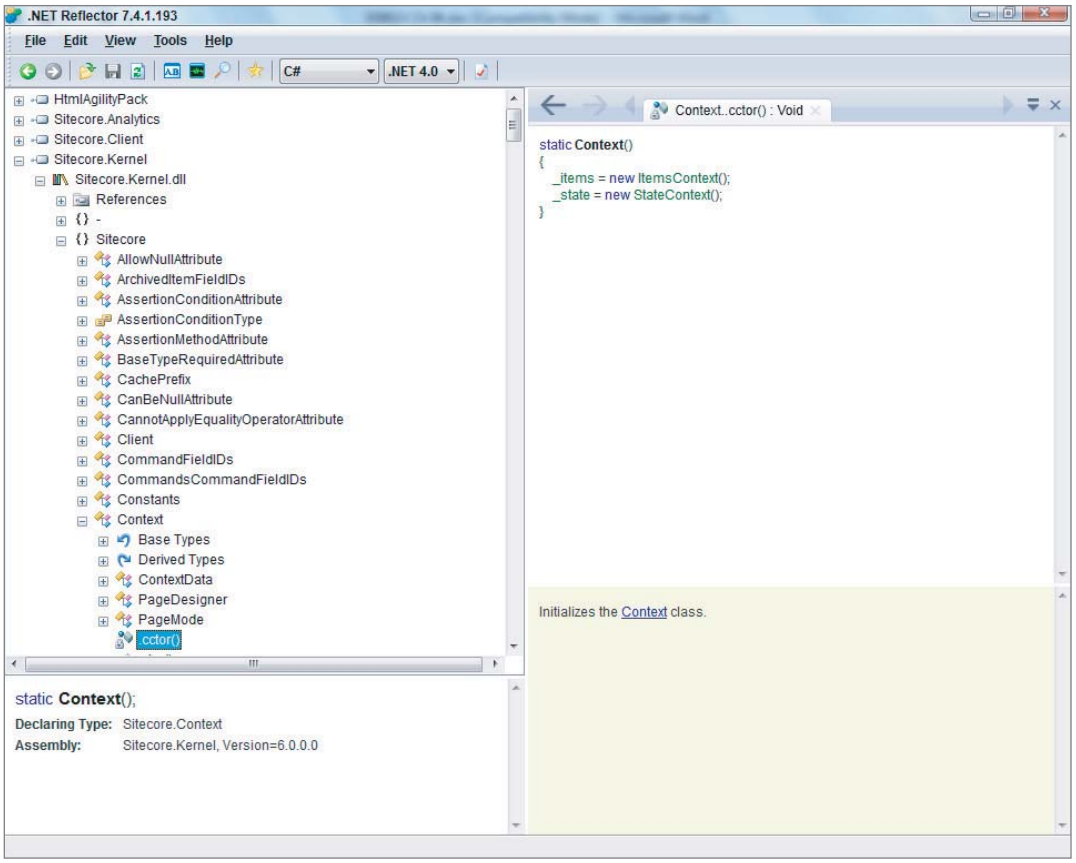


FIGURE 6-9



Always review the constructors for any classes that you disassemble. Otherwise, you can spend a significant amount of time investigating other methods and never find the logic that you actually need to examine.

Unless you have a stack trace or something else to identify the types that you need to investigate, you will likely need to search. Using Reflector, you can search for the name of a type, for the name of a member of a type, or for specific tokens in the code. If you know or can guess the name of the type (most commonly a class) but not the namespace that contains that type, or you do not care to navigate to the type, you can search for a type by name. If you know or can guess the name of a member (most commonly a method or property) but not the name of the type containing that

www.it-ebooks.info

member, you can search for a member by name. If you would like to search the code for a string, such as the name of a pipeline to see what invokes that pipeline, you can search for a token.

To initiate a search, press F3 to show the search bar. By default, Reflector searches for types with names that match the value you enter. To search for a member, press Ctrl+M (for member). For example, you can review the Web.config file to identify a setting, and then search for a member of the same name as that setting to locate the type that exposes that setting as a member. To search for a token in code, such as the name of a pipeline, press Ctrl+S (for string). To revert to searching for a type, press Ctrl+T (for type). Once you find an item to investigate further, double-click it in the search results, and then disassemble it and review the source code.

Another useful feature of Reflector enables you to find all usages of a type or member. To do so, select the type or member in the tree on the left, and then press Ctrl+R or right-click and select Analyze from the context menu that appears. In the Analyzer pane that opens, expand the type, and then expand Used By. As shown in Figure 6-10, a very large number of types use the `Sitecore.Context` class.

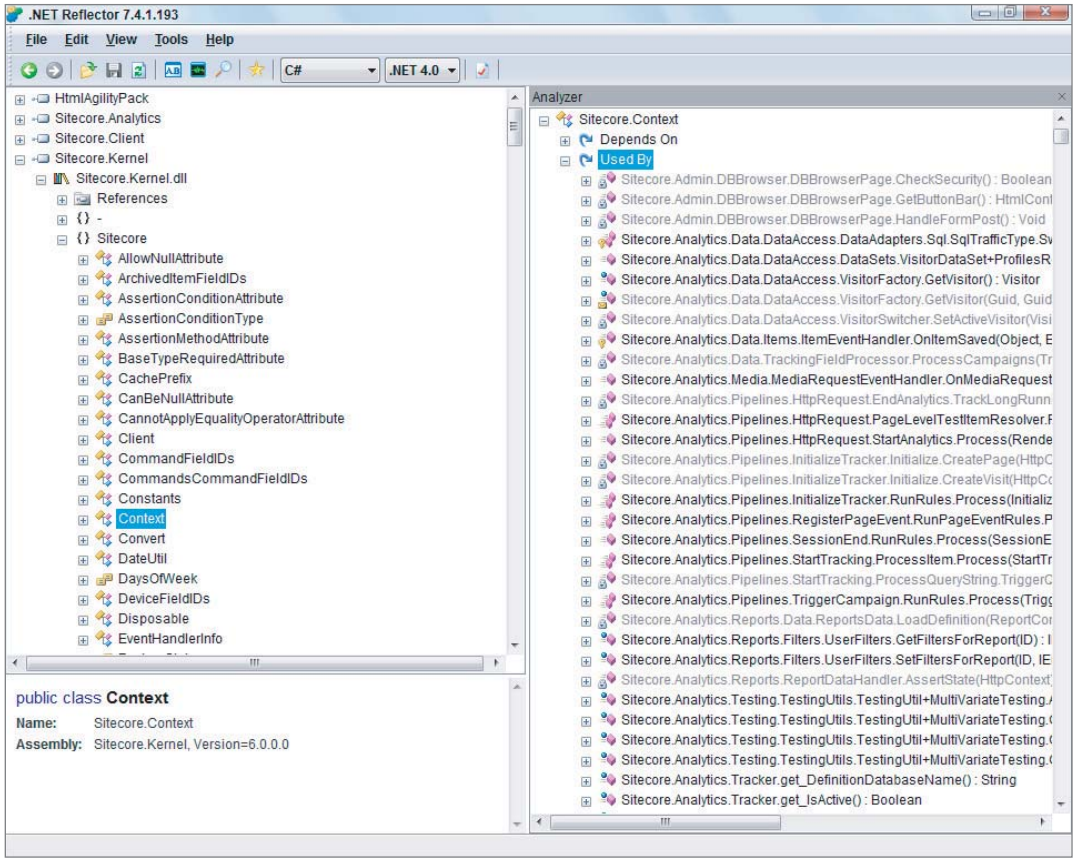


FIGURE 6-10

You can expand these types in the Analyzer pane as well, providing something like a list of all possible stack traces that can lead to that type or member.

www.it-ebooks.info

This section really just skims the surface of what you can achieve with Reflector, highlighting the features that I use most frequently. You can download add-ins from `http://bit.ly/yNd4VH` to extend Reflector with even more functionality. For maximal productivity, I recommend that you devote some time to reading about and working with this or another disassembly tool of your choice.

Reviewing Stack Traces

A stack trace indicates the hierarchy of method calls that lead to a point in the program. Especially in combination with reflection and disassembly, you can learn a great deal from a stack trace, and you can investigate stack traces to determine the cause of exceptions.

The static `System.Environment.StackTrace` property exposes the stack trace to the current line of code. The following example writes the stack trace to the point of execution to the system log, prefixed by an identifier of the class writing the log message:

```
Sitecore.Diagnostics.Log.Info(
  this + " : " + System.Environment.StackTrace,
  this);
```

More commonly, stack traces appear in the Sitecore logs to indicate where to begin your troubleshooting analysis. For information about special characters that can appear in stack traces, see my blog post at `http://bit.ly/ys9gi5`.

Browsing Sitecore Databases

Sitecore cannot prevent you from taking certain actions that can render your solution unusable, specifically the Sitecore browser-based user interfaces. For example, I recently helped someone on the Sitecore Developer Network forums with a problem that involved two data template field types with the same key. Somehow, in the Core database, someone had created a second definition item for a field type with the same name as one of the default field types. When they attempted to log in to any Sitecore browser-based graphical user interface (GUI), the system attempted to assemble a dictionary using the names of the field type definition items as keys, raising an exception due to the duplicated key. Because users could not log into the user interfaces due to this exception, they could not use the GUI to correct the issue. There are at least two ways to address this type of issue, both of which involve access to a Sitecore database using interfaces other than the browser-based GUIs.



Before manually altering a Sitecore database using either of the techniques described in this section, back up the database so that you can restore it if your efforts actually worsen the problem. Always be cautious when working directly with a Sitecore database, even with the tools described.

The first solution requires Visual Studio and the free Sitecore Rocks extension for Sitecore developers (`http://sitecorerocks.net`). Using these tools, you can connect to any Sitecore instance without accessing a Sitecore browser-based user interface, which enables you to correct issues using Sitecore Explorer. For more information about Sitecore Rocks, see my blog post at `http://bit.ly/z4HjP0`.

www.it-ebooks.info

The second solution does not require Visual Studio and Sitecore Rocks, which are not always available or may not be able to connect to the Sitecore instance experiencing the issue. You can use the database browser tool to investigate Sitecore databases using a very lightweight browser-based user interface. Sitecore does not support or document this tool, but provides it so that you can address issues such as that described previously, specifically under direction from the Sitecore customer service department. To access the database browser, navigate your web browser to `/sitecore/admin/dbbrowser.aspx` on the Sitecore instance, such as `http://sitecorebook/sitecore/admin/dbbrowser.aspx`.

TAKE HOME POINTS

Sitecore includes a number of features that you can use to improve the performance of your Sitecore solution. You can configure the contents of the prefetch caches to increase solution performance at the cost of initialization time. Use the cache administration page to tune size limits for your caches, or in more recent versions, set the `Caching.DisableCacheSizeLimits` setting in the Web.config file to `true` to disable cache size limits entirely. Use performance thresholds to identify underperforming pages and the rendering statistics page to identify underperforming presentation components.

Sitecore provides an almost infinite range of configuration options to support scalability and reliability. To scale your Sitecore solutions, start by separating the production content management environment from the production content delivery environment, and using dedicated database servers rather than local databases on the web servers. After implementing, optimizing, and load testing your solution, determine how many load-balanced instances you need to service your peak traffic load in the content delivery environment. Finally, consider whether you need to load balance the content management environment, and if you have a very large number of concurrent CMS users or a very large content repository, whether you should dedicate an instance in that environment solely for publishing.

An understanding of the Sitecore logging mechanism and the contents of the Sitecore logs can help in diagnosing problems with your web solution, as can familiarity with the contents of the Web .config file. Most Sitecore developers consider disassembly a critical aspect in their ability understand and troubleshoot the workings of a Sitecore solution, but also use this technology to further their ability to extend the system.

www.it-ebooks.info

www.it-ebooks.info

7

Extending and Integrating Sitecore

WHAT'S IN THIS CHAPTER?

- Using the configuration factory
- Implementing .NET extension methods
- Understanding the Sitecore user interface framework
- Configuring ribbons and item context menu commands
- Enhancing user interfaces
- Employing the rules engine
- Scheduling processes
- Initializing with hooks
- Handling events
- Processing pipelines
- Extending the Sitecore Page Editor

No CMS vendor can provide software that includes all features that every organization could possibly require. Even you cannot completely define your current requirements, let alone your future requirements. Fortunately, Sitecore's flexible platform architecture enables you to quickly implement almost any feature you could need and adapt to almost any requirements as they emerge. In no particular order, Sitecore provides for at least the following potential points of extension:

- **Pipelines —** Pipelines define processes as sequences of discrete steps implemented as pipeline processors.

www.it-ebooks.info

- **Command templates** — These templates enable you to implement code to help users create new items.
- **Validation** — Validation enables you to implement custom logic to define data validity.
- **The rules engine** — This engine enables you to configure logic to apply under various circumstances, including conditional rendering to support personalization, multivariate testing, and other advanced features.
- **Scheduled processes** — The capability to schedule processes enables you to invoke logic at predefined periods.
- **Hooks** — Hooks provide for application initialization logic.
- **Event handlers** — Event handlers support extension before, during, and after various operations.
- **Custom editors** — Editors enable you to embed user interfaces in Sitecore items.
- **The ribbon and commands** — The ribbon and commands enable you to add functionality to the Content Editor.
- **Custom applications** — Create your own applications to do almost anything within the Sitecore desktop.
- **Data providers** — These providers represent data in external systems as Sitecore items.
- **The configuration factory** — The configuration factory enables you to configure and override numerous Sitecore subsystems.
- **Sitecore Rocks** — Sitecore Rocks extends Visual Studio with features for Sitecore developers, and Sitecore Rocks itself supports extensions.
- **Override User Interfaces** — You can replace markup and code-behind in XML and Web Forms (`.aspx` files) files that define user interfaces (UIs) included with the CMS.
- **Page Editor commands** — These commands add features to elements, fields, renderings, and other presentation components.
- **Context menus** — These menus appear for items in the content tree and support any number of custom entries.
- **Extension classes** — This .NET technology enables you to extend classes provided by Sitecore with custom methods.
- **Web services** — Web services, including the Good Old Web Service provided with the ASP .NET CMS and the Hard Rock Web Service provided with Sitecore Rocks, provide for extension from remote systems.
- **ASP.NET security providers** — These providers support integration with external systems for authentication, role, and user profile management, such as using a customer relationship management (CRM) system for authentication.
- **The Rich Text Editor** — This editor supports custom toolbar buttons and other features.
- **Content Editor warnings** — These warnings enable you to dynamically add notification for items in the Content Editor.

www.it-ebooks.info

- **Page Editor commands —** These commands enable you to add features around markup, fields, presentation controls, and placeholders in the Page Editor.
- **Custom field types and the IFrame field type —** These field types enable you to embed your own controls as Sitecore data template fields.

Whenever you extend Sitecore, in addition to considering the potential for incorrect interaction with existing components, you must consider performance. Many types of components run extremely frequently, sometimes in multiple contexts simultaneously. For example, Sitecore invokes `item:saved` event handlers every time a user or API call updates an item. Event handlers, pipeline processors, validators, and other components should return immediately if they do not apply to the arguments passed to the component or other current processing conditions.

For more information about the extensibility of the Sitecore ASP.NET CMS, see my blog posts at `http://bit.ly/refgxB` and `http://bit.ly/q9JGV8`.

DETERMINING TYPES WITH THE CONFIGURATION FACTORY

The Sitecore configuration factory works as a dependency injection container to configure types to insert dynamically into the running application. Dependency injection determines at runtime the software components that implement specific features. The Sitecore configuration factory works like a provider for .NET types. You can use the configuration factory to override the .NET types that Sitecore uses to implement specific features, to pass parameters to those types, and to otherwise configure those types.

The Sitecore configuration factory uses the Web.config file, as well as definition items in the Sitecore databases. To get an idea of how to configure Sitecore, first investigate the Web.config file.



Remember to use the `/sitecore/admin/showconfig.aspx` *page to investigate configuration after the application of Web.config include files.*

While pipelines are just one example of components that you can address with the configuration factory, they provide a good example of this functionality. Sitecore implements numerous features with pipelines, which a subsequent section of this chapter describes in detail. These include the `httpRequestBegin` pipeline described previously in this book, such as in Chapter 5. Sitecore invokes the `httpRequestBegin` pipeline when it begins processing each HTTP request. The configuration factory reads pipeline definitions from the Web.config file. The `/configuration/sitecore/pipelines/httpRequestBegin` element in the Web.config file defines the `httpRequestBegin` pipeline using a series of nested `<processor>` elements. Each `<processor>` element specifies a .NET class that contains a method that implements that step in the pipeline. When Sitecore invokes a pipeline, it invokes those methods in the order of the elements in the Web.config file that define the pipeline. You can update the Web.config file (or preferably, add Web.config include files) to:

- Override the types used to implement processors in a pipeline
- Remove processors from a pipeline

www.it-ebooks.info

- Add custom processors to a pipeline
- Change the order of processors in a pipeline
- Declaratively pass parameters to pipeline processors

The configuration factory uses type signatures to identify the .NET types that implement features. A type signature consists of a namespace, a type name (such as the name of a class), and the name of the assembly containing that class (following a comma (,) and without the `.dll` extension):

```
NamespaceName.TypeName, AssemblyName
```



In some cases, Sitecore uses a separate field or attribute for the assembly name; and in some cases, separate fields or attributes for the namespace and the type name.

If an element within the Web.config file specifies a type attribute, the text of any child `<param>` elements represent parameters to the constructor for that type. Other children typically represent the names of properties of the type. A child element with a hint attribute often indicates a list or a specific method to invoke on the type after instantiation.



In some versions of Sitecore, each repeated element in a Web.config include file must have a unique value. For example, in place of the sequence `<element />` `<element />`*, you could use a convention such as* `<element unique="1" />` `<element unique="2" />` *to ensure that each element has a unique value in the attribute named* `unique`.

In some cases, the configuration factory returns an object that works like another configuration factory. For example, as described in Chapter 5, you cannot use the configuration factory directly to specify the control types that Sitecore binds to placeholders to implement sublayouts. The default value of the `type` attribute of the `/configuration/sitecore/renderingControls/control` element in the Web.config file specifies the `Sitecore.Web.UI.SublayoutRenderingType` class. The configuration factory instantiates an instance of that type, which works as a factory to return instances of the actual type that implements sublayouts (`Sitecore.Web.UI.WebControls.Sublayout` by default). You can override the type specified by such elements to return instances of your overrides for the types returned by factories of this kind.

In addition to the `/web.config` file and Web.config include files, for some features Sitecore uses fields in items in databases to specify types and parameters. Some Sitecore products, such as Sitecore E-Commerce products, use the Microsoft Unity Application Block for dependency injection. For more information about dependency injection in general, see `http://bit.ly/1511WG`. For more information about the Microsoft Unity Application Block, see `http://unity.codeplex.com`. For more information about dependency injection with Sitecore, see my blog post at `http://bit.ly/r1rwBr`. For more information about the Sitecore configuration factory, see my blog post at `http://bit.ly/n6mr29`.

www.it-ebooks.info

EXTENDING SITECORE CLASSES WITH EXTENSION METHODS

You can implement .NET extension methods to extend Sitecore classes with your own methods. To add an extension method for a class, create a static class containing the extension methods, where the first parameter to the method uses the `this` keyword to designate the class to extend. For example, the following extends the `Sitecore.Data.Items.Item` class that represents an item in a Sitecore database with a method named `MethodName()`:

```
public static class ItemExtension
{
  public static void MethodName(this Sitecore.Data.Items.Item me)
  {
    // method body accesses the argument
  }
}
```

Developers frequently extend the `Sitecore.Sites.SiteContext` class used to represent a managed website with methods that access custom attributes of the `/configuration/sitecore/sites/site` elements in the Web.config file.

The following example enhances the `Sitecore.Sites.SiteContext` class to support a custom `twitter` attribute for each `/configuration/sitecore/sites/site` element in the Web.config file. You can use the `twitter` attribute to specify the name of a Twitter account associated with each managed site.




```
namespace SitecoreBook.Sites
{
  using System;

  public static class SiteContextTwitter
  {
    public static string GetTwitterAccount(this Sitecore.Sites.SiteContext site)
    {
      Sitecore.Diagnostics.Assert.IsNotNull(site, "site");
      return site.Properties["twitter"] ?? String.Empty;
    }
  }
}
```

SiteContextTwitter.cs

To use the extension method, add a `using` statement for the namespace containing the extension class, and then you can invoke the method as you would any other method on the extended class. For example, use the following to determine the Twitter account associated with the context site:

```
using Sitecore.Book.Sites
...
Sitecore.Sites.SiteContext siteContext = Sitecore.Context.Site;
string twitterAccount = siteContext.GetTwitterAccount();
```

You can use this approach to define an attribute for the Google Analytics account ID, error pages, or any other property of each managed website.

www.it-ebooks.info

For more information about implementing extension methods with Sitecore, see the Library (`http://bit.ly/vhoFOK`) and the PageNotFound Sitecore Shared Source projects (`http://bit.ly/vIoNwP`), as well as my blog posts at `http://bit.ly/ntowNQ` and `http://bit.ly/r9Wsgw`.

LEVERAGING THE SITECORE USER INTERFACE FRAMEWORK

You can extend the Sitecore user interface using commands, ribbons, context menu entries, and Web Forms (`.aspx` files), and using Sitecore-specific user interface framework components. In fact, the Sitecore user interface framework probably deserves at least one book of its own.



Always consider the effect of the context site on other aspects of the Sitecore context, such as the context database. When you access a published site, the context site is that published site and the context database is a publishing target database such as the Web database associated with that site. When you access a Sitecore user interface such as the Content Editor or the browser-based desktop, the context site is the site named `shell`*, the context database (*`Sitecore.Context.Database`*) is the Core database (which controls the Sitecore applications presented by the site named shell through which you access those interfaces), and the content database (*`Sitecore.Context.ContentDatabase`*) is the Master database. When you select a database using the database icon in the lower-right corner of the desktop, you change the content database, not the context database. In Sitecore user interfaces such as Preview and the Page Editor, the context site is the published site, but the context database is the Master database. In the browser-based debugger, the context site is the published site and the default context database is the publishing target database. This distinction is especially important to Sitecore user interface components. The context database configures the application; the content database contains the data managed by the context user.*

Introducing Sitecore Commands

Sitecore commands implement features of the CMS. When you click a button in the ribbon in the Content Editor, such as the Delete button in the Operations group on the Home tab, Sitecore invokes a command that implements the deletion function. You can think of Sitecore commands as circumstances — user actions, timeouts, scheduled processes or otherwise — that require the system to pass zero or more parameters to some facility in Sitecore that implements an operation that may or may not present a user interface to interact with the user.

Sitecore uses entries in the Core database to define CMS user interfaces including the ribbon visible in the Content Editor, but also ribbons and other features of other browser-based applications. Specific items in the Core database contain metadata about buttons in the ribbon, including command codes for Sitecore to issue when CMS users click those buttons. Sitecore uses the `/App_Config/Commands.config` file to map those command codes to the .NET classes that implement those features. Some applications register commands dynamically; not all commands appear in the `/App_config/Commands.config` file.

www.it-ebooks.info

In the Core database, the `/sitecore/content/Applications/Content Editor/Ribbons` item contains strip, group, and command definition items to configure the Content Editor ribbon. For example, the `/sitecore/content/Applications/Content Editor/Ribbons/Strips/Home` item in the Core database defines the Home tab that appears in the Content Editor. The `/sitecore/content/Applications/Content Editor/Ribbons/Strips/Home/Operations` child of that item defines the Operations group that appears on the Home tab (in this context, a strip contains the features shown when you click a tab). The Reference field in the Data section of the Operations item specifies the `/sitecore/content/Applications/Content Editor/Ribbons/Chunks/Operations` item that contains definition items for the commands that appear in the Operations group. The `/sitecore/content/Applications/Content Editor/Ribbons/Chunks/Operations/Delete` command definition item contains data about the Delete command that appears in the Operations group on the Home tab. The Click field of that item contains the value `item:delete`, which the `/App_Config/Commands.config` file maps to the `Sitecore.Shell.Framework.Commands.Delete` class that implements item deletion logic.

The `Sitecore.Shell.Framework.Commands.Delete` class inherits from the `Sitecore.Shell.Framework.Commands.Command` abstract base class for commands. The `Execute()` method in that class contains the command implementation. The Menu field in the Data section of the `/sitecore/content/Applications/Content Editor/Ribbons/Chunks/Operations/Delete` item specifies the `/sitecore/content/Applications/Content Editor/Menues/Delete` [sic] item, which defines the menu that can appear under the Delete command when you select an item that has children. The children of that item define commands that appear on that menu.

Some groups on the ribbon expose *dialog launchers*, which appear as small square icons after the names of specific groups and open additional dialogs. For example, the Sorting group on the Home tab provides a dialog launcher that lets you choose a child sorting rule for the selected item. To implement a dialog launcher, in the Core database, in the group definition item (such as `/sitecore/content/Applications/Content Editor/Ribbons/Chunks/Operations` in the Delete example provided in the previous paragraph, in the Click field (found in the Data section), enter a command code that the `/App_Config/Commands.config` class maps to the class that implements that command.

Extending Ribbons

For many reasons, including familiarity, capability, manageability, consistency, and usability, Sitecore browser-based user interfaces such as the Content Editor borrow the ribbon user interface paradigm from the Microsoft Windows operating system.

A ribbon consists of tabs. If a ribbon consists of only one tab, the tab names may not appear in the ribbon. Clicking a tab shows a strip consisting of groups containing related commands that you can click to invoke specific functions. Groups are simply visual containers for commands; you can use the same group on multiple strips and the same command in multiple groups. Some commands display an interface, while others update data in the background and may subsequently refresh the user interface. Some commands expose drop-down menus, while some trigger flyouts, modal dialogs, or other user interfaces.



Some aspects of Sitecore may refer to groups on the ribbon as chunks.

www.it-ebooks.info

The ribbon that appears at the top of the Content Editor includes a Sitecore logo, which is actually a button that you can click to access configuration options, as well as information about the current item.

To override one of the default Sitecore commands, follow these steps:

1. Create a class that inherits from the `Sitecore.Shell.Framework.Commands.Command` abstract base class or from the default implementation of the command.
2. Implement the `Execute()` method in the new class.
3. Replace the type specified by the `type` attribute of the `/commands/command` element in the `/App_ConfigCommands.config` file for which the value of the name attribute indicates the command to override.

Here are the steps to create a tab that appears only for items based on specific data templates:

1. Select the Core database using the database the icon at the lower-right corner of the desktop, and then launch the Content Editor.
2. Select the `/sitecore/content/Applications/Content Editor/Ribbons/Contextual Ribbons` item.
3. Duplicate one of the children of the `/sitecore/content/Applications/Content Editor/Ribbons/Contextual Ribbons` item to create a new strip.
4. Update the new item and its descendants to define the strip.
5. Select the Master database using the database icon at the lower-right corner of the Sitecore desktop, and then launch the Content Editor.
6. Select the data template to which the new tab should apply.
7. Select the Configure tab, and then click Contextual Tab in the Appearance group. The Contextual Tab dialog appears as shown in Figure 7-1, which presents this dialog as it appears for the `Media/Versioned/File` data template that specifies a custom tab containing commands for working with media items.
8. Select your custom tab, which updates the `__Ribbon` (`Sitecore.FieldIDs.Ribbon`) field defined in the Appearance section of the standard template, and then click Open. The Contextual Tab dialog disappears and you return to the Content Editor.

You can implement the `QueryState()` method in your command classes to control whether commands are hidden or visible, or disabled or enabled. The `QueryState()` method returns one of the entries from the `Sitecore.Shell.Framework.Commands.CommandState` enumeration, which includes values for `Disabled` (grayed out), `Enabled` (the default), and `Hidden` (not visible in the ribbon). You can validate that the `Items` array of the `Sitecore.Shell.Framework.Commands.CommandContext` argument passed to the `QueryState()` method contains a single item, and then return a value depending on properties of that item. For example, to disable your command for clones:

```
public override Sitecore.Shell.Framework.Commands.CommandState QueryState(
  Sitecore.Shell.Framework.Commands.CommandContext context)
{
  if (context != null
    && context.Items.Length == 1
```

www.it-ebooks.info

```
      && context.Items[0] != null
      && context.Items[0].IsClone)
    {
      return Sitecore.Shell.Framework.Commands.CommandState.Disabled;
    }

    return base.QueryState(context);
  }
```

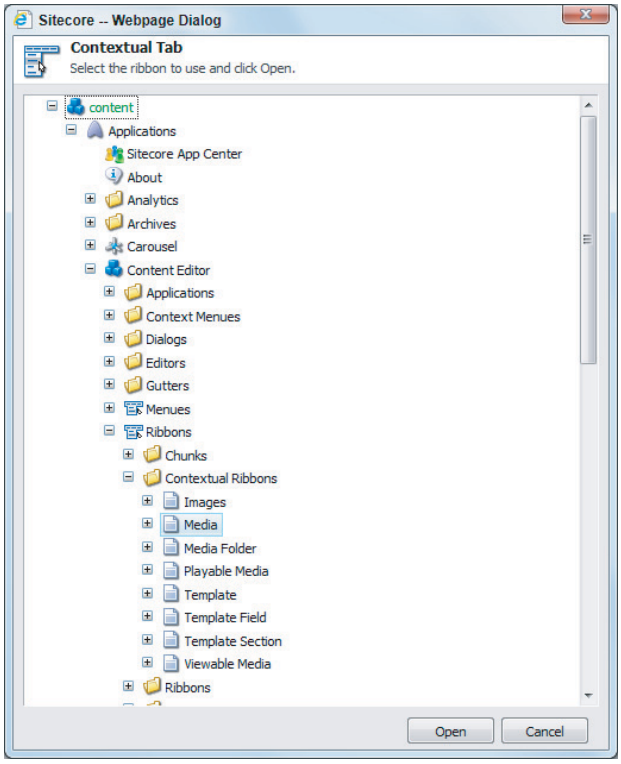


FIGURE 7-1



Showing and hiding commands on the ribbon depending on the selected item can distract users by changing the ribbon as they navigate from item to item. Instead of hiding commands, disable them, or create a custom tab for the command and associate that tab with the data template(s) to which that command applies.

To define which of the CMS users can access each command, Sitecore defines default access rights for the Sitecore client roles to the command definition items and the group definition items that contain those commands. To control access to features without increasing any administrative burden, add users to Sitecore client roles. If the default access rights do not meet your requirements, you can create custom client roles, set access rights for those roles, and add users (and nested roles) to those roles.

www.it-ebooks.info

You can rearrange, duplicate, remove, and add commands to a ribbon. For example, you can add a command to the ribbon in the Content Editor to debug the selected item in the content database. For more information about the Sitecore debugger, see Chapter 6. By default, the Debug command on the Sitecore menu in the desktop simply opens a new browser window with a URL that contains query string parameters to enable all debugging features. Under the default configuration, the debugger runs against the home item of the default managed site named `website` in the default publishing target database named Web. The custom implementation of the Debug command described in this section is a bit more complicated, as it must determine whether to debug a specific item in a specific database, and if so, which item and database to debug.

I like this example for a number of reasons. One of the best ways to enhance existing functionality is to create a class that inherits from the existing implementation of a feature, and then override methods of that class as required. Like most enhancements to Sitecore, instead of starting from scratch, the solution described in this section leverages something already provided by Sitecore: not just the debugger itself, but the `system:debug` command used by the Debug button on the Sitecore menu, defined by the `/sitecore/content/Documents and settings/All users/Start menu/Right/Debug` item in the Core database.

By default, the `type` attribute of the `/configuration/command` element named `system:debug` in the `/App_Config/Commands.config` file indicates that this command uses the `Sitecore.Shell.Framework.Commands.System.Debug` class in the `Sitecore.Kernel.dll` assembly. This class inherits from the `Sitecore.Shell.Framework.Commands.Command` abstract base class that defines a contract for commands.

The `Execute()` method of the `Sitecore.Shell.Framework.Commands.System.Debug` class contains logic to invoke the debugger. The default implementation always opens a new browser window using the same protocol (HTTP or HTTPS) and hostname of the current browser window, no path, and the following query string parameters to enable debugging and specific debugging features:

- `sc_debug` — Disable or enable Sitecore debugging
- `sc_prof` — Disable or enable page profiling
- `sc_trace` — Disable or enable page tracing
- `sc_ri` — Hide or show rendering information

After you click Debug on the Sitecore menu in the desktop and Sitecore opens the debugger, you can set each of these query string parameters to `0` or `1` to disable or enable the related feature, respectively, or you can use the corresponding features in the ribbon that appears in browser window hosting the debugger.

Under the default configuration, when Sitecore processes the HTTP request from this new browser window, the request matches the `/configuration/sitecore/sites/site` element named `website` in the Web.config file, and the `SiteResolver` processor in the `httpRequestBegin` pipeline sets the context site based on that site definition. You can add the `sc_site` query string parameter set to the name of a `/configuration/sitecore/sites/site` element in the Web.config file to specify an alternate site to debug.

The `database` attribute of the `/configuration/sitecore/sites/site` element named `website` in the Web.config file specifies the publishing target database named Web by default, so the

www.it-ebooks.info

`DatabaseResolver` processor in the `httpRequestBegin` pipeline sets the context database to that database. You can add the `sc_database` query string parameter set to the name of an alternate database to debug.

Because the URL in the browser window that the Debug command opens does not include a path (only the query string parameters such as those used to enable the debugger and its specific features as listed previously), the `ItemResolver` processor in the `httpRequestBegin` pipeline sets the context item to the home item defined for the context site (`/sitecore/content/home` by default). To specify the ID of the item to debug, you can add a path or the query string parameter named `sc_itemid`. Alternatively, you can use a browser bookmark or otherwise navigate to the page to debug.

In other words, the debugger always opens to the home page of the default published site using the published database (the Web database by default). It may be convenient to click a command to debug the item selected in the Content Editor using the content database (the Master database by default, or whichever database you select in the desktop). The custom implementation shown in Listing 7-1 adds the `sc_itemid`, `sc_database`, and `sc_lang` query string parameters to specify the item, database, and language to debug, respectively.



Available for download on Wrox.com

LISTING 7-1: Debug custom implementation, Debug.cs

```
using System;

namespace SitecoreBook.Shell.Framework.Commands.System
{
  [Serializable]

  // implements the debug command on the Sitecore menu of the desktop
  // and in the ribbon of the Content Editor
  public class Debug : Sitecore.Shell.Framework.Commands.System.Debug
  {
    public override void Execute(
      Sitecore.Shell.Framework.Commands.CommandContext context)
    {
      // validate arguments and processing context
      Sitecore.Diagnostics.Assert.ArgumentNotNull(
        context,
        "context");
      Sitecore.Diagnostics.Assert.IsNotNull(
        Sitecore.Context.ClientPage,
        "ClientPage");
      Sitecore.Diagnostics.Assert.IsNotNull(
        Sitecore.Context.ClientPage,
        "ClientResponse");

      // ensure the new browser window authenticates as the current CMS user
      // (in case the user formerly previewed as another user)
      Sitecore.Publishing.PreviewManager.RestoreUser();

      // open the new browser window
      Sitecore.Web.UI.Sheer.SheerResponse.Eval(
        "window.open('" + this.GetDebuggingUrl(context) + "', '_blank');");
```

continues

www.it-ebooks.info

LISTING 7-1 *(continued)*

```
}

// construct a URL to launch the debugger
private string GetDebuggingUrl(
  Sitecore.Shell.Framework.Commands.CommandContext
  context)
{
  // whether to use the sc_lang query string parameter to specify the language
  bool includeLanguage = Sitecore.Links.LinkManager.LanguageEmbedding !=
    Sitecore.Links.LanguageEmbedding.Never;

  // URL of the debugging window defaults to home page of managed site
  Sitecore.Text.UrlString url = new Sitecore.Text.UrlString("/");

  // enable all debugging options
  url.Add("sc_debug", "1"); // enable the debugger
  url.Add("sc_prof", "1");  // enable profiling
  url.Add("sc_trace", "1"); // enable tracing
  url.Add("sc_ri", "1");    // enable rendering information

  // if the user has selected an item, ensure they have saved,
  // then debug the item in the database and language associated with that item
  if (context != null
    && context.Items != null
    && context.Items.Length > 0
    && context.Items[0] != null)
  {
    Sitecore.Context.ClientPage.ClientResponse.CheckModified(false);
    Sitecore.Data.Items.Item item = context.Items[0];
    url.Add("sc_database", item.Database.Name);
    url.Add("sc_itemid", item.ID.ToString());

    if (includeLanguage)
    {
      url.Add("sc_lang", item.ID.ToString());
    }
  }

  // if the user has not selected an item,
  // if there is a content database, debug that database
  // using the content language
  else if (Sitecore.Context.ContentDatabase != null)
  {
    url.Add("sc_database", Sitecore.Context.ContentDatabase.Name);

    if (includeLanguage)
    {
      url.Add("sc_lang", Sitecore.Context.ContentLanguage.Name);
    }
  }

  // return a URL to open the debugger
  return url.GetUrl();
```

www.it-ebooks.info

```
    }

    public override Sitecore.Shell.Framework.Commands.CommandState QueryState(
      Sitecore.Shell.Framework.Commands.CommandContext context)
    {
      // if the user has selected at least one item
      if (context.Items != null
        && context.Items.Length > 0
        && context.Items[0] != null)
      {
        // if that item does not specify a layout for any device, disable this command
        if (!this.HasLayoutForAnyDevice(context.Items[0]))
        {
          return Sitecore.Shell.Framework.Commands.CommandState.Disabled;
        }
      }

      return base.QueryState(context);
    }

    // returns true if the item specifies a layout for any device
    protected bool HasLayoutForAnyDevice(Sitecore.Data.Items.Item item)
    {
      Sitecore.Diagnostics.Assert.IsNotNull(item, "item");

      // evaluate each device in the database containing the item
      foreach (Sitecore.Data.Items.DeviceItem compare
        in item.Database.Resources.Devices.GetAll())
      {
        // if the item specifies layout details for that device, return true
        if (item.Visualization.GetLayout(compare) != null)
        {
          return true;
        }
      }

      // layout details for the item do not specify a layout for any device
      return false;
    }
  }
}
```

The custom `system:debug` implementation shown in Figure 7-1 accounts for the content language, but the default implementation does not (neither accounts for the selected version within that language). To restore the default logic for managing the language accessed by the debugger, comment out the two lines that set the `sc_lang` query string parameter from the sample code provided in Listing 7-1:

```
url.Add("sc_lang", item.ID.ToString());
...
url.Add("sc_lang", Sitecore.Context.ContentLanguage.Name);
```

The `Sitecore.Shell.Framework.Commands.System.Debug` class does not implement the `QueryState()` method, which controls whether to show, hide, enable, or disable the command in the ribbon. Hence, it uses the default implementation of this method in the `Sitecore.Shell`

www.it-ebooks.info

.Framework.Commands.Command class, which always returns Sitecore.Shell.Framework.Commands.CommandState.Enabled. Any user with read access to the item in the Core database that defines the Debug command can attempt to debug any item, which could generate an error if the item does not contain layout details. For demonstration purposes, the example included in Listing 7-1 implements the QueryState() method to return Sitecore.Shell.Framework.Commands.CommandState.Disabled if the item selected in the Content Editor does not specify a layout for any device. You can implement your own logic in the QueryState() method, or you can remove this method from your command implementation. As an alternative to implementing the QueryState() method to disable the command, you could add logic to the Execute() method to ask users whether they meant to debug the home item, similar to the default system:webedit command implementation that launches the Page Editor.

To add the command to the Go To group on the Developer tab in the Content Editor, follow these steps:

1. Set the type attribute of the /configuration/command element named system:debug in the /App_Config/Commands.config file to the signature of your implementation of this command.
2. Select the Core database in the Sitecore desktop, and then launch the Content Editor.
3. Insert a command definition item under the /sitecore/content/Applications/Content Editor/Ribbons/Chunks/Goto item named Debug using the System/Ribbon/Small Button data template.
4. Set the Header field in the Data section to **Debug**, the Icon field to Software/16x16/debug_run.png, the Click field to system:debug, and the Tooltip field to **Start the debugger**.
5. Use the database selection icon in the lower-right corner of the Sitecore desktop to select the Master database, and then launch the Content Editor.
6. If the Developer tab is not visible, right-click the tab strip, and then select Developer.
7. Select an item that does not contain layout details, and then select the Developer tab. The Debug command appears disabled in the Go To group.
8. Select an item that contains layout details for any device. The Debug command appears enabled.
9. Click the Debug command. The debugger opens to that item in a new browser window.

After step 1, the Debug command on the Sitecore menu in the desktop activates the debugger with the context database set to the content database, rather than the default database associated with the context site determined for the requested URL. To debug a publishing target database without entering the sc_database query string parameter, select that database from the list that appears when you click the database icon in the lower-right corner of the desktop. Remember to select the Master database or close the desktop window after debugging. Alternatively, comment out the following line from the code shown in Listing 7-1:

```
url.Add("sc_database", Sitecore.Context.ContentDatabase.Name);
```

For more information about the Content Editor ribbon, see my blog post at http://bit.ly/nNPloM. Note that the Content Editor is not the only Sitecore application that exposes a ribbon, and

www.it-ebooks.info

the ribbon is not the only feature that exposes commands. For example, the User Manager application has its own ribbon, and features such as Content Editor warnings (described later in this chapter) can expose commands.

Adding Entries to Item Context Menus

You can add commands to the context menu that appears when you right-click an item in Sitecore. Using the context menu instead of the ribbon avoids the need to click a tab before clicking a command. More importantly, context menu entries make commands available when the ribbon does not appear, such as in dialogs outside of the Content Editor. One disadvantage of using the context menu is that usability decreases as you the number of commands shown increases.

As an example, consider the Path command in the Show group on the Developer tab, which copies the path to the item selected in the Content Editor to the operating system clipboard. To add this command to the context menu for all items, follow these steps:

1. Select the Core database using the selection icon in the lower-right corner of the Sitecore desktop, and then launch the Content Editor.
2. Navigate to the `/sitecore/content/Applications/Content Editor/Context Menues/Default` [sic] item.
3. Insert an item named **Path** using the `System/Menus/Menu` item data template. To control the order of entries in the context menu, sort this item and its siblings.
4. Set the Display name field in the Data section to **Copy Path**, the Icon field to **applications/16x16/window_dialog.png**, and the Message field to **clipboard:copypathtoclipboard(id=$Target)**.
5. Select the Master database using the selection icon in the lower-right corner of the Sitecore desktop. The Content Editor disappears and you return to the Sitecore desktop.
6. Open the Content Editor, right-click an item, and then click Copy Path.

For more information about replicating the Path command in the Show group on the Developer tab in the Content Editor to the item context menu, see my blog post at `http://bit.ly/qXKmvr`.

You can also create custom context menus for all items based on specific data templates and for individual items. Unfortunately, you cannot clone items in the Core database, so this technique results in some duplication:

1. Select the Core database using the selection icon in the lower-right corner of the Sitecore desktop, and then launch the Content Editor.
2. Duplicate the `/sitecore/content/Applications/Content Editor/Context Menues/Default` [sic] item with a name that identifies the context menu, such as the name of a relevant data template.
3. Add or remove commands from the new menu.
4. Select the Master database using the selection icon in the lower-right corner of the Sitecore desktop. You see the Sitecore desktop.

www.it-ebooks.info

5. Launch the Content Editor and navigate to the standard values definition item for the data template.
6. Select the Configure tab, and then click Context Menu in the Appearance group. The Context Menu dialog appears.
7. Select the context menu definition item that you created previously, and then click Open. The Context Menu dialog disappears and you return to the Content Editor.

Using Content Editor Warnings

When a user selects an item in the Content Editor, Sitecore invokes the `getContentEditorWarnings` pipeline defined in the Web.config file to generate the list of warnings to display.



Processors in the `getContentEditorWarnings` *pipeline often warn about conditions opposite to what their names imply. For example, the* `CanWrite` *processor generates a warning if the context user cannot write to the selected item. Others do exactly what their names imply. For example, the* `HasNoVersions` *processor generates a warning if the selected item has no versions in the current language.*

The default Content Editor warnings include the following:

- `ItemNotFound` — The specified item does not exist or the context user does not have read access.
- `CanReadLanguage` — The context user cannot read the specified item.
- `HasNoVersions` — The selected item has no versions in the current language.
- `CanWrite` — The context user does not have write access to the selected item.
- `CanWriteWorkflow` — The context user does not have write access to the selected version's current workflow state, and therefore cannot write to that version.
- `CanWriteLanguage` — The context user does not have write access to the current language, and therefore cannot write to the selected version.
- `IsReadOnly` — The selected item is protected (read-only).
- `IsLocked` — The selected item is locked to users other than the context user.
- `HasNoFields` — The data template for the selected item defines no fields other than those defined by the standard template.
- `NeverPublish` — Publishing restrictions for the selected item prevent its publication.
- `ItemPublishingRestricted` — Publishing restrictions for the selected item prevent its publication at this time.
- `VersionPublishingRestricted` — Publishing restrictions for the selected version prevent its publication at this time.

www.it-ebooks.info

- `ShowingInputBoxes` — Raw field values are visible in the Content Editor.
- `FeedIsEmpty` — Configuration error in RSS feed definition item.
- `RunRules` — Invokes the rules engine to invoke rules defined under the `/sitecore/system/Settings/Rules/Content Editor Warnings/Rules` item in the current database in a subsequent section of this chapter.
- `Notifications` — Renders information from the notification engine, such as information about a cloned item.

In cases of multiple warnings, the order of processors in the pipeline controls the order of warnings in the Content Editor. You can remove processors from, override processors in, and add processors to the `getContentEditorWarnings` pipeline to add any number of warnings to alert the user to various conditions. Each warning can include actions the user can click to resolve those issues.

You can add processors to the `getContentEditorWarnings` pipeline to report your own warning conditions when a user selects an item in the Content Editor. You can use warnings to display important information and convenient tasks. To implement a `getContentEditorWarnings` pipeline processor:

1. Create a class that implements the following method in your Visual Studio project:

```
public void Process(
  Sitecore.Pipelines.GetContentEditorWarnings.GetContentEditorWarningsArgs
  args)
```

2. Invoke the `Add()` method of the `Sitecore.Pipelines.GetContentEditorWarnings.GetContentEditorWarningsArgs` (`GetContentEditorWarningsArgs` for the remainder of these instructions) argument passed to the `Process()` method. The `Add()` method returns a `Sitecore.Pipelines.GetContentEditorWarnings.GetContentEditorWarningsArgs.ContentEditorWarning` (`ContentEditorWarning` in the remainder of these instructions) object that contains information to appear in the warning displayed to the user.
3. Set the `Title` and `Text` properties on the `ContentEditorWarning` object.
4. (Optional) Set the `Icon` property of the `ContentEditorWarning` object.
5. (Optional) Set the `HideFields` property of the `ContentEditorWarning` object to prevent the user from updating the item.
6. (Optional) Set the `IsExclusive` property of the `ContentEditorWarning` object to prevent other warnings from appearing (in which case, your code should call the `AbortPipeline()` method of the `GetContentEditorWarningsArgs` argument passed to the `Process()` method to prevent Sitecore from invoking additional processors in the pipeline).
7. (Optional) Set the `IsFullScreen` property of the `ContentEditorWarning` object to cause the warning to consume the entire editing pane in the Content Editor.
8. (Optional) Invoke the `AddOption()` method of the `ContentEditorWarning` object to add commands for the user to invoke in order to resolve the condition that resulted in the warning.
9. Add a `/configuration/sitecore/pipelines/getContentEditorWarnings/processor` element in the Web.config file with a `type` attribute containing the signature of your class.

www.it-ebooks.info

For more information about Content Editor warnings, see my blog post at `http://bit.ly/mYOQXT`. That blog post provides an example processor for the `getContentEditorWarnings` pipeline that warns the user if the selected item, the latest version of that item in the current language, or the current revision of that item in that language does not exist in all publishing target databases, and includes a command to refresh the user interface. That blog post also links to another example processor for the `getContentEditorWarnings` pipeline that notifies CMS users in the Content Editor before a code deployment.



As demonstrated in the section of this chapter about the rules engine, you can use the rules engine to configure Content Editor warnings. Content Editor warnings generated by the rules engine cannot expose commands to help the user address the issue.

Overriding Sitecore User Interface Applications

While Sitecore uses ASP.NET Web Forms (`.aspx` files) to implement some user interfaces, it uses a more elegant (and hence, sometimes complex) approach for developing certain user interface components. Instead of `.aspx` files for these components, Sitecore uses XML files under the `/sitecore/shell/applications` subdirectory within the document root. The Sitecore UI layer converts these XML files to ASP.NET controls hierarchies that render the user interface.



The names of the XML files do not always match the names of the applications they implement. Try to determine the XML file that implements a feature by locating the corresponding item(s) in the Core database, and validate your guess before you investigate that file too closely.

XML presentation components reference classes that effectively provide code-beside for the user interface represented by the XML. These classes tend to be in the `Sitecore.Client.dll` assembly. The `<CodeBeside>` element in XML presentation component files specifies the code-behind that contains logic for the control. Once you become familiar with the Sitecore UI architecture and the controls it uses, you can update existing XML controls, and override and even implement your own controls.

For example, use the following steps to investigate the Licenses command on the System submenu under All Applications on the Sitecore menu in the desktop:

1. Select the Core database using the icon in the lower-right corner of the Sitecore browser-based desktop.
2. Open the Content Editor and select the `/sitecore/content/Documents and settings/All users/Start menu/Programs/System/Licenses` item that defines the command that appears on the menu. Note the value in the Application field in the Data section that defines which application to invoke when a CMS user clicks this command.

www.it-ebooks.info

3. Select the `/sitecore/content/Applications/Licenses/LicenseOverview` item specified in the Application field that defines which application to invoke when a CMS user clicks this command. Note that you can update fields in the Appearance section to control the height, width, and other characteristics of the window that appears when the user opens this application.



The `/sitecore/content/Documents and settings/All users/Start menu/Programs/System/Licenses` *and* `/sitecore/content/Applications/Licenses/LicenseOverview` *items separate properties of the shortcut to the application from properties of the application itself.*

4. Select the Presentation tab, and then click Details in the Layout group. The Layout Details dialog appears. Note the layout specified for the Internet Explorer device, which in this case actually represents all browsers, and then click Cancel. The Layout Details dialog disappears and you see the Content Editor.
5. Select the `/sitecore/layout/Layouts/Control panel/Licenses/License overview` item, which is the layout specified for the Internet Explorer device. Note the value in the Control field in the Data section.
6. Select the Master database using the icon in the lower-right corner of the Sitecore desktop.

When a CMS user invokes this application, Sitecore uses the `.xml` file with the name specified in the Control field of the Data section of the layout definition item. Sitecore locates this file by checking for its existence in the subdirectories specified by the `folder` attributes of the `/configuration/sitecore/controlSources/source` elements in the Web.config file. Sitecore uses the file from the first subdirectory specified in which a file by that name exists. Under the default configuration, Sitecore uses the file from the `/sitecore/shell/override` subdirectory if it exists, or from another subdirectory of the `/sitecore/shell` subdirectory otherwise, making it easy to copy a file from another subdirectory of `/sitecore/shell` to `/sitecore/shell/override` to override a CMS UI component. In this case, the first matching file is `/sitecore/shell/Applications/Licenses/LicenseOverview/LicenseOverview.xml`.

To customize the licensing overview UI, follow these steps:

1. Copy `/sitecore/shell/Applications/Licenses/LicenseOverview/LicenseOverview.xml` in Visual Studio to the `/sitecore/shell/Override` subdirectory, and add the `/sitecore/shell/Override/LicenseOverview.xml` file to your Visual Studio project.



For UI components that use `.aspx` *files rather than XML interfaces, back up the original file and edit it in place, rather than copy the file to* `/sitecore/shell/Override`*. Whether you edit a file in place or copy it to the* `/sitecore/shell/Override` *subdirectory, add that file to your source code management system.*

www.it-ebooks.info

2. Add the following markup before the close of the `/control/LicenseOverview/FormPage/GridPanel/GridPanel` element in the `/sitecore/shell/override/LicenseOverview.xml` file:

```
<Border Background="white" Padding="8px">
  <Literal ID="ContributorPrefix" Text="Customized by " />
  <a ID="ContributorName" target="_blank"
    href="http://sitecorejohn.net">@sitecorejohn</a>
</Border>
```

3. Log into the Sitecore desktop, click the Sitecore button, and select All Applications ➪ System ➪ Licenses. The Licenses dialog shown in Figure 7-2 appears, which contains the value of the `Text` attribute of the `/control/LicenseOverview/FormPage/GridPanel/Border/Literal` element as well as the link that you added after that `<Literal>` element.

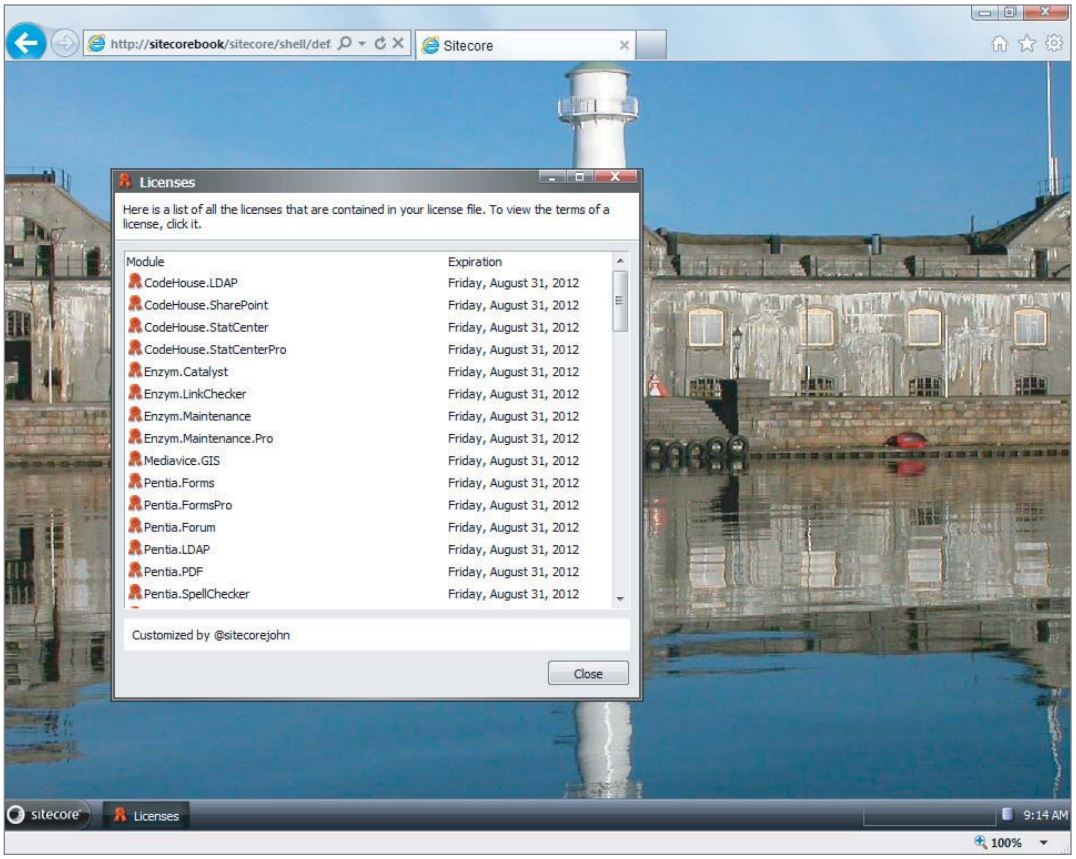


FIGURE 7-2

4. Click OK. You see the Sitecore desktop.

You can also override the code-beside for XML components. Your class can expose properties that correspond to elements in the XML file, where the type of the property is the same as the type of the

www.it-ebooks.info

element, and the name of the property is the value of the `ID` attribute of that element. For example, the class shown in Listing 7-2 inherits from the default implementation of the code-beside for the Licenses dialog. To use it, compile this class into your project and update the `Type` attribute of the `/control/LicenseOverview/FormPage/CodeBeside` element in the `/sitecore/shell/override/LicenseOverview.xml` file to the signature of that class.



LISTING 7-2: Overriding code-beside for the Licenses dialog, LicenseOverviewForm.cs

```
namespace SitecoreBook.Shell.Applications.Licenses.LicenseOverview
{
  using System;

  public class LicenseOverviewForm
    : Sitecore.Shell.Applications.Licenses.LicenseOverview.LicenseOverviewForm
  {
    public Sitecore.Web.UI.HtmlControls.Literal ContributorPrefix
    {
      get;
      set;
    }

    protected override void OnPreRender(EventArgs e)
    {
      this.ContributorPrefix.Text = "Recustomized by ";
      base.OnPreRender(e);
    }
  }
}
```

Updating the code-beside of the Licenses dialog to that shown in Listing 7-2 changes the text shown before the link based on the `Text` attribute of the `/control/LicenseOverview/FormPage/GridPanel/Border/Literal` dynamically.

Some Sitecore applications use standard ASP.NET Web Forms rather than XML controls. For example, the `/sitecore/content/Applications/Content Editor` item in the Core database defines the Content Editor application. Layout details for this item reference the `/sitecore/layout/Layouts/Applications/Content Manager` layout definition item.



For legacy reasons, some places in Sitecore may display the text Content Manager in place of the Content Editor.

The Path field in the Data section of that item references the `/sitecore/shell/applications/Content Manager/default.aspx` file, which implements the Content Editor. You can update the markup in this file or, just as you could in an XML control, you can specify an alternate code-behind. The default implementation of the Content Editor uses pipelines, including the `renderContentEditor`, `getContentEditorFields`, `getContentEditorSkin`, and `getContentEditorWarnings` pipelines. To minimize the chance of issues occurring during upgrades, use these pipelines whenever possible rather than customize the Content Editor.

www.it-ebooks.info

Whether you log in to the Content Editor or access the Content Editor within the Sitecore desktop, Sitecore uses the `/sitecore/content/Applications/Content Editor` item in the Core database. If you log in to the Content Editor, Sitecore directs the browser to the path `/sitecore/shell/Applications/Content%20editor.aspx` on the Sitecore server. This path triggers the `/configuration/sitecore/sites/site` element named `shell` in the Web.config file, which specifies the Core database as the default database, and Sitecore maps the URL to the item. In the definition for the Content Editor command on the Sitecore menu in the desktop, the Application field in the Data section references the `/sitecore/content/Applications/Content Editor` item. When you click that command, Sitecore loads the same URL used by the standalone Content Editor into an HTML `<iframe>` element within the desktop.

ENGAGING THE RULES ENGINE

The Sitecore rules engine provides a browser-based user interface that you can use to define logic and apply it under specific conditions in the CMS and the content delivery environment. The rules engine can associate rules with events and operations, where a rule consists of a number of conditions and a number of actions. When the event occurs, the rules engine evaluates the conditions and if the result is true, it invokes the actions. For example, you can define a rule that specifies that after an event, such as saving an item, under some condition, such as whether the item is associated with a specific data template, the rules engine invokes an action to update a field in that item.

Sitecore maintains information about rules in rule definition items using the Rule field type, which stores information about the conditions, actions, and parameters that you can pass to each. The location in the content tree determines the event that triggers some types of rule definition items, but you can add any number of fields of type Rule to your own data templates to define a rule in any type of item to use for your own purposes. Like any other field, the conditions and actions that appear in a field of type Rule depend on the `Source` property of that field, which specifies an item in the content tree. Use different data templates for different types of rules to apply a different `Source` property to the Rule field so that you can define conditions and actions for various kinds of rules in other locations.

Like so many things in Sitecore, conditions and actions involve definition items that contain fields specifying the .NET class that implements the component, and metadata about that component. Create condition and action definition items in folders under the `/sitecore/system/Settings/Rules` item using the `System/Rules/Condition` and `System/Rules/Action` data templates, respectively.

You can create rule definition items beneath the following items to invoke under each of the conditions described:

- `/sitecore/system/Settings/Rules/Content Editor Warnings/Rules` — Use the rules engine to create Content Editor warnings.
- `/sitecore/system/Settings/Rules/Insert Options` — Use the rules engine to define insert options.
- `/sitecore/system/Settings/Rules/Item Deleted/Rules` — Define rules to invoke after a user deletes an item.

www.it-ebooks.info

- `/sitecore/system/Settings/Rules/Item Saved` — Define rules to invoke after a user saves an item.
- `/sitecore/system/Settings/Rules/Version Removed/Rules` — Define rules to invoke after a user removes a version.

Certain tokens in the Text field in the Data section of condition definition items affect the user interface where CMS users select the condition and enter its parameters. If the word `if`, `when`, or `where` appears in the Text field, when presenting that value to the user, the word is a link that enables the user to reverse the condition to `except`, `except when`, or `except where`.

A sequence of tokens within square brace (`[]`) characters enables the CMS user selecting the condition to specify four parameters — separated by commas (`,`) — to pass when evaluating the condition. These four parameters within the braces specify the following, where the position of each parameter defines its purpose:

- The name of a property of the .NET class that implements the condition. Before evaluating the condition, the rules engine sets that property to the value specified by the user who defines the rule.
- An empty string, or the name of an item beneath the `/Sitecore/System/Settings/Rules/Common/Macros` item that controls the user interface the CMS user sees when defining a value for the parameter. For example, to activate a user interface allowing the user to select an item from the content tree, specify the value `tree`.
- An empty string, or parameters to pass to the user interface specified by the macro. For example, to specify the `/Sitecore/Content/Home` item as the root for a selection tree, enter `root=/sitecore/content/home`.
- The text to display until the CMS user specifies a value for the parameter.

Define global conditional rendering rules under the `/sitecore/system/Settings/Rules/Conditional Renderings/Global Rules` item. For more about conditional rendering, see Chapter 3.

Rather than write something here similar to existing resources, including those listed at the end of this section, I think a few visuals can truly assist you in understanding the rules engine. One simple example uses an action to generate a Content Editor warning message under the condition defined by the rule. To define a rule that can generate a Content Editor warning message in the Content Editor, follow these steps, but select conditions and actions and enter parameters appropriate for your requirements:

To use the rules engine to generate Content Editor warnings:

1. Select the `/sitecore/system/Settings/Rules/Content Editor Warnings/Rules` item in the Content Editor.
2. Insert a rule definition item using the `System/Rules/Content Editor Warning Rule` data template.
3. Click Edit rule above the Rule field in the Data section. The Rule Set Editor appears as shown in Figure 7-3.

www.it-ebooks.info

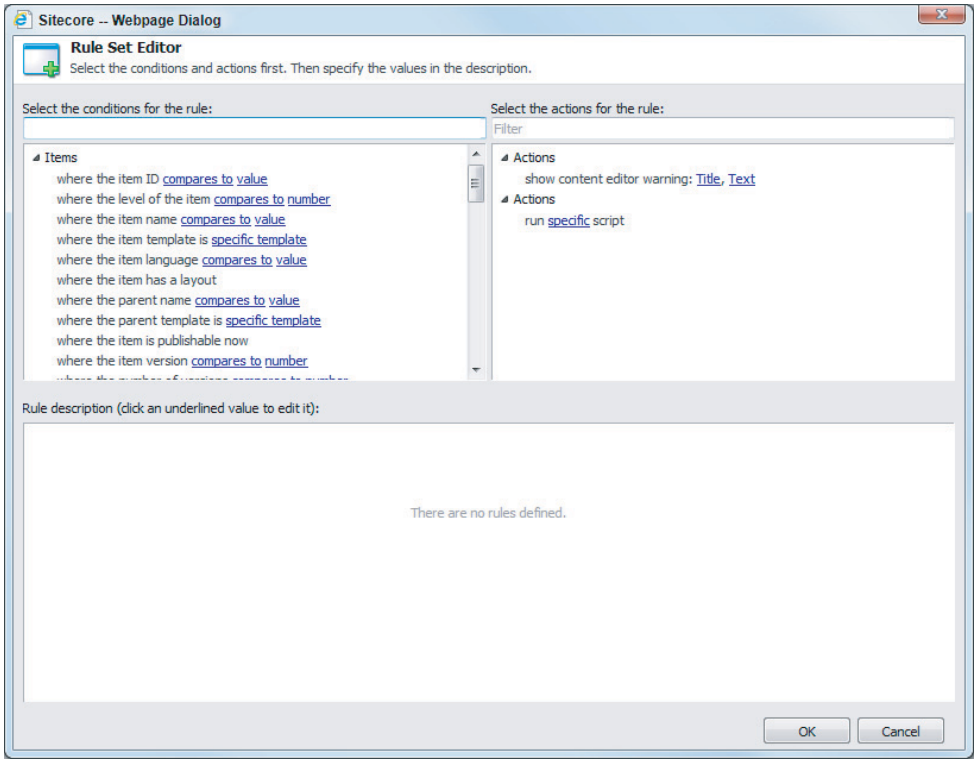


FIGURE 7-3



In the Rule Set Editor, you can enter text under "Select the conditions for the rule" to filter the list of condition definition items to only those with a specific value in the Text field.

4. Click the conditions that should trigger your warning from the list at the left.



To create combinatorial conditions, click additional conditions to add them to the rule. If you then click the word And that appears in the Rule Set Editor, Sitecore changes that group of conditions from an And condition to an Or condition. Clicking the word Or in the condition again changes it back to an And condition. When you hover the mouse over a condition, the Rule Set Editor exposes features that you can use to move that condition up or down or delete it from the list.

5. Click the Show Content Editor Warning: Title, Text action from the list at the right. Figure 7-4 shows the Rule Set Editor after clicking the Where True (Actions Always Execute) condition and the Show Content Editor Warning: Title, Text action.

www.it-ebooks.info

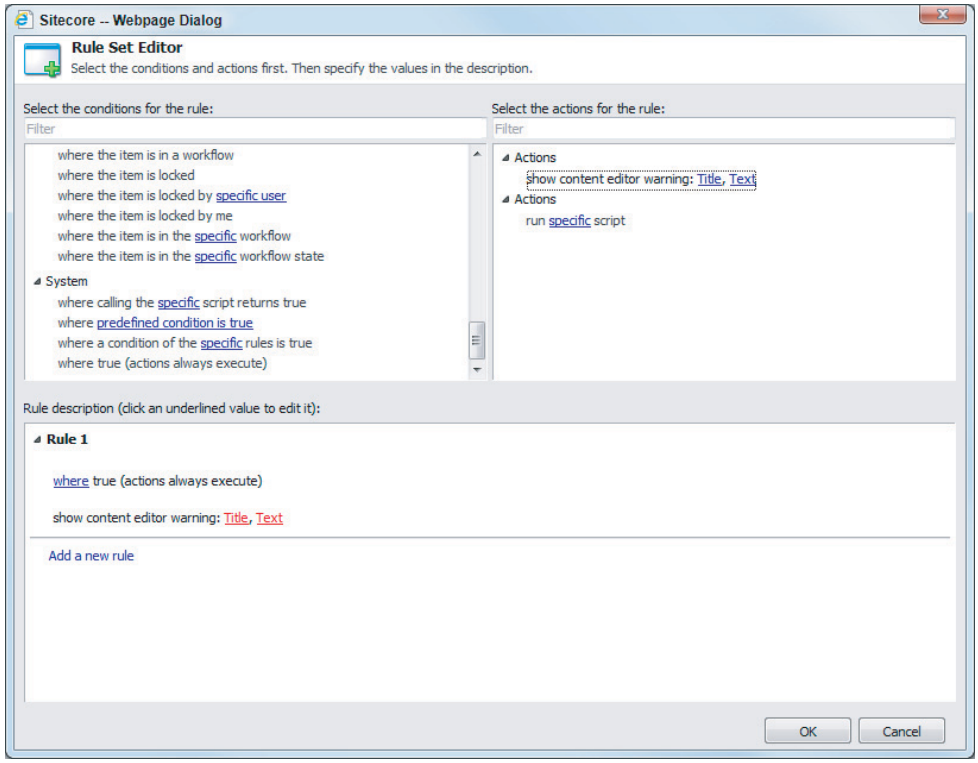


FIGURE 7-4



You can click links in the Rule description (such as the word Where in Figure 7-4) to reverse conditions (such as to Except Where).

6. Click Title and then Text to enter values for those properties of the warning in the rule description.
7. Click OK to close the Rule Set Editor and return to the Content Editor. When the condition is true, the warning appears at the top of the editing pane as shown in Figure 7-5 (at step 6 in these instructions, I entered **This is the title** for the Title parameter and **This is the text** for the Text parameter).

For more information about the rules engine and conditional rendering, see The Sitecore Rules Engine Cookbook (`http://bit.ly/skDqss`) and my blog post at `http://bit.ly/vb5rOA`. That blog post links to additional resources, such as my blog post about using the rules engine in a custom context to set the context device (`http://bit.ly/n2X3Pz`) and my blog post about using the rules engine to control item names (`http://bit.ly/qn0w1l`).

www.it-ebooks.info

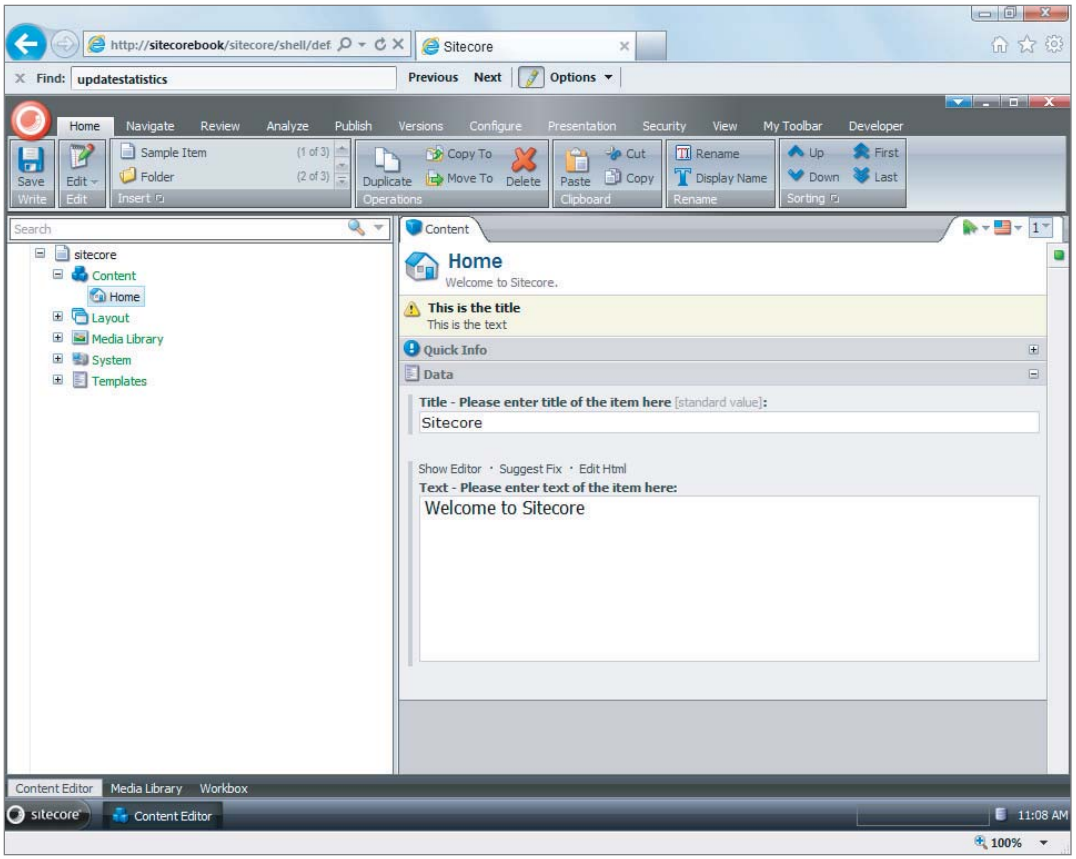


FIGURE 7-5

VALIDATING DATA

As mentioned in Chapter 2, you can validate individual fields and entire items. You can use validation for a surprising number of purposes. To implement validation, create a validator for field values or entire items, and then apply that validator to the fields or items.

Whether you validate a field or an item, the process is the same:

1. Create a class with the `[Serializable]` attribute that inherits from the `Sitecore.Data.Validators.StandardValidator` abstract class.
2. Implement the default constructor and a constructor with two arguments to support serialization.
3. Implement the `Evaluate()` method to validate the field or item. If the data is valid, then return `Sitecore.Data.Validators.ValidatorResult.Valid`. If the data is invalid, set the `Text` property defined by the `Sitecore.Data.Validators.BaseValidator` abstract class

www.it-ebooks.info

from which the `Sitecore.Data.Validators.StandardValidator` class inherits to a helpful message and return the value returned by calling the `GetMaxValidatorResult()` method.

4. Implement the `GetMaxValidatorResult()` method to return a value of the `Sitecore.Data.Validators.ValidatorResult` enumeration that defines the highest validation error level applied by the validator.
5. Implement the `Name` property to define a friendly name for the validator.
6. Register a validation definition item in Sitecore to define characteristics of your validator.
7. Select your validation definition item in fields of the Validation section of the items to which the validator should apply.

The main differences between field validators and item validators are as follows:

- Field validators validate the values of fields, which can contain changes that the CMS user has not yet saved. Item validators always validate saved items.
- Field validators validate the `ControlValidationValue` property defined in the `Sitecore.Data.Validators.BaseValidator` abstract class. Item validators validate the value in the item returned by the `GetItem()` method.
- You create field and item validator definition items in different locations.
- You select field and item validator definition items in the Validation section of different types of items.

In addition to the examples in the following sections, you can find numerous item field validators in the Sitecore Stuff (`http://bit.ly/uyoiCw`) Shared Source project.

Validating Fields

Field validators apply to values in individual fields. When a field contains invalid content, a bar appears next to the field in the Content Editor. The color of the bar indicates the severity of the error.

You can use the field validator shown in Listing 7-3 to ensure that all anchor (`<a>`) elements in all Rich Text Editor field values contain `title` attributes with values.



LISTING 7-3: Creating a field validator, LinkTitleValidator.cs

```
namespace SitecoreBook.Data.Validators.FieldValidators
{
  using System;
  using System.Runtime.Serialization;

  // field validator to ensure title attributes for <a> elements in RTE fields
  [Serializable]
  public class LinkTitleValidator : Sitecore.Data.Validators.StandardValidator
  {
    // default constructor
    public LinkTitleValidator()
```

continues

www.it-ebooks.info

LISTING 7-3 *(continued)*

```
{
}

// constructor supporting serialization
public LinkTitleValidator(SerializationInfo info, StreamingContext context)
{
}

// validate the field
protected override Sitecore.Data.Validators.ValidatorResult Evaluate()
{
  // optimization in case the field does not contain any links
  if (String.Compare(base.ControlValidationValue, "<a") < 0)
  {
    return Sitecore.Data.Validators.ValidatorResult.Valid;
  }

  // object representation of HTML value in field
  HtmlAgilityPack.HtmlDocument doc = new HtmlAgilityPack.HtmlDocument();

  // parse field contents into object
  doc.LoadHtml(base.ControlValidationValue);

  // use XPath-based syntax to retrieve <a> elements without title attributes
  HtmlAgilityPack.HtmlNodeCollection nodes = doc.DocumentNode.SelectNodes(
    "//a[@title='' or not(@title)]");

  // if there are no <a> elements without title attributes, the data is valid
  if (nodes == null || nodes.Count < 1)
  {
    return Sitecore.Data.Validators.ValidatorResult.Valid;
  }

  // otherwise, the data is invalid. use the href of the first <a> element
  // without a title attribute in the validation error message
  this.Text = "Title (tooltip) missing for link to "
    + nodes[0].GetAttributeValue("href", String.Empty)
    + ".";
  return this.GetMaxValidatorResult();
}

// defines the highest validation error level returned by this validator
protected override Sitecore.Data.Validators.ValidatorResult
  GetMaxValidatorResult()
{
  return base.GetFailedResult(Sitecore.Data.Validators.ValidatorResult.Error);
}

// the friendly name of this validator
public override string Name
{
  get { return "LinkTitleValidator"; }
```

www.it-ebooks.info

```
        }
      }
    }
```

This code uses the open-source HTML Agility Pack (HAP) distributed with Sitecore and documented at `http://htmlagilitypack.codeplex.com`. To use this library in your Visual Studio project, add a reference to the `HtmlAgilityPack.dll` assembly in the `/bin` subdirectory to your project. Remember to set the Copy Local property of the reference to False; otherwise, Visual Studio may delete assemblies from the `/bin` subdirectory when you compile.

This example depends on the `/Sitecore/System/Settings/Validation Rules/Field Types/Rich Text` item in the Master database. As mentioned in Chapter 2, due to a defect in some versions of Sitecore, the `/Sitecore/System/Settings/Validation Rules/Field Types/Rich Text` item erroneously uses the display name Multilist. In this case, there are two child items apparently named Multilist under the `/Sitecore/System/Settings/Validation Rules/Field Types` item. You can correct this issue in the Content Editor:

1. Click each of the children named Multilist under the `/Sitecore/System/Settings/Validation Rules/Field Types` item until the header above the editing pane fields displays the text *Multilist* - [*Rich Text*].
2. Select the Home tab, and then click Display Name in the Rename field. A Sitecore dialog appears containing a single field.
3. Clear the value in the field and then click OK. When the Content Editor appears, the item title bar contains the text *Rich Text*.

To configure this field validator, follow these steps in the Content Editor:

1. Navigate to the `/sitecore/system/Settings/Validation Rules/Field Rules` item.
2. Create a folder for your project, and potentially additional folders for different categories of validators.
3. Within one of these folders, insert a validator definition item using the `System/Validation/Validation Rule` data template.
4. Enter a short title for the validator in the Title field in the Text section, such as **Links Require Titles**.
5. Describe what the validator requires in the Description field in the Text section, such as **Links must specify the title attribute (tooltip)**.
6. Enter the signature of the validator in the Type field of the Data section.
7. Navigate to the `/Sitecore/System/Settings/Validation Rules/Field Types/Rich Text` item in the Content Editor.
8. In the fields of the Validation Rules section (Quick Action bar, Validate command, Validator bar, and Workflow), select the validator(s) to apply under those conditions.

www.it-ebooks.info

In addition to validating all fields of a specific type, as demonstrated in this example, you can validate specific fields in individual data templates. Follow these steps to validate an individual field in a specific data template:

1. In the Content Editor or the Template manager, expand the data template definition item, expand the section containing the field, and select the field definition item.
2. Select the validator(s) in the fields of the Validation Rules section (Quick Action bar, Validate command, Validator bar, and Workflow) to apply under those conditions.

Validating Items

Item validators apply to entire items, including all fields of the item. You can validate conditions about all fields, all fields of a given type, a specific combination of fields, values in other items, or any other logic you require. For an example of an item validator that ensures that the value of one date field follows the value in another date field, see my blog post at `http://bit.ly/rzYwVl`.

For example, to help ensure that users do not create links to items that Sitecore cannot render to a browser, the item validator shown in Listing 7-4 ensures that all items (excluding media items) referenced in all RTE (Rich Text Editor) fields contain some form of layout details.



LISTING 7-4: Defining an item validator, RTELinkValidator.cs

```
namespace SitecoreBook.Data.Validators.ItemValidators
{
  using System;
  using System.Runtime.Serialization;


  // item validator to ensure target items in <a> elements reference items with layout
  // details
  [Serializable]
  public class RTELinkValidator : Sitecore.Data.Validators.StandardValidator
  {
    // default constructor
    public RTELinkValidator()
    {
    }

    // constructor supporting serialization
    public RTELinkValidator(SerializationInfo info, StreamingContext context)
    {
    }

    // the friendly name of this validator
    public override string Name
    {
      get { return "RTELinkValidator"; }
    }

    // validate the item
    protected override Sitecore.Data.Validators.ValidatorResult Evaluate()
```

www.it-ebooks.info

```
{
  // the item to validate
  Sitecore.Data.Items.Item item = this.GetItem();

  if (item == null)
  {
    this.Text = "Item does not exist";
    return Sitecore.Data.Validators.ValidatorResult.FatalError;
  }

  // good practice before operations that access all fields
  item.Fields.ReadAll();

  foreach (Sitecore.Data.Fields.Field field in item.Fields)
  {
    // if it's not a rich text field, or it doesn't contain any links, ignore it
    if (field.TypeKey != "rich text" || field.Value.IndexOf("<a") < 0)
    {
      continue;
    }

    // object representation of HTML value in field
    HtmlAgilityPack.HtmlDocument doc = new HtmlAgilityPack.HtmlDocument();

    // parse field contents into object
    doc.LoadHtml(field.Value);

    // use XPath-based syntax to retrieve <a> elements with href attributes
    HtmlAgilityPack.HtmlNodeCollection nodes =
      doc.DocumentNode.SelectNodes("//a[@href]");

    // if there are no <a> elements without title attributes, the data is valid
    if (nodes == null || nodes.Count < 1)
    {
      return Sitecore.Data.Validators.ValidatorResult.Valid;
    }

    // otherwise, iterate the <a> elements in the field value
    foreach (HtmlAgilityPack.HtmlNode node in nodes)
    {
      // the value of the href attribute
      string href = node.GetAttributeValue("href", String.Empty);

      // ignore links to external web sites
      if (Sitecore.Web.WebUtil.IsExternalUrl(href))
      {
        continue;
      }

      // retrieve the item referenced by the href attribute
      Sitecore.Data.Items.Item target = this.GetItemFromHref(href);

      // if that item does not exist or the context user does not have read access
```

continues

www.it-ebooks.info

LISTING 7-4 *(continued)*

```
        // set the error message and return a value to indicate the data is invalid
        if (target == null)
        {
          this.Text = "Target item for link " + href + " does not exist";
          return this.GetMaxValidatorResult();
        }

        // ignore links to media items
        if (item.Paths.IsMediaItem)
        {
          continue;
        }

        // otherwise, if that item does not contain layout details for any device
        // set the error message and return a value to indicate invalid data
        if (String.IsNullOrEmpty(target[Sitecore.FieldIDs.LayoutField]))
        {
          this.Text = "Target item for link "
            + target.Paths.FullPath
            + " does not contain layout details.";
          return this.GetMaxValidatorResult();
        }
      }
    }

    // if no links are invalid, then all links are valid
    return Sitecore.Data.Validators.ValidatorResult.Valid;
  }

  // retrieve item referenced by href attribute of an RTE field value
  protected Sitecore.Data.Items.Item GetItemFromHref(string href)
  {
    Sitecore.Diagnostics.Assert.ArgumentNotNull(href, "href");

    // parse the href into an object
    Sitecore.Links.DynamicLink link;

    try
    {
      link = Sitecore.Links.DynamicLink.Parse(href);
    }
    catch (Sitecore.Web.InvalidLinkFormatException)
    {
      // the href does not reference an item that exists
      return null;
    }

    // retrieve the database
    Sitecore.Data.Database db = Sitecore.Configuration.Factory.GetDatabase(
      this.ItemUri.DatabaseName);

    // retrieve the item associated with the ID specified in the href attribute
    return db.GetItem(link.ItemId);
```

www.it-ebooks.info

```
        }

        // defines the highest validation error level returned by this validator
        protected override Sitecore.Data.Validators.ValidatorResult GetMaxValidatorResult()
        {
          return this.GetFailedResult(Sitecore.Data.Validators.ValidatorResult.Error);
        }
      }
    }
```

To configure this validator:

1. Navigate to the `/sitecore/system/Settings/Validation Rules/Item Rules` item.
2. Create a folder for your project, and potentially additional folders for different categories of validators.
3. Insert a validator definition item within one of these folders using the `System/Validation/Validation Rule` data template.
4. Enter a short title for the validator in the Title field in the Text section, such as **Links Require Layout Details**.
5. Describe what the validator requires in the Description field in the Text section, such as **The target items of links contain layout details**.
6. Enter the signature of the validator in the Type field in the Data section.
7. Navigate to the `/sitecore/system/Settings/Validation Rules/Global Rules` item in the Content Editor.
8. Select the validator in the fields of the Validation Rules section (Quick Action bar, Validate command, Validator bar, and Workflow) to apply the validator under those conditions.

Follow these steps to apply item validators to individual items in the Content Editor:

1. Select the View tab, and then select Standard Fields in the View group. The Content Editor displays the fields that Sitecore uses for validation, which the standard template defines.
2. Select the validator in the fields of the Validation Rules section (Quick Action bar, Validate command, Validator bar, and Workflow) to apply the validator under those conditions.

To apply validation to all items based on a data template, select validators in the standard values of that data template. For more information about validation, see The Sitecore Data Definition Reference (`http://bit.ly/nmGuiB`), The Sitecore Client Configuration Cookbook (`http://bit.ly/qS8Dc1`), and my blog post at `http://bit.ly/pxabIE`, which links to a number of sample validators.

SCHEDULING SITECORE PROCESSES

You can use the following techniques to schedule processes with Sitecore:

- **Agent —** Create a `/configuration/sitecore/scheduling/agent` element in the Web.config file. Agents provide the simplest mechanism to schedule tasks that repeat indefinitely at some interval.

www.it-ebooks.info

- **Task** — Create a command definition item under the `/sitecore/system/Tasks/Commands` item and a schedule definition item under the `/sitecore/system/Tasks/Schedules` item. Tasks enable you to control scheduled processes dynamically through APIs without updating the Web.config file, which restarts the ASP.NET worker process.
- **External Scheduler** — Use an external scheduling engine such as that built into Windows to invoke a command-line tool that invokes web services against the Sitecore solution. External scheduling engines are the only practical solution when you need to run something at an exact time, or in scenarios where the ASP.NET worker process is not active.

As an ASP.NET application, it is impractical for Sitecore to schedule a process to begin at an exact point in time. If needed, you can use Windows Scheduler to invoke a command-line tool that invokes a web service in the production content management environment to initiate a publishing operation. You can use additional scheduling engines and other techniques to schedule processes.



For Sitecore to invoke scheduled agents and tasks, the ASP.NET worker process must be running. To ensure an active ASP.NET worker processes, request an ASP.NET resource, such as the home page of the solution.

For more information about scheduling processes with Sitecore, see my blog post at `http://bit.ly/ncXS3p`.

Defining Agents in the Web.config File

The `/configuration/sitecore/scheduling/agent` elements in the Web.config file define agents, which are processes that run at a given interval. You can add `<agent>` elements to the Web.config file to schedule processes that repeat at an interval you specify.

The `type` attribute of each `<agent>` element contains the signature of the .NET class to invoke. The `method` attribute of each `<agent>` element defines the method of that class to call. The `interval` attribute of the `<agent>` element defines the minimal interval between invocations of the agent (`d.HH:mm:ss` format). To disable an agent, set the value of the `interval` attribute of the `/configuration/sitecore/scheduling/agents` element in the Web.config file to `00:00:00`.



Multiples of 24 hours, such as `24:00:00` or `48:00:00`, evaluate to 0 hours. You can prefix the number of hours with a number of days, such as `1.00.00.00` for 24 hours.

Comments above the `/configuration/sitecore/scheduling/agent` elements in the Web.config file describe the functions of the following default agents:

- `DatabaseAgent` — Invokes tasks scheduled in a specific Sitecore database (configured for both the Master database and the Core database by default)

www.it-ebooks.info

- `UrlAgent` — Periodically requests a page in an attempt to prevent the application pool from expiring
- `TaskDatabaseAgent` — Invokes the tasks scheduled in the database specified by the `connectionString` attribute of the first `<param>` element within the `/configuration/sitecore/TaskDatabase` element in the Web.config file (used for reminder and archive features not described in this book)
- `CompactClientDataAgent` — Optimizes information stored for CMS users
- `CleanupHistory` — Deletes expired information from the database table supporting Sitecore's history engine
- `CleanupPublishQueue` — Deletes expired information from the database table supporting incremental publishing operations
- `CleanupEventQueue` — Deletes expired information from the database table supporting remote event queues
- `HtmlCacheClearAgent` — Periodically clears output caches for all managed websites (disabled by default)
- `PublishAgent` — Periodically publishes (configure multiple `<agents>` to implement multiple publishing configurations)
- `CleanupAgent` — Deletes expired files from the Sitecore data subdirectory
- `CounterDumpAgent` — Periodically writes diagnostics information to files in the `/diagnostics` subdirectory of the Sitecore data subdirectory



The default Sitecore configuration disables some of these agents.

Note especially `UrlAgent`. Under some configurations, periodically requesting a page will prevent IIS from terminating the application pool. Restarting the application pool leads to application re-initialization and hence reduces performance for some HTTP requests immediately after startup. Ironically, keeping ASP.NET running also keeps the scheduling engine running.

If the URL specified by the first `<param>` element within the `/configuration/sitecore/scheduling/agent` does not specify a protocol and host, Sitecore assumes `http://127.0.0.1`, which is not necessarily a URL that maps to the Sitecore solution. This can lead to Sitecore log entries such as the following:

```
ManagedPoolThread #3 10:26:51 ERROR Exception in UrlAgent
  (url: /sitecore/service/keepalive.aspx)
Exception: System.Net.WebException
Message: The remote server returned an error: (404) Not Found.
```

To address this issue, add the appropriate protocol and domain to that URL.

To control the polling frequency, or how often Sitecore checks for agents to invoke, set the value of the `/configuration/sitecore/scheduling/frequency` element in the Web.config file (`d.HH:mm:ss` format).

www.it-ebooks.info



Set the polling frequency to half or less of the smallest value for the interval attribute in all `/configuration/sitecore/scheduling/agent` *elements defined in the Web.config file.*

To create a custom agent, follow these steps:

1. Create a class containing the method to invoke (typically named `Run()` with no parameters).
2. Add a `/configuration/sitecore/scheduling/agent` element to the Web.config file.
3. Set the `type` attribute to the signature of your class in the new `<agent>` element.
4. Set the `method` attribute to the name of your method in the new `<agent>` element.
5. Set the `interval` attribute to the frequency at which Sitecore should invoke the agent (`d.HH:mm:ss` format) in the new `<agent>` element.
6. If necessary, update the value of the `/configuration/sitecore/scheduling/frequency` element to ensure it is less than the `interval` attribute of the new `<agent>` element.



By default, Sitecore logs the invocation and completion of each agent. For more information about Sitecore logging, see Chapter 6.

For an example of a scheduled agent that removes old versions of items, see my blog post at `http://bit.ly/rfxtL7`.

Registering Tasks in a Sitecore Database

In addition to agents defined in the Web.config file, you can create task definition items in a Sitecore database. Ironically, Sitecore uses the `DatabaseAgent` and `TaskDatabaseAgent` agents defined in the Web.config file to invoke tasks defined in a Sitecore database. The difference between these two agents is that you must specify the database using the first `<param>` element of the `DatabaseAgent` `/configuration/sitecore/scheduling/agent` element in the Web.config file, whereas the `TaskDatabaseAgent` determines the database from the first `<param>` element within the `/configuration/sitecore/TaskDatabase` element in the Web.config file. The default Sitecore configuration invokes tasks defined in the Master and Core databases.

To define a task to invoke in a Sitecore database:

1. Create a .NET class that contains the logic in your Visual Studio project.
2. Implement a method in the new class with a signature based on the following prototype:

```
public void MethodName(
  Sitecore.Data.Items.Item[] items,
  Sitecore.Tasks.CommandItem command,
  Sitecore.Tasks.ScheduleItem schedule)
```

www.it-ebooks.info

You can pass items in the first parameter to your method. Sitecore passes the command and schedule definition items as the second and third parameters. Alternatively, you can add base templates to the data templates for commands and schedules, and retrieve values from those fields using the arguments passed to your task.

To define the schedule to invoke a task:

1. Create a command definition item using the `System/Tasks/Command` data template under the `/sitecore/system/Tasks/Commands` item in the Content Editor.
2. Set the value of the Method field in the Data section to the name of the method to invoke in the new class.
3. Insert one or more items using the `System/Tasks/Schedule` data template under `/sitecore/system/Tasks/Schedules`.
4. Select the command definition item that you created previously for the Command field in the Data section.
5. Enter values to control the schedule in the Schedule field (found in the Data section) as described in the following list, separated by pipe (`|`) characters:
   - The first value contains the start date for the schedule in `yyyyMMdd` format. For an explanation of the characters in date format strings, see `http://bit.ly/Ajmy6f`.
   - The second value contains the end date for the schedule in the same format.
   - The third value contains the days of the week on which to run the task, where 1=Sunday, 2=Monday, 4=Tuesday, 8=Wednesday, 16=Thursday, 32=Friday, and 64=Saturday, so every day is 1+2+4+8+16+32+64 = 127. The value is the minimum interval between invocations of the task in `d.HH:mm:ss` format.
6. (Optional) Indicate a list of items to pass to your command in the Items field, separated by pipe (`|`) characters. Alternatively, you can enter a Sitecore query without the `query:` prefix.
7. (Optional) Enter the date and time for Sitecore to determine when it last invoked the command due to the existence of this schedule definition item for `Last Run`. Sitecore automatically updates this field after invoking the command to control the processing schedule.

You can also use the `Sitecore.Globals.TaskDatabase` class to manipulate scheduled tasks in a Sitecore database.

To publish periodically, you can enable the `PublishAgent/configuration/sitecore/scheduling/agent` in the `Web.config` file. To publish in different configurations periodically, you can replicate this `<agent>` and change its parameters. For another approach to scheduled publishing, see the AutomatedPublisher (`http://bit.ly/u4TtVw`) Sitecore Shared Source project.



Publishing clears caches, which can affect performance. For optimal performance, do not schedule frequent publishing unnecessarily.

www.it-ebooks.info

INTEGRATING FROM THE BACK END

This section describes hooks, events, pipelines, pipeline processors, and data providers. You can use hooks to implement logic when the Sitecore application initializes. You can implement event handlers to respond to events that occur on the system. You can use processors to intercept pipelines that implement operations, in which you can interact with the user.

Initializing with Hooks

Sitecore hooks are .NET classes that contain logic to invoke on system initialization, before servicing any HTTP requests. Hooks have fewer capabilities than event handlers and especially pipeline processors.

The default hooks subscribe to events dynamically and configure memory and health monitoring. You can implement hooks similar to the default hooks that configure memory and system health monitoring, to register event handlers programmatically, to raise custom events for operations that do not raise events by default, to apply logic specific to development, test, production, and other environments, or for any other purposes you see fit. Hooks often leave behind a timer or other persistent artifact.

The `Sitecore.Pipelines.Loader.LoadHooks` processor in the `initialize` pipeline invokes the `Initialize()` method of each of the types defined by a `/configuration/sitecore/hooks/hook` element in the Web.config file.

To implement a hook:

1. Create a class that implements the `Sitecore.Events.Hooks.IHook` interface in your Visual Studio project.
2. Implement the `Initialize()` method in the class.
3. Add a `/configuration/sitecore/hooks/hook` element in the Web.config file based on the following example, changing the signature to that of your class:

```
<hook type="Namespace.ClassName, assembly" />
```

For more information about hooks, see my blog post at `http://bit.ly/o52FH9`.

Handling Events

Numerous operations in Sitecore raise events. Sitecore invokes any number of handlers for each event. The `name` attribute of each `/configuration/sitecore/events/event` element in the Web.config file defines an event. The `type` attribute of each `<handler>` child of the `/configuration/sitecore/events/event` element specifies the type that implements a handler for that event; the `method` attribute indicates the event handling method in that class. You can remove, override, and add event handlers, but be careful not to interfere with existing Sitecore functionality.

Event handlers accept two arguments: an `object` representing the entity that raised the event and an instance of the `Sitecore.Events.SitecoreEventArgs` class containing event arguments. Sitecore can pass zero or more parameters to each event. For example, Sitecore passes the item to save to handlers for the `item:saving` and `item:saved` events. Event handler parameters are positional: You access these parameters by numerical index, rather than by name. You can use the static

www.it-ebooks.info

`Sitecore.Events.Event.ExtractParameters()` argument method to retrieve a parameter from the `Sitecore.Events.SitecoreEventArgs` object passed to the method.

In load-balanced environments, you may need to handle events raised by remote instances as well. For each event defined in the Web.config file, such as `item:saved`, there is also a remote event, such as `item:saved:remote`, raised when the event occurs on a remote instance.

Many operations actually raise two separate events: one before or during the operation (the "ing" event, such as `item:saving`), and another afterward (the "ed" event, such as `item:saved`). If you need to access the values of fields as they were before an event, or to prevent Sitecore from triggering other event handlers for the event or from completing the operation that raised the event (such as update or deletion), use the "ing" event, such as `item:saving`. To cancel an event, set the `Result` property of the `Cancel` property of the `Sitecore.Events.SitecoreEventArgs` argument to `false` as demonstrated in the following example. There are no "ing" remote events, only "ed" remote events.

One common requirement for event handlers and other customizations is that they should run only for items in specific databases. For example, you might want an event handler to process only items in the Master database or the Core database, but not publishing target databases. You can use the configuration factory to specify the relevant database(s) for event handlers. You can also use the configuration factory to specify data templates, item paths, or other relevant processing criteria.

Listing 7-5 provides an example of an `item:saving` event handler that prevents users other than one specific user from saving changes to items previously updated by that user. The `SitecoreBook.Tasks.PreventSaveEventHandler` class could contain event handlers (methods with the signature of an event handler) for multiple events. Because this event handler depends on parameters (the username and databases on which to operate) that are not relevant to handlers for other events, it makes sense to isolate those properties and this logic into its own class. Because each class can have only a single event handler of a given name, you might also implement multiple classes if you need multiple handlers for an event and want to use a method naming convention such as `OnItemSaved()` for the `item:saved` event. The `Username` property enables you to specify the security domain and username. The `AddDatabase()` method enables you to specify the databases that this event handler should monitor. The `OnItemSaving()` method contains the event handler implementation.



LISTING 7-5: Defining an event handler, ItemEventHandler.cs

```
namespace SitecoreBook.Tasks
{
  using System;
  using System.Collections.Generic;

  // example class to contain event handler methods
  public class PreventSaveEventHandler
  {
    // ignore events for items not in these databases
    private List<string> _databases = new List<string>();

    // prevent other users from overwriting changes by this user
    public string Username
    {
      get;
```

continues

www.it-ebooks.info

LISTING 7-5 *(continued)*

```
    set;
  }

  // add a database to the list of databases this event handler monitors
  public void AddDatabase(string database)
  {
    // validate and convert arguments
    Sitecore.Diagnostics.Assert.ArgumentNotNullOrEmpty(database, "database");
    Sitecore.Diagnostics.Assert.IsNotNull(this._databases, "_databases");
    database = database.ToLower();

    // if the list already contains this database
    if (this._databases.Contains(database))
    {
      throw new ArgumentException(database + " specified twice.");
    }

    // if the specified database does not exist
    if (Sitecore.Configuration.Factory.GetDatabase(database) == null)
    {
      throw new ArgumentException(database + " does not exist.");
    }

    // add the name of the database to the list
    this._databases.Add(database);
  }

  // item:saving event handler to prevent users
  // other than that specified by the Username property
  // from saving changes to items last modified by that user
  public void OnItemSaving(object sender, EventArgs args)
  {
    // ensure the configuration specifies a username
    Sitecore.Diagnostics.Assert.ArgumentNotNullOrEmpty(
      this.Username,
      "specify username in event handler configuration");

    // if the context user is the specified user, they can always save
    if (String.Compare(Sitecore.Context.User.Name, this.Username, true) == 0)
    {
      return;
    }

    // validate and cast arguments
    Sitecore.Diagnostics.Assert.ArgumentNotNull(sender, "sender");
    Sitecore.Diagnostics.Assert.ArgumentNotNull(args, "args");
    Sitecore.Events.SitecoreEventArgs scArgs =
      args as Sitecore.Events.SitecoreEventArgs;
    Sitecore.Diagnostics.Assert.ArgumentNotNull(scArgs, "scArgs");
    Sitecore.Diagnostics.Assert.IsTrue(
      this._databases.Count > 0,
      "specify databases in event handler configuration");
```

www.it-ebooks.info

```
        // updated item (includes updates for the current save operation)
        Sitecore.Data.Items.Item updatedItem =
          Sitecore.Events.Event.ExtractParameter(args, 0) as Sitecore.Data.Items.Item;
        Sitecore.Diagnostics.Assert.ArgumentNotNull(updatedItem, "updatedItem");

        // if the operation is in a database other than those specified, ignore it
        if (!this._databases.Contains(updatedItem.Database.Name.ToLowerInvariant()))
        {
          return;
        }

        // the existing item (excludes updates from the current save operation)
        Sitecore.Data.Items.Item oldItem = updatedItem.Database.GetItem(
          updatedItem.ID);

        // if the item does not exist or the specified user was not last to update it,
        // the context user can save their changes
        if (oldItem == null
          || String.Compare(oldItem.Statistics.UpdatedBy, this.Username, true) != 0)
        {
          return;
        }

        // attempt to inform user their change won't persist
        if ((!Sitecore.Context.IsBackgroundThread)
          && Sitecore.Context.ClientPage.IsEvent)
        {
          Sitecore.Web.UI.Sheer.SheerResponse.Alert("You cannot save this item.");
        }

        // prevent the save operation and further item:saving event handlers, as well as
        // item:saved event handlers
        scArgs.Result.Cancel = true;
      }
    }
  }
```



In `item:saving` *event handlers, the instance of the* `Sitecore.Data.Items.Item` *class available through the arguments passed to the event handler (*`updatedItem` *in Listing 7-5) contains unsaved data. Sitecore APIs such as the* `GetItem()` *method of the* `Sitecore.Data.Database` *class retrieve the previously saved version of the item (*`oldItem` *in Listing 7-5).*

You can use a Web.config include file such as the following to insert this handler before the first existing handler for the `item:saving` event:




```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <events>
      <event name="item:saving">
        <handler type="SitecoreBook.Tasks.ItemEventHandler, assembly"
```

www.it-ebooks.info

```
            method="OnItemSaving" patch:before="handler[1]">
            <username>sitecore\jw</username>
            <databases hint="list:AddDatabase">
              <database hint="core">core</database>
              <database hint="master">master</database>
            </databases>
          </handler>
        </event>
      </events>
    </sitecore>
</configuration>
```

PreventSave.config



While this `item:saving` *event handler effectively blocks save operations, it does not prevent Sitecore from writing a log entry to audit the attempted save operation. Nor does it prevent the "shrinking" effect in the Content Editor that indicates that save completed, and it may not function correctly in the Content Editor. If this type of functionality is important to you, consider overriding commands or implementing a processor for the* `saveUI` *pipeline.*

Typically, Sitecore developers are most likely to implement handlers for the following events:

- `item:added` — Raised when the user adds an item using insert options (including the Insert from Template option)
- `item:creating/item:created` — Raised when creating an item, whether using insert options or not
- `item:copying/item:copied` — Raised when copying an item
- `item:deleting/item:deleted` — Raised when deleting an item
- `item:moving/item:moved` — Raised when moving an item
- `item:renaming/item:renamed` — Raised when renaming an item
- `item:saving/item:saved` — Raised when saving an item
- `item:sortorderchanged` — Raised after changing the sort order of an item
- `item:templateUpdated` — Raised after changing an item to use a different data template
- `item:versionAdding/item:versionAdded` — Raised when adding a version of an item
- `item:versionRemoving/item:versionRemoved` — Raised when removing a version of an item
- `publish:begin/publish:end` — Raised before and after publishing operations
- `publish:fail` — Raised if publishing fails

Of these, Sitecore developers are likely to implement handlers for the `item:saving` and `item:saved` events most often. Sitecore raises the `item:saving` and `item:saved` events immediately after some

www.it-ebooks.info

operations that update item data, such as renaming an item. In any case, users or API calls will eventually, inevitably save each item. Therefore, in some cases you can implement your item change handling logic using only `item:saving` and `item:saved` events instead of configuring handlers for multiple events that can involve item data, such as both `item:saved` and `item:renamed`. The NewsMover (`http://bit.ly/pljJVC`) Sitecore Shared Source project uses this approach to organize items into a hierarchy based on the value of a Date field. One potential drawback of this technique is that Sitecore raises your handler more frequently than needed.

To minimize the performance impact of event handlers, exit immediately when their logic is not relevant to the processing context. Gracefully handle multiple occurrences of an event when you might expect a single occurrence. For more information about events, including the parameters passed to handlers for each type of event, see `http://bit.ly/mWBrvL` on the Sitecore Developer Network, as well as my blog post at `http://bit.ly/nPpWD2`.

Processing Pipelines

Sitecore uses pipelines to implement a number of features. Each pipeline implements a single logical procedure, such as defining the Sitecore context for each HTTP request or generating a list of warning messages for an item in the Content Editor. Each pipeline consists of a series of processor classes, where each processor implements one aspect of the common function defined by the pipeline.

Pipelines and processors support separation of concerns between aspects of the system, providing for testability, configurability, and extensibility. You can remove existing processors, override default processors, and add your own processors, but be sure not to affect default Sitecore functionality.

Processor classes implement a method named `Process()` that accepts a single parameter and returns void. Some pipelines use a class specific to the pipeline to pass arguments, where that class inherits from the default `Sitecore.Pipelines.PipelineArgs`.

To implement a pipeline processor:

1. Create a class that implements a method named `Process()` in your Visual Studio project that has the same signature as the other processors in the pipeline. Your processor can inherit from an existing processor, and you can add, remove, replace, and rearrange processors in the pipelines to suit your requirements.
2. Add a `<processor>` element with the `type` attribute set to the signature of your processor to the pipeline definition in the Web.config file. Use the configuration factory to pass parameters by setting properties of the class.



Exit the `Process()` method immediately if the logic contained in the pipeline processor is not relevant to the processing context.

To cancel a pipeline from a pipeline processor, you can call the `AbortPipeline()` method of the argument passed to the `Process()` method of your processor.

Sitecore provides more pipelines than I can describe. Some of the most important pipelines under the `/configuration/sitecore/pipelines` element in the Web.config file, which typically implement

www.it-ebooks.info

system operations such as initialization or extending the ASP.NET page life cycle, and which you are most likely to extend, include the following:

- `initialize` — Runs when ASP.NET initializes Sitecore. In some cases, you can use a hook rather than add a processor to the initialize pipeline.
- `preprocessRequest` — Runs for each HTTP request for which IIS invokes ASP.NET to determine whether Sitecore should handle the request. The `httpRequestBegin` pipeline also includes processors that apply logic to each HTTP request.
- `httpRequestBegin` — Runs for each HTTP request for which IIS invokes ASP.NET
- `getContentEditorWarnings` — Runs each time you select an item in the Content Editor to generate a list of notifications (warnings) to display at the top of the editing pane.
- `httpRequestEnd` — Performs diagnostics after each HTTP request
- `insertRenderings` — Determines presentation components to render an item
- `job` — Invokes jobs (background processes that can run for long periods) in dedicated threads
- `publish` — Manages publishing
- `publishItem` — Runs to publish each item. For an example `publishItem` pipeline processor, see my blog post at `http://bit.ly/n0Wsn6`.
- `renderLayout` — Populates placeholders to assemble the control hierarchy for rendering a page
- `healthMonitor` — Periodically invokes diagnostic operations
- `sessionEnd` — Runs when an ASP.NET session ends
- `getMediaStream` — Retrieves the binary stream for a media request
- `expandInitialFieldValue` — Expands tokens in standard values such as `$name` when you create items
- `getLookupSourceItems` — Determines the items to include in selection data template fields. For an example `getLookupSourceItems` pipeline processor, see my blog post at `http://bit.ly/q7oGx8`.
- `getContentEditorFields` — Determines the fields to display in the Content Editor
- `getPlaceholderRenderings` — Determines the renderings a user can add to a placeholder
- `getChromeData` — Determines features available for placeholders, fields, markup elements, and renderings in the Page Editor



In the `getChromeData` *pipeline, chrome refers to HTML markup elements around components in the Page Editor, not the Google Chrome browser.*

- `getRenderingDatasource` — Controls the CMS user interface that allows the user to create or select a data source for a presentation component
- `renderField` — Renders the value of a field

www.it-ebooks.info

- `search` — Implements search operations
- `filterItem` — Determines which version of items to display in interfaces such as Preview

Some of the most important pipelines defined under the `/configuration/sitecore/processors` element in the `Web.config` file, which typically define pipelines used by Sitecore user interfaces, include the following:

- `uiAddFromTemplate` — Invoked when the user adds an item based on a data template, branch template, or command template
- `uiCopyItems` — Copies an item and its descendants
- `uiCloneItems` — Clones an item and its descendants
- `uiDeleteItems` — Deletes an item and its descendants
- `uiDragItemTo` — Invoked when a user drags and drops an item
- `uiDuplicateItem` — Duplicates an item
- `uiMoveItems` — Moves an item and its descendants
- `uiRenameItem` — Renames an item
- `uiGetMasters` — Determines effective insert options for an item
- `loggingin` — Invoked when a user logs in
- `loggedin` — Invoked after a user logs in. For an example `loggedin` pipeline processor, see my blog at `http://bit.ly/pM4FdY`.
- `logout` — Invoked after a user logs out
- `saveUI` — Invoked when a CMS user saves an item
- `uiUpload` — Invoked when a user upload a media item

Sitecore modules and customizations, especially the Digital Marketing System (DMS), introduce additional pipelines, typically using Web.config include files.

For more information about pipelines, including a link to a blog post about creating and invoking your own pipelines, see my blog post at `http://bit.ly/oLbcKL`. For more information about important CMS pipelines, see my blog post at `http://bit.ly/q8Ppms`.

Handing the httpRequestBegin Pipeline

For each HTTP request processed by Sitecore, the layout engine invokes the `httpRequestBegin` pipeline, primarily to define the Sitecore context, but also for general HTTP request handling such as redirection. The `httpRequestBegin` pipeline consists of the following processors:

- `CheckIgnoreFlag` — Aborts the pipeline if the `preprocessRequest` pipeline determined that Sitecore should not process this request
- `StartMeasurements` — Sets up a timer to calculate the amount of time elapsed while processing the request and stores information used later to approximate the number of items

www.it-ebooks.info

accessed by the rendering and the amount of memory consumed to generate threshold warnings in the Sitecore log

- `IgnoreList` — Aborts the pipeline if the URL matches any of the prefixes in the `IgnoreUrlPrefixes` setting in the Web.config file
- `SiteResolver` — Determines the context site
- `UserResolver` — Determines the context user
- `DatabaseResolver` — Determines the context database
- `BeginDiagnostics` — Initiates diagnostics if enabled
- `DeviceResolver` — Determines the context device
- `LanguageResolver` — Determines the context language
- `CustomHandlers` — Activates the appropriate handler if the URL matches one of the trigger attributes of any of the `/configuration/sitecore/customHandlers/handler` elements in the Web.config file
- `FilterUrlExtensions` — Does nothing (the `preprocessRequest` pipeline and the `CheckIgnoreFlag` processor earlier in the `httpRequestPipieline` seem to have rendered this processor obsolete). You can remove this processor from the pipeline.
- `DefaultResolver` — Determines the context item if the request triggers the home page of a managed website
- `FileResolver` — Handles URLs that map to files under the document root, rather than items in a database
- `ItemResolver` — Determines the context item from the path in the URL
- `LayoutResolver` — Determines the layout or Sitecore user interface control to process the request
- `ExecuteRequest` — Redirects under various error conditions



The `httpRequestBegin` *pipeline does not actually assemble a page. Processors in the* `renderLayout` *pipeline manipulate the ASP.NET control tree, but Sitecore then returns control to ASP.NET, which assembles the page from the control tree.*

You can use the `httpRequestBegin` pipeline for a number of purposes:

- To determine whether Sitecore should process a request, use a specific handler to process a request, let ASP.NET and IIS process a request without Sitecore, redirect a request, or otherwise
- To rewrite URLs, including mapping legacy URLs to the URLs of corresponding Sitecore items

www.it-ebooks.info

- To resolve Sitecore context properties, such as the context item, using custom logic, allowing custom URLs
- To enforce canonical URLs
- To implement performance diagnostics
- To manipulate HTTP headers
- To authenticate users, such as to support SSO (single sign-on)
- For additional purposes, such as custom HTTP 404 (page not found) management

For example, you might want to include ETag (Entity Tag — see `http://bit.ly/8cQ8Z3`) headers in the HTTP responses for some of your managed websites. Conveniently, Sitecore updates a unique revision identifier each time an item changes; you can use the revision identifier as the ETag to uniquely identify a revision of a version within a language.

The `Sitecore.Context.Site` static property exposes the context site determined from the current HTTP request using the `Sitecore.Sites.SiteContext` class. The extension class in Listing 7-6 adds the `GetETag()` method to that class, which returns the value of the `etag` attribute that you can add to `/configuration/sitecore/sites/site` elements in the Web.config file that configures each managed site.


Available for download on Wrox.com

LISTING 7-6: Extending the context site, SiteContextETag.cs

```
namespace SitecoreBook.Sites
{
  // extend Sitecore.Sites.SiteContext (and hence Sitecore.Context.Site)
  // with the GetETag() method to evaluate the etag attribute of the
  // corresponding /configuration/sitecore/sites/site element in web.config
  public static class SiteContextETag
  {
    // evaluate the etag attribute of the site
    public static bool GetETag(
      this Sitecore.Sites.SiteContext me)
    {
      // validate arguments
      Sitecore.Diagnostics.Assert.IsNotNull(me, "me");

      // retrieve the attribute value
      string tag = me.Properties["etag"];

      // compare the attribute value and return the result
      return tag != null && (tag == "true" || tag == "yes" || tag == "1");
    }
  }
}
```

If the `etag` attribute of the `/configuration/sitecore/sites/site` element in the Web.config file associated with the context site is `true`, `yes`, or `1`, the `httpRequestBegin` pipeline processor in Listing 7-7 sets the ETag header in the HTTP response to the revision identifier in the context item.

www.it-ebooks.info



LISTING 7-7: Setting the ETag header, ETag.cs

```
namespace SitecoreBook.Pipelines.HttpRequest
{
  // in order to use the GetETag() extension method without full qualification
  using SitecoreBook.Sites;

  // httpRequestBegin pipeline processor to set the ETag HTTP header
  // to the revision identifier for the context item if the etag
  // attribute of the /configuration/sitecore/sites/site element
  // defining the context site is true
  public class ETag : Sitecore.Pipelines.HttpRequest.HttpRequestProcessor
  {
    // processor implementation
    public override void Process(
      Sitecore.Pipelines.HttpRequest.HttpRequestArgs args)
    {
      // if the context item or the context site is unknown,
      // or if the etag attribute of the context site is absent or false,
      // do nothing
      if (Sitecore.Context.Item == null
        || Sitecore.Context.Site == null
        || !Sitecore.Context.Site.GetETag())
      {
        return;
      }

      // set the ETag header in the HTTP response to the revision identifier
      // of the context item
      args.Context.Response.Headers["ETag"] =
        Sitecore.Context.Item.Statistics.Revision;
    }
  }
}
```

You can use the following Web.config include file to insert this processor before the last existing processor in the `httpRequestBegin` pipeline, and to add the `etag` attribute for the default `/configuration/sitecore/sites/site` element named `website` in the Web.config file. Most important, this processor must follow the default ItemResolver processor and any custom processors that set the context item (the `Sitecore.Context.Item` static property).




```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <sites>
      <site name="website">
        <patch:attribute name="etag">true</patch:attribute>
      </site>
    </sites>

    <pipelines>
      <httpRequestBegin>
        <processor type="SitecoreBook.Pipelines.HttpRequest.ETag,assembly"
          patch:before="processor[last()]" />
```

www.it-ebooks.info

```
      </httpRequestBegin>
    </pipelines>
  </sitecore>
</configuration>
```

AddETag.config

You might be amazed by how many challenges you can address with these techniques, often where you might expect to use an ASP.NET module or handler. For additional example `httpRequestBegin` pipeline processors, see the PageNotFound (`http://bit.ly/vIoNwP`) Sitecore Shared Source project.

Processing the renderField Pipeline

The `renderField` pipeline applies two fundamental transformations to field values:

- It expands dynamic links, including both content and media references.
- It adds inline editing features if the user is inline editing in the Page Editor.

Processors in the `renderField` pipeline accept a parameter of type `Sitecore.Pipelines.RenderField.RenderFieldArgs`, which exposes a `Sitecore.Xml.Xsl.RenderFieldResult` property named `Result`. The `Result.FirstPart` property of this object typically contains the field value of the field, and each processor in the pipeline can transform that value, but you should apply the same logic to the `Reply.LastPart` property in case it also contains a value. For example, some field types may store an opening element, such as `<a>`, in `Result.FirstPart`, and store the closing `</a>` in `Result.LastPart`.

The `renderFileld` pipeline consists of the following processors:

- `SetParameters` — Processes the `disable-web-editing`, `show-title-when-blank`, `linebreaks`, and `default-text` attributes if passed to the pipeline
- `GetFieldValue` — Retrieves the value of the field
- `ExpandLinks` — Expands dynamic links in the field value
- `GetImageFieldValue` — Manages inline editing features for Image fields in the Page Editor
- `GetLinkFieldValue` — Manages inline editing features for Link and General Link fields in the Page Editor
- `GetInternalLinkFieldValue` — Manages inline editing features for Internal Link fields in the Page Editor
- `GetMemoFieldValue` — Handles line breaks in Multi-Line Text fields
- `GetDateFieldValue` — Handles formatting and manages inline editing features for Date and Datetime fields
- `GetDocxFieldValue` — Manages inline editing features for the Word Document field types, and renders Word Document field values as HTML where appropriate
- `AddBeforeAndAfterValues` — Coalesces values retrieved thus far in the pipeline
- `RenderWebEditing` — Adds inline editing features around the field value if the user is inline editing in the Page Editor

www.it-ebooks.info

The `renderField` pipeline processor shown in Listing 7-8 sets the `target` attribute of any HTML anchor (`<a>`) elements in Rich Text fields for which the `target` attribute is missing or empty to `_blank`, causing the browser to open a new window if the user clicks such a link.



Available for download on Wrox.com

LISTING 7-8: Adding target attributes to external links, AddLinkTargets.cs

```
namespace SitecoreBook.Pipelines.RenderField
{
  using System;
  using System.IO;
  using System.Text;

  // renderField pipeline processor to add target attributes to external links
  public class AddLinkTargets
  {
    // renderField pipeline processor implementation
    public void Process(Sitecore.Pipelines.RenderField.RenderFieldArgs args)
    {
      // ignore field types other than rich text
      if (args.FieldTypeKey != "rich text")
      {
        return;
      }

      // transform both parts of the result
      args.Result.FirstPart = this.EnsureTargets(args.Result.FirstPart);
      args.Result.LastPart = this.EnsureTargets(args.Result.LastPart);
    }

    // returns the input string after adding target attributes to external links
    protected string EnsureTargets(string markup)
    {
      // optimization in case the markup does not contain any links
      if (!markup.Contains("<"))
      {
        return markup;
      }

      // parse the markup into an object
      HtmlAgilityPack.HtmlDocument doc = new HtmlAgilityPack.HtmlDocument();
      doc.LoadHtml(markup);

      // retrieve all <a> elements with an empty target attribute
      // or no target attribute
      HtmlAgilityPack.HtmlNodeCollection nodes = doc.DocumentNode.SelectNodes(
        "//a[@target = '' or not(@target)]");

      // if there are no such nodes, return the original markup
      if (nodes == null || nodes.Count < 1)
      {
        return markup;
      }
```

www.it-ebooks.info

```
        // avoid rewriting the markup if no external links found
        bool foundLink = false;

        // for each such anchor
        foreach(HtmlAgilityPack.HtmlNode node in nodes)
        {
          // value of href attribute of <a> element
          string href = node.GetAttributeValue("href", String.Empty);

          // if href is empty or does not start with http or https, ignore it
          if (String.IsNullOrEmpty(href)
            || !(href.StartsWith("http://") || href.StartsWith("https://")))
          {
            continue;
          }

          // found an external link that requires a target attribute
          foundLink = true;
          node.SetAttributeValue("target", "_blank");
        }

        // if the markup did not contain any external links, return the original
        if (!foundLink)
        {
          return markup;
        }

        // convert from object form back to markup, and return that markup.
        StringBuilder sb = new StringBuilder();
        StringWriter sw = new StringWriter(sb);
        doc.Save(sw);
        sw.Close();
        return sb.ToString();
      }
    }
  }
```

You can use the following Web.config include file to add this processor toward the end of the `renderField` pipeline, after retrieving and transforming field values, but before merging `FirstPart` and `LastPart`:




```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <pipelines>
      <renderField>
        <processor type="SitecoreBook.Pipelines.RenderField.AddLinkTargets,assembly"
          patch:before="processor[last() - 2]" />
      </renderField>
    </pipelines>
  </sitecore>
</configuration>
```

AddLinkTargets.config

www.it-ebooks.info

For additional examples of `renderField` pipeline processors, see my blog posts at `http://bit.ly/qlA19p` and `http://bit.ly/nhEsOm`, as well as the MediaUrlTransformer (`http://bit.ly/vf6Lie`) Sitecore Shared Source project.

Intercepting the renderLayout Pipeline

The `renderLayout` pipeline assembles the control tree that ASP.NET uses to render the page, and sets some client cache configuration options for the page. The `renderLayout` pipeline consists of the following processors:

- `CheckIgnoreFlag` — Aborts the pipeline if the `preprocessRequest` pipeline determined that Sitecore should not process this request
- `PageHandlers` — Handles specific HTTP requests by invoking a page from the server and returning its response or by invoking a method. Sitecore has deprecated this processor.
- `SecurityCheck` — Aborts the pipeline and redirects to the login page for the context site if the user does not have access to enter the context site or read the context item
- `InsertRenderings` — Binds renderings to placeholders
- `PageExtenders` — Adds the page-appropriate extenders defined by `/configuration/sitecore/pageextenders/pageextender` elements in the Web.config file, including support for the Page Editor, Preview, and the browser-based Sitecore debugger
- `BuildTree` — Expands controls and placeholders
- `InsertSystemControls` — Inserts system controls, such as those used by page extenders
- `InsertUnusedControls` — Attaches presentation components not bound to any placeholder to the `<form>` element in the page (for internal use by Sitecore)
- `BrowserCaching` — Sets the `HTTP Last-Modified` header based on the last update date for the version of the context item in the context language, and the `Cache-Control` and `Pragma` HTTP headers according to the `DisableBrowserCaching` setting in the Web.config file.

The `renderLayout` pipeline processor shown in Listing 7-9 logs the requested URL and the current structure of the control tree.



LISTING 7-9: Logging the control tree, LogControlTree.cs

```
namespace SitecoreBook.Pipelines.RenderLayout
{
  using System.Web.UI;

  // renderLayout pipeline processor to log the current ASP.NET control tree
  public class LogControlTree : Sitecore.Pipelines.RenderLayout.RenderLayoutProcessor
  {
    // message to include in header log messages
    public string Message
    {
      get;
      set;
```

www.it-ebooks.info

```
    }

    // renderLayout pipeline processor implementation
    public override void Process(
      Sitecore.Pipelines.RenderLayout.RenderLayoutArgs args)
    {
      // if the Sitecore page or the ASP.NET page is null, do nothing
      if (Sitecore.Context.Page == null || Sitecore.Context.Page.Page == null)
      {
        return;
      }

      // log a header message indicating the requested URL
      Sitecore.Diagnostics.Log.Info(
        this + " : " + this.Message + " : " + Sitecore.Context.RawUrl,
        this);

      // log the control tree
      this.LogControls(Sitecore.Context.Page.Page, 0);
    }

    // recursive method to log information about a control and its descendants
    protected void LogControls(Control control, int level)
    {
      // log entries includes simple spacing indent, control IDs, and control types
      string msg = this
        + " : "
        + Sitecore.StringUtil.Repeat(" ", level)
        + control.ID
        + "("
        + control.GetType()
        + ")";
      Sitecore.Diagnostics.Log.Info(msg, this);

      // process all controls in the tree
      foreach(Control child in control.Controls)
      {
        this.LogControls(child, level + 1);
      }
    }
  }
}
```

You can use the following Web.config include file to insert this processor before the first existing processor and after the last existing processor in the `renderLayout` pipeline:




```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <pipelines>
      <renderLayout>
        <processor patch:before="processor[1]" hint="before"
          type="SitecoreBook.Pipelines.RenderLayout.LogControlTree,assembly">
          <message>before expansion</message>
        </processor>
```

www.it-ebooks.info

```
        <processor patch:after="processor[last()]" hint="after"
          type="SitecoreBook.Pipelines.RenderLayout.LogControlTree,assembly">
          <message>after expansion</message>
        </processor>
      </renderLayout>
    </pipelines>
  </sitecore>
</configuration>
```

LogControlTree.config

DMS Pipelines

The Sitecore Digital Marketing Suite (DMS) uses the `/App_Config/Include/Sitecore.Analytics.config` `web.config` include file to define several pipelines and update pipelines in the `Web.config` file provided with the CMS.

DMS pipelines include the following:

- `automation` — Applies engagement automation
- `startTracking` — Invokes the `initializeTracker` pipeline, parses tracking as specified by query string parameters, processes tracking defined for the context item
- `initializeTracker` — Initializes the `Sitecore.Analytics.Tracker` class used to implement tracking
- `parseReferrer` — Identifies search engines and search terms provided by the browser using the referrer HTTP header
- `trafficTypes` — Identifies the traffic type associated with a visit (referral, organic search, branded search, or otherwise)
- `createVisit` — Records an analytics visit
- `registerPageEvent` — Associates page events, including goals and failures, with pages
- `triggerCampaign` — Triggers campaigns
- `processMessage` — Processes outbound e-mail messages

DMS updates the following CMS pipelines:

- `renderLayout` — Adds event handlers to support tracking
- `getContentEditorWarnings` — Warns about items missing marketing profiles
- `initialize` — Initializes tracking and engagement automation
- `httpRequestBegin` — Initiates analytics diagnostics and sets context items depending on page-level multivariate testing
- `httpRequestEnd` — Applies analytics tracking
- `sessionEnd` — Invokes rules and automations

For more information about important DMS pipelines, including links to example pipeline processors, see my blog post at `http://bit.ly/nPlMqj`.

www.it-ebooks.info

Hook, Pipeline Processor, Event Handler, or Rule?

In some cases, you can implement the same feature in Sitecore using an initialization hook, a pipeline processor, an event handler, or the rules engine. This is especially true for managing what happens when a user or the system updates data, which can involve the `item:saving` event, the `item:saved` event, the `saveUI` pipeline, item saved event rules, validation, and potentially other features. Here are some general guidelines:

- Use an initialization hook for system initialization logic that does not depend on any processing context.
- Use events to handle both interactive (UI) and automated (API) operations.
- If you use events, use "ed" events when you can; use "ing" events only when necessary, such as to cancel an operation.
- Use pipelines to interact with the user.
- Use the rules engine to provide a browser-based interface to configure logic, especially for nontechnical business users.
- Always remember to consider whether validation can meet your requirements, rather than more complex approaches that can adversely interact with Sitecore itself, especially for expensive operations.

Most operations that update data raise item saved events. Certain operations raise additional events, such as the creation of an item. In some cases, you can use either event, such as the creation event or the save event that follows it. One of the advantages of using save handlers and pipelines (as opposed to creation or other events or pipelines) is that Sitecore invokes the save event for each save operation. If you associate an event handler with item creation, it only runs when you create the item; if the logic fails or data changes on which the event handler depends, your logic will not run again. Additionally, you might have to intercept multiple events, such as creation and renaming. If you handle a save event instead, your logic runs when the user creates, renames, updates, or otherwise touches an item, but not when the user duplicates an existing item. Save handlers are more expensive in terms of compute resources, but more reliable for handling data changes over time, and easier to implement and configure if you need to handle multiple events.

For more ideas about choosing between events, pipelines, the rules engine, and validators, see my blog post at `http://bit.ly/nmA1Ov`.

EXTENDING THE SITECORE PAGE EDITOR

You can extend the Page Editor in a number of ways. As described in Chapter 3, your presentation controls can generate different output depending on the active Sitecore mode (`Sitecore.Context.PageMode`), such as whether the user is debugging or inline editing in the Page Editor. This book does not provide a specific example; but similar to the Content Editor, you can add commands to the Page Editor ribbon. As explained in the following instructions, you can extend the Page Editor with commands that appear in the chrome around the markup element, field, rendering, or placeholder selected by the Page Editor.

www.it-ebooks.info



For optimal usability, whenever possible, use field, rendering, and placeholder commands in favor of edit frames around an individual markup element. For more information about edit frames, see The Sitecore Client Configuration Cookbook (`http://bit.ly/qS8Dc1`*).*

The chrome that can appear around each HTML element, field, rendering, and placeholder in the Page Editor provide inline editing, access to layout details, personalization, and other features for these objects in the Page Editor. Sitecore uses the `getChromeData` pipeline to determine which commands to expose in the chrome for each component. The `RenderChromeData` processor in the `renderField` pipeline invokes the `getChromeData` pipeline around fields.

To create a Page Editor command for a field, rendering, or placeholder:

1. Create a class that inherits from the `Sitecore.Shell.Applications.WebEdit.Commands.WebEditCommand` class in your Visual Studio project.
2. Create an entry in the `/App_Config/commands.config` file mapping a command code to the signature of your class.
3. Add a button definition item using the `System/WebEdit/WebEdit Button` data template in the Core database, under the `/sitecore/content/Applications/WebEdit/Custom Experience Buttons` item, using the Content Editor.

You can use the solution shown in Listing 7-10 to add a Page Editor command that resets a field to its standard value. As shown in this example, Sitecore developers often use pipelines to interact with the user (in this case, to confirm they meant to reset the field).



Available for download on Wrox.com

LISTING 7-10: Adding a Page Editor command, ResetField.cs

```
namespace SitecoreBook.Shell.Applications.WebEdit.Commands
{
  using System.Collections.Specialized;

  // Page Editor field command to reset a field to its standard value
  public class ResetField
    : Sitecore.Shell.Applications.WebEdit.Commands.WebEditCommand
  {
    // Page Editor infrastructure invokes this method to call the command
    public override void Execute(
      Sitecore.Shell.Framework.Commands.CommandContext context)
    {
      // validate arguments
      Sitecore.Diagnostics.Assert.ArgumentNotNull(context, "context");
      Sitecore.Diagnostics.Assert.ArgumentNotNull(
        context.Parameters["field"],
        "field");

      // if unable to determine the item containing the field to reset, do nothing
```

www.it-ebooks.info

```
  if (context.Items == null
    || context.Items.Length != 1
    || context.Items[0] == null)
  {
    return;
  }

  // the item containing the field to reset
  Sitecore.Data.Items.Item item = context.Items[0];

  // construct key-value parameters for the pipeline
  // defined by the following Run() method
  NameValueCollection parameters = new NameValueCollection();

  // ID of the selected item
  parameters["id"] = item.ID.ToString();

  // database containing the selected item
  parameters["database"] = item.Database.ToString();

  // language of the selected item
  parameters["language"] = item.Language.ToString();

  // version of the selected item
  parameters["version"] = item.Version.ToString();

  // name of field in selected item associated with this command
  parameters["field"] = context.Parameters["field"];

  // invoke the pipeline
  Sitecore.Context.ClientPage.Start(this, "Run", parameters);
}

// command implementation
protected void Run(Sitecore.Web.UI.Sheer.ClientPipelineArgs args)
{
  // validate parameters
  Sitecore.Diagnostics.Assert.ArgumentNotNull(args.Parameters["field"], "field");
  Sitecore.Diagnostics.Assert.ArgumentNotNull(
    args.Parameters["database"],
    "database");
  Sitecore.Diagnostics.Assert.ArgumentNotNull(
    args.Parameters["language"],
    "language");
  Sitecore.Diagnostics.Assert.ArgumentNotNull(
    args.Parameters["version"],
    "version");

  // if the user has changed the item, prompt to save and save if approved
  // if the user chooses to cancel, do not reset the field value
  if (!Sitecore.Web.UI.Sheer.SheerResponse.CheckModified())
  {
    return;
  }
```

continues

www.it-ebooks.info

LISTING 7-10 *(continued)*

```
    // if the user has not already confirmed that
    // they want to reset the field to its standard value
    // then prompt and wait for confirmation or cancellation.
    if (!args.IsPostBack)
    {
      string message = Sitecore.Globalization.Translate.Text(
        "Reset {0} to its standard value?",
        new object[] { args.Parameters["field"] });
      Sitecore.Web.UI.Sheer.SheerResponse.Confirm(message);
      args.WaitForPostBack();
    }

    // if the user did not confirm the reset, do nothing
    if ((!args.HasResult) || args.Result != "yes")
    {
      return;
    }

    // the database containing the item
    Sitecore.Data.Database database = Sitecore.Configuration.Factory.GetDatabase(
      args.Parameters["database"]);

    // the item containing the version
    Sitecore.Globalization.Language language =
      Sitecore.Globalization.Language.Parse(args.Parameters["language"]);

    // the version containing the field value to reset
    Sitecore.Data.Version version = Sitecore.Data.Version.Parse(
      args.Parameters["version"]);

    // the item containing the field to reset
    Sitecore.Data.Items.Item item = database.GetItem(
      args.Parameters["id"],
      language,
      version);

    // field to reset
    Sitecore.Data.Fields.Field field = item.Fields[args.Parameters["field"]];

    // reset the field to its standard value
    using (new Sitecore.Data.Items.EditContext(item))
    {
      field.Reset();
    }
  }

  // indicate whether to show or hide the command in the chrome for the field
  public override Sitecore.Shell.Framework.Commands.CommandState QueryState(
    Sitecore.Shell.Framework.Commands.CommandContext context)
  {
    // validate arguments
    Sitecore.Diagnostics.Assert.ArgumentNotNull(context, "context");
```

www.it-ebooks.info

```
        Sitecore.Diagnostics.Assert.ArgumentNotNull(
          context.Parameters["field"],
          "field");

        // if unable to determine the item containing the field, disable the command
        if (context.Items == null
          || context.Items.Length != 1
          || context.Items[0] == null)
        {
          return Sitecore.Shell.Framework.Commands.CommandState.Disabled;
        }

        // the field to reset
        Sitecore.Data.Fields.Field field =
          context.Items[0].Fields[context.Parameters["field"]];

        // if unable to determine the item containing the field,
        // or if that field contains its standard value, disable the command
        if (field == null || field.ContainsStandardValue)
        {
          return Sitecore.Shell.Framework.Commands.CommandState.Disabled;
        }

        // otherwise, apply the default logic to determine the state of the command
        return base.QueryState(context);
      }
    }
  }
```

To make this command available to all fields rendered using the `renderField` pipeline:

1. Create a `/configuration/command` element in the `/App_Config/commands.config` file.
2. Set the `name` attribute to a code that uniquely identifies the command, such as `sitecorebook:webedit:resetfield`.
3. Set the `type` attribute to the signature of your class.
4. Click the database icon in the lower-right corner of the Sitecore desktop, and then select Core from the menu that appears to work with items in the Core database.
5. Open the Content Editor and navigate to the `/sitecore/content/Applications/WebEdit/Common Field Buttons` item.
6. Insert a Page Editor command definition item named Reset using the `System/WebEdit/WebEdit Button data` template.
7. Enter **Reset** in the Header field, `Applications/16x16/checkbox.png` for Icon, the command code (such as `sitecorebook:webedit:resetfield`) for Click, and select Common for Type in the Data section. Save the item.
8. Click the database icon in the lower-right corner of the Sitecore desktop, and then select Master from the menu that appears to revert to editing the Master database.

www.it-ebooks.info

Now, in the Page Editor, when you click the More drop-down for a field, you can use this command to reset the field to its standard value as shown in Figure 7-6.

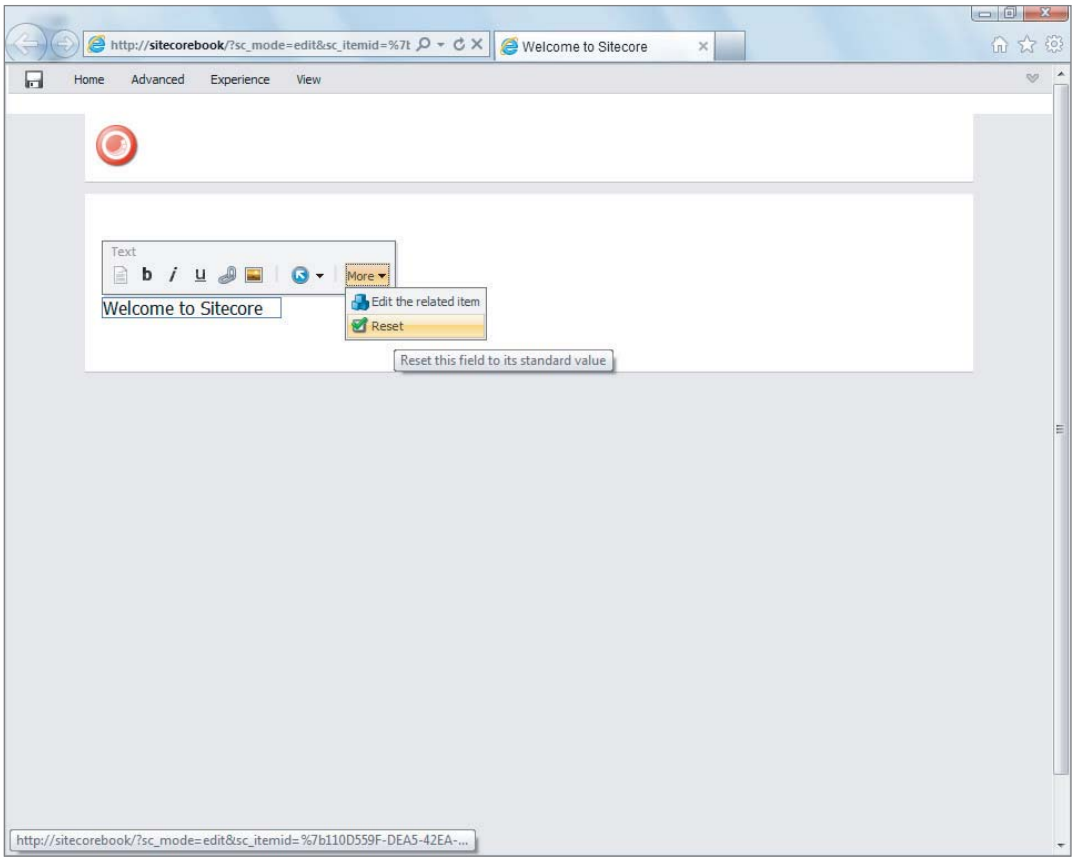


FIGURE 7-6

Click the Reset command. If you have made changes to the item, Sitecore displays a dialog asking whether you want to save the changes to the item, as shown in Figure 7-7, before resetting the field to its standard value.

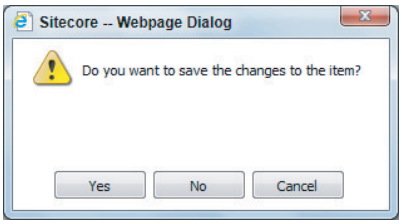


FIGURE 7-7

www.it-ebooks.info

If you click Cancel, Sitecore does nothing. If you click No, Sitecore discards the changes to the item. If you click Yes, Sitecore saves the item. In either case, Sitecore then prompts for confirmation as shown in Figure 7-8 before proceeding to reset the field to its standard value.

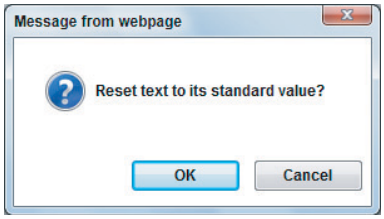


FIGURE 7-8

Click OK to reset the field to its standard value. The Page Editor refreshes to show that the field now contains its standard value. Due to the `QueryState()` method implementation, the Reset command does not appear in the More menu for the field because the field already contains its standard value.



Don't confuse resetting a field to its standard value with discarding your changes. You can set access rights on the definition item for the Reset command in the Core database to control who can invoke that command.

Use these instructions to add a Page Editor command to the following:

- **An individual field** — Select the command in the Page Editor Buttons field in the Data section of the item that defines the data template field.
- **All fields of a specific type** — Insert an item using the `System/WebEdit/WebEdit Button` data template under the WebEdit Buttons child of the field type definition item (such as `/sitecore/system/Field types/Link Types/General Link/WebEdit Buttons` for the General Link field type) in the Core database.
- **All fields** — Add a button definition item using the `System/WebEdit/WebEdit Button` data template under the `/sitecore/content/Applications/WebEdit/Common Field Buttons` item in the Core database, and select Custom in the Type field in the Data section as described in the previous example.
- **An individual rendering** — In layout details or in the rendering definition item, select the command in the Page Editor Buttons field of the Editor Options section.
- **All renderings** — Add one or more button definition items using the `System/WebEdit/WebEdit Button` data template under the `/Sitecore/Content/Applications/WebEdit/Default Rendering Buttons` item in the Core database.
- **All placeholders** — Add one or more button definition items using the `System/WebEdit/WebEdit Button` data template under the `/sitecore/content/Applications/WebEdit/Default Placeholder Buttons` item in the Core database.

www.it-ebooks.info

TAKE HOME POINTS

There are probably more ways to customize Sitecore than there are Sitecore developers in the world. I have worked with Sitecore for more than seven years and I still find more features to explore than I have time to investigate. Although this chapter only touched on what is possible with the Sitecore ASP.NET CMS, it should have provided some ideas to help you approach specific objectives with your Sitecore solutions.

Before you extend the CMS, ensure that you cannot achieve your objectives using default functionality. To minimize the learning and development effort as well as the chance of introducing conflicts with existing components, use the simplest possible approach that meets your requirements. For example, Content Editor warnings, validation, and even custom ribbon commands are simple, lightweight, and insulated well enough from other components to reduce the chance of interference with other functionality. Custom user interfaces and complex pipeline processors present more effort and risk, especially during upgrades.

Take advantage of the configuration factory rather than hard-coding type references. For your convenience, implement extension methods that .NET supports natively. Employ the rules engine to let CMS users configure processing logic. Use scheduled processes to invoke long-running jobs in the background. Always ensure that you optimize usability in the Page Editor.

www.it-ebooks.info

8

Automated Testing

WHAT'S IN THIS CHAPTER?

- Understanding automated testing
- Creating test projects
- Testing through an HTTP interface
- Interacting with a page under test through a browser
- Limiting the scope of tests for presentation components
- Calling the Sitecore API
- Creating test data

This chapter introduces general concepts involved in automated testing, and then describes a number of techniques that you can use to create and invoke automated tests for your Sitecore solutions. These techniques address various challenges that you will face when designing tests for code that executes in the context of a third-party system — in this case, the Sitecore ASP.NET web content management system (CMS). This chapter describes these approaches, including their advantages and disadvantages to help you determine when each is appropriate.

After explaining some of the complexities involved in automated testing of Sitecore solutions, this chapter presents three specific testing techniques, each of which address some of those challenges. These techniques include the following:

- **Testing over HTTP** — .NET code run on a remote system retrieves a web page from the Sitecore solution, parses that page, and then ensures that the parsed content meets expectations.

www.it-ebooks.info

- **Testing with a web browser driver** — .NET code run on a remote system instantiates a web browser to retrieve a web page from the Sitecore solution, and then ensures that the resulting representation of the page meets expectations.
- **Instantiating controls** — .NET code on the Sitecore server instantiates presentation controls, and then ensures that the output meets expectations.

In addition to providing details about each of these approaches, this chapter describes some considerations for managing data used by your testing solution, and then describes some considerations for tests that call Sitecore application programming interfaces (APIs) directly, both with an HTTP context and without. It concludes by providing some direction for assessing which components of the project will most benefit from automated testing.

UNDERSTANDING AUTOMATED TESTING

Automated testing is the act of programmatically verifying aspects of a piece of code. One benefit of automated tests is that you can execute them with minimal effort. This can save you a great deal of time in the later phases of a project by providing a set of regression tests that can be run repeatedly without requiring significant manual effort.

Automated testing achieves two objectives. First, it verifies that code functions as expected. Second, it provides a safety net to prevent the introduction of defects when you update existing code. Writing automated tests requires you to think about the conditions under which the code must operate, and to ensure that the results of code are predictable under those conditions.

A common practice when writing *unit tests*, which are a form of automated test, is to provide many different values for a required parameter, including normal expected values but also invalid values, very large values, very low values, `0` and `null`. By creating tests for all of these conditions, it becomes quite easy to uncover defects due to common mistakes such as not checking whether a parameter passed to a method is `null`.

Having the capability to rerun the tests easily is meaningless unless the tests are predictable and return the same results each time you execute them under equivalent conditions. A good automated test is repeatable, reliable, and resilient.

- *Repeatable* means the test will always return the same result given the same inputs.
- *Reliable* means you can trust that the tests cover enough conditions to give a true reflection of the completeness and quality of the code under test.
- *Resilient* means creating code that does not fail after the introduction of minor or insignificant changes.

An example of resiliency is a test that verifies the output of a page for the structure of a latest news component. Changes to other components on the page, such as a navigational element across the top of the page, should have no effect on tests for the latest news component.

www.it-ebooks.info



One famous blogger who writes about unit testing in .NET is Roy Osherove, author of the book The Art of Unit Testing *(ISBN-13: 978-1933988276, Manning Publications). Roy maintains a current definition of a good unit test at* `http://bit.ly/n7DqGI`. *Using Roy's definition, the techniques covered in this chapter are integration tests, as they are not isolated from Sitecore and do not run entirely in memory. To avoid confusion between unit tests and integration tests, and because the difference between the two can be subjective at times, this book refers to these combined topics as automated tests. When searching for additional sources to gain a deeper understanding of the material presented in this chapter, you may find the best results by searching for "unit testing" rather than "automated testing."*

COMPLEXITIES OF TESTING AGAINST SITECORE

In a standalone .NET application such as a command-line tool, the developer has full control of the code and the environment in which the code runs as shown in the simplified example in Figure 8-1.

This is not the case with ASP.NET applications such as Sitecore. Your code runs in the context of the ASP.NET environment and the Sitecore application built on top of the ASP.NET framework, which differs significantly from the context of a command-line tool. The ASP.NET framework is always at the top of the method call stack for code that you develop for that environment. Functioning something like a logical layer between the developer and ASP.NET itself, Sitecore extends the ASP.NET framework. When designing tests for Sitecore components, you must consider this context, which generally depends on an HTTP request. Figure 8-2 shows a typical flow of control for a Sitecore application.

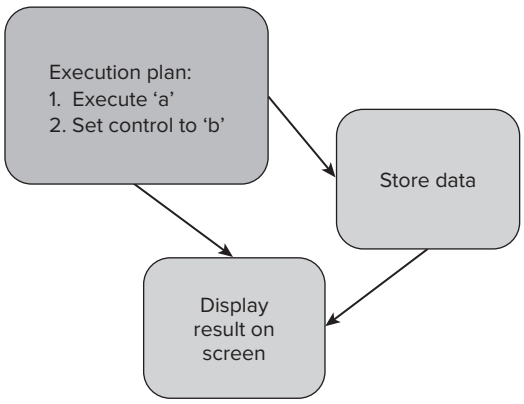


FIGURE 8-1

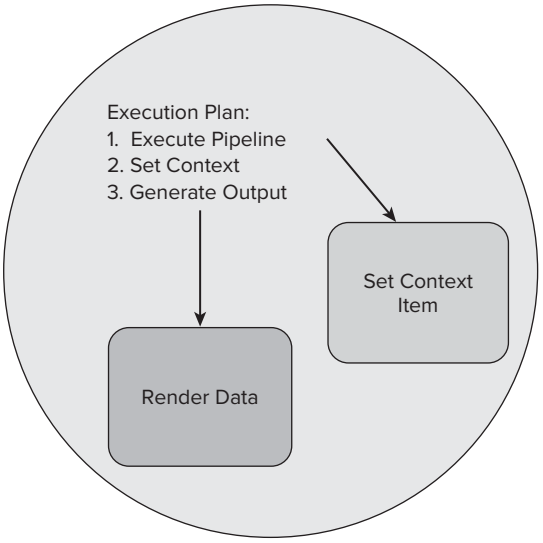


FIGURE 8-2

www.it-ebooks.info

Some code may depend on the context that invokes that code, such as the HTTP context exposed to ASP.NET applications by the `System.Web.HttpContext` (HTTP context) class provide by the `System.Web.Context.Current` property. Sitecore is an ASP.NET application that services requests sent to it through HTTP, with pieces of that application heavily dependent upon the HTTP context. To aid in testing, to the extent possible, endeavor to ensure that no code in your Sitecore solutions depends on the Sitecore context or the HTTP context.

Specifically, your code should not rely directly upon the Sitecore context, such as static properties of the `Sitecore.Context` class. Relying on the Sitecore context makes the code much harder to test, as you cannot simply supply the test values to a method or class, but must set up the Sitecore context before calling the method or class under test.

A well-designed solution exposes an API that you can easily test. This is in fact one way to measure the quality of the implementation. If the majority of methods and classes allow you to call them directly from test code, then that indicates a well-considered architecture abstracted to an acceptable degree. A bad design can be extremely hard to test, requiring the developer to implement workarounds for various issues when writing tests.



As you write code, it is vital to consider how you can test it. For example, when writing a method, you can access the context item or you can pass the context item to the method. Not only is a method that accesses the context item more limited than a method that accepts an item as a parameter, but testing a method that uses the context item is more difficult than testing a method that accepts an item as a parameter. Small design decisions like this can lead to code that is much easier to test and reduces testing requirements. You can find valuable suggestions for writing testable code on the Google testing blog at `http://bit.ly/2UVnhs`.

TESTING TECHNIQUES FOR SITECORE

This section details various techniques that you can apply to a Sitecore project to enable testing of the major project components. You can use the following techniques described throughout this book to test presentation components including layouts, sublayouts, and web controls:

- Testing through HTTP (.NET code to simulate a limited web browser)
- Testing using a web browser driver (.NET code to manipulate an actual browser)
- Instantiate controls (.NET code to invoke individual presentation components)

These techniques focus on the markup produced by the component. They can verify the markup or the inferred behavior and functionality encapsulated by the component.

You can use the following techniques to test components not related to presentation, such as methods and other code that calls the Sitecore API to verify that the code behaves as expected:

- Calling the Sitecore API directly (.NET code run on the Sitecore instance)
- Calling the Sitecore API outside of an HTTP context (.NET code calling Sitecore APIs without a Sitecore instance)

www.it-ebooks.info

The latter techniques focus more on the functionality and logic of sections of the project. A project will result in a more rational architecture and structure when you write code that enables testing using these techniques.

THE TEST PROJECT

You can write an automated test in any manner that you like, but the most common way is to use a framework specifically designed for unit testing. A number of unit testing frameworks are available for use with .NET applications. This chapter exlpains how to use NUnit, an open-source unit testing framework that is freely available for download from `www.nunit.org`.

NUnit exposes an API, and you write NUnit tests in .NET code using that API. It is a good idea to write the unit tests in a separate project from the code under test. This helps isolate the production code from the test code and makes it easy to remove the test code before deployment to production. This in turn decreases the attack surface of the application for malicious users. The testing techniques covered in this section use a class library project to maintain the tests.

Creating the Test Project

The following instructions describe how to obtain NUnit and create a test project you can use to hold the automated tests. Later, you will use a test runner, which is a software component specifically designed for this purpose, to invoke the tests you define in your test project.

Before you can use NUnit in a test project, you must download NUnit from `www.nunit.org`. You can download and expand a `.zip` file containing NUnit or you can download and run the `.msi` (Windows Installer Package). Whichever approach you use, I recommend that you download the latest stable version rather than a beta of NUnit. The location of the NUnit test runner program (`nunit.exe`) and assemblies (`.dll` files) depends on whether you install using an `.msi` or a `.zip` release of NUnit. When I used the `.msi` to install NUnit, files and subdirectories appeared under the `C:\Program Files (x86)\NUnit 2.6` subdirectory, while I controlled the location of expansion for the `.zip`. In both cases, the subdirectory structure within the root NUnit subdirectory was equivalent, but the `.zip` installation included files not installed by the `.msi`, which depend on the options you select during installation.



After my initial NUnit installation using the `.msi`*, the NUnit Windows application (*`nunit.exe`*) raised error messages about missing files such as* `pnunit-agent.exe` *until I reinstalled using the* `.msi` *and chose the PNUnit Runner option. If you use the* `.msi` *to install NUnit and experience this issue, reinstall NUnit, select the Custom Installation option, and include PNUnit Runner (Parallel NUnit Runner). The* `.zip` *automatically includes PNUnit. Based on this experience, I would just use the* `.zip` *file in the future. Bless open-source software — I love the idea, but each piece of each version of each ware usually sneezes on me a few times initially.*

www.it-ebooks.info

Follow these steps to create a new class project for unit tests with the existing Visual Studio solution or your Sitecore project:

1. Open the existing Sitecore solution in Visual Studio.
2. Right-click the top element in Solution Explorer, which represents the solution. In the context menu that appears, click Add ➪ New Project. The Add New Project dialog appears as shown in Figure 8-3.

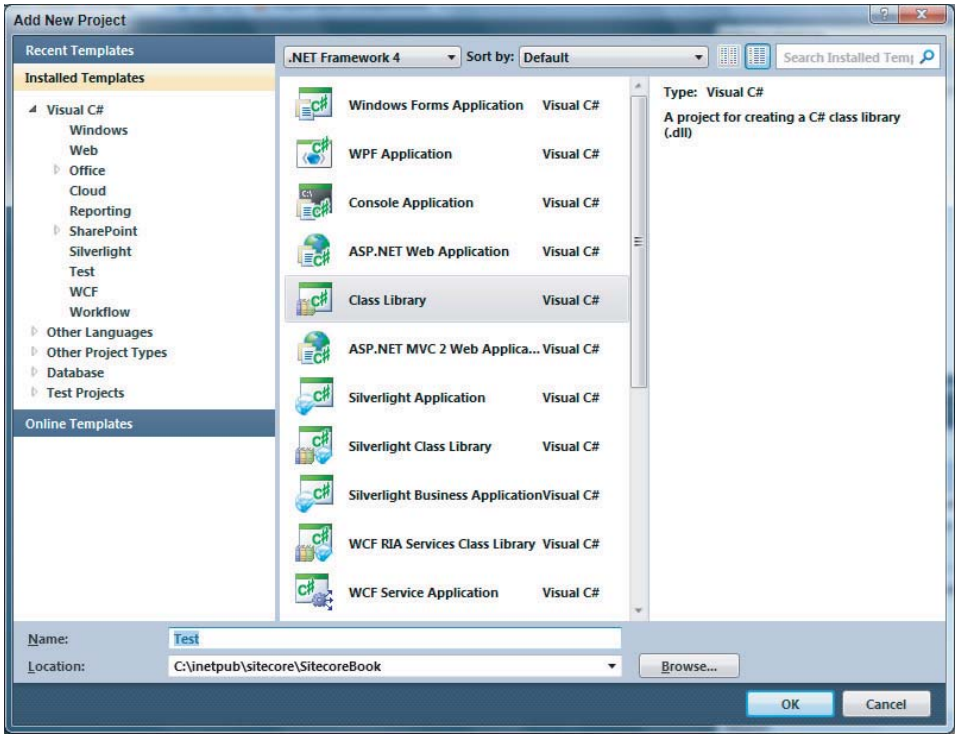


FIGURE 8-3

3. Select the Class Library project model.
4. In the Name field, enter a name for the unit test project, such as `SitecoreBook.Tests`, and then click OK. The Add New Project dialog disappears and you return to Visual Studio.
5. Optionally, select your new project in the solution explorer and double-click Properties to enter values in the Assembly name, Default namespace, and potentially other fields to control project properties. The examples in this chapter assume the same namespace and assembly as the test project name: `SitecoreBook.Tests` and `SitecoreBook.Tests.dll`, respectively.

To make the NUnit APIs available to the project:

1. To add a subdirectory to your test project to contain the NUnit assembly (`.dll` file) that contains the NUnit APIs your tests will use, right-click the test project in Solution Explorer.

www.it-ebooks.info

In the context menu that appears, click Add ➪ New Folder. Name the new subdirectory **lib** (short for library).

2. Copy the `nunit.framework.dll` assembly from the `/bin/framework` subdirectory of your NUnit installation subdirectory to the `/lib` subdirectory within your test project.

3. Right-click the References element in the test project in Solution Explorer, and then click Add Reference. The Add Reference dialog appears as shown in Figure 8-4.

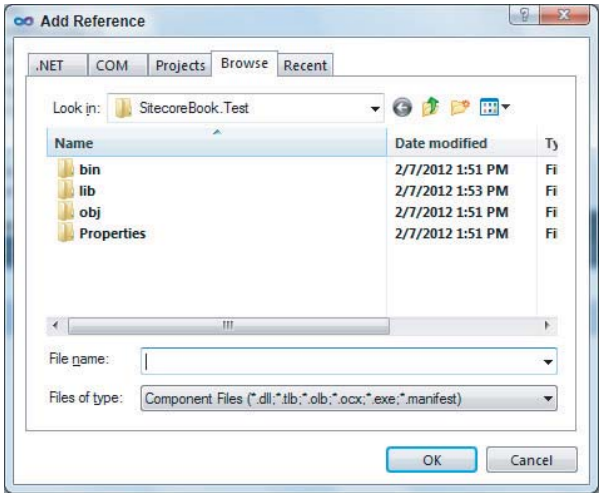


FIGURE 8-4

4. Click the Browse tab, navigate to the `/lib` subdirectory, select the `nunit.framework.dll` assembly, and then click OK. The Add Reference dialog disappears and you return to Visual Studio. Now you can add test fixtures (classes that contain tests) to your test project. To indicate that a class in the test project contains NUnit tests, add the `[NUnit.Framework.TestFixture]` attribute before the class definition as shown in the following example:

```
namespace SitecoreBook.Tests
{
    [NUnit.Framework.TestFixture]
    public class MyTestClass
    {
    }
}
```

Alternatively, you can add the following `using` statement to the class and then use the `[TextFixture]` attribute without the namespace:

```
using NUnit.Framework;
```

For clarity outside of an integrated development environment (IDE) such as Visual Studio, except for namespaces under `System`, this book always includes namespaces inline rather than relying on a `using` directive. Whether you qualify all types with namespaces or include `using` directives, when

www.it-ebooks.info

you use a test runner, NUnit attempts to load tests only from those classes attributed as `[NUnit.Framework.TextFixture]`.

Each test is a method of a test fixture class, which is a class attributed with `[NUnit.Framework.TextFixture]`. After attributing the class as a test fixture, add the `[NUnit.Framework.Test]` attribute to each method that you want the test runner to invoke automatically when testing your solution. Test runners invoke methods decorated with the `[NUnit.Framework.Test]` attribute in classes decorated with the `[NUnit.Framework.TextFixture]` attribute automatically. You can include any number of test methods in a single test fixture class. Test methods do not accept arguments. You can include methods without the `[NUnit.Framework.Test]` attribute in a test fixture class, such as to contain utility routines used by one or more of the actual tests. The test runner does not invoke such methods directly or automatically. Listing 8-1 provides an example of a single test.



Optionally, you can add the `[NUnit.Framework.Category]` *attribute to your class to define a category for tests in that class as demonstrated in Listing 8-1. When you use a test runner to invoke the tests as described later in this chapter, you can select one or more test categories, or run all tests in all categories. Add the* `[NUnit.Framework.Category]` *attribute to entire test fixture classes; it has no effect if you add it to individual test methods.*



Available for download on Wrox.com

LISTING 8-1: Example test

```
namespace SitecoreBook.Tests
{
  using System;

  [NUnit.Framework.TestFixture]
  [NUnit.Framework.Category("ExampleTests")]
  public class ExampleTests
  {
    [NUnit.Framework.Test]
    public void TestOne()
    {
      // Populate result from calling code under test
      string result = "output";

      // Perform assertions on test execution
      NUnit.Framework.Assert.AreEqual("output", result);
    }
  }
}
```

A typical test case would set the variable `result` shown in Listing 8-1 by performing an operation such as calling a method in a class under test. The test then asserts that the value of the variable `result` matches expectations after that method call. This example uses the static `NUnit.Framework.Assert.AreEqual()` assertion method, which verifies that both parameters have equal values. Unlike other assertion methods with which you may be familiar, failing an NUnit assertion does not raise an exception. If the assertion succeeds, the test passes; if the assertion fails, the test fails, and appears in failure reports generated by the test runner.

www.it-ebooks.info

Running the Tests

To execute the tests, a test runner must load the testing assembly. NUnit ships with a standalone test runner that you can use to run the tests using a Windows application (`nunit.exe`). To run the test in the preceding section, follow these steps:

1. If you have not already done so already, build the test project to generate an assembly (`.dll` file) containing the test fixture classes.
2. Invoke the `nunit.exe` application in the appropriate subdirectory of your NUnit installation or from the Windows Start menu if you used the `.msi` to install NUnit. The NUnit test runner appears as shown in Figure 8-5.

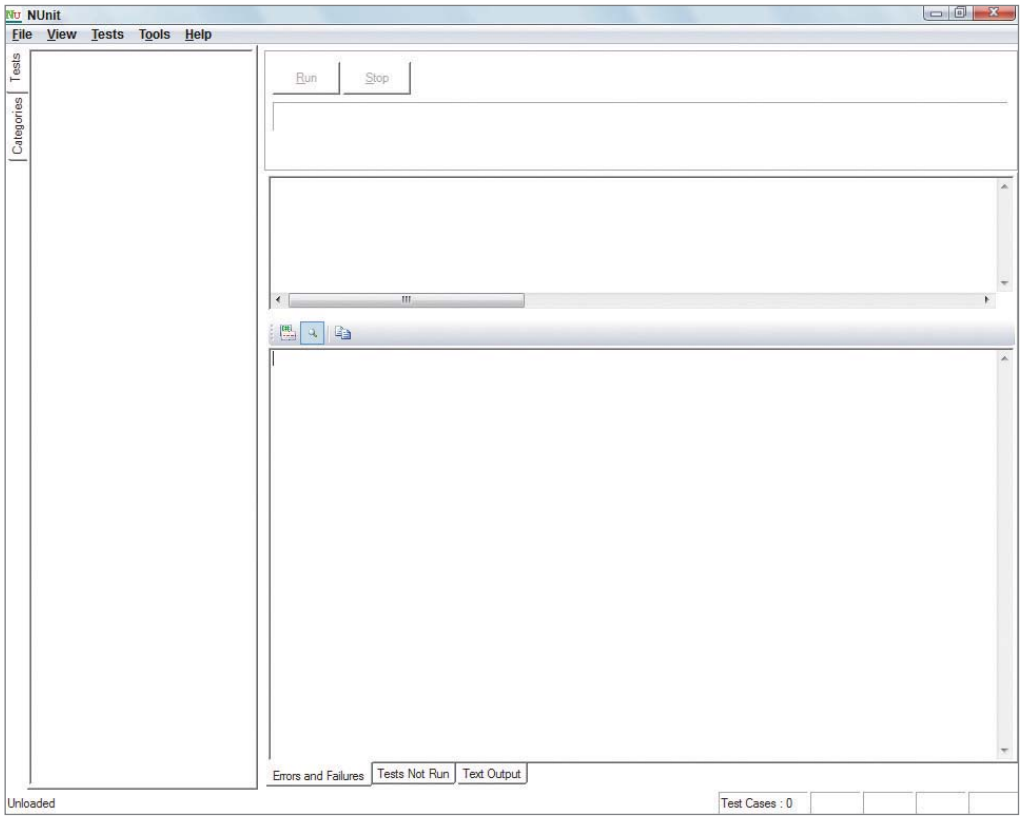


FIGURE 8-5

3. Select the File menu, and then select Open Project. In the Open Project dialog that appears, select the assembly generated by your test project (such as `/bin/debug/SitecoreBook.Tests.dll`, and then click Open. You see the NUnit test runner as shown in Figure 8-6.
4. Select namespace(s), class(es), or individual method(s) to test, and then click Run. If the test passes, you see only green. If one or more tests failed, you see red and one or more error messages for you to resolve. Figure 8-7 shows a test that passes.

www.it-ebooks.info

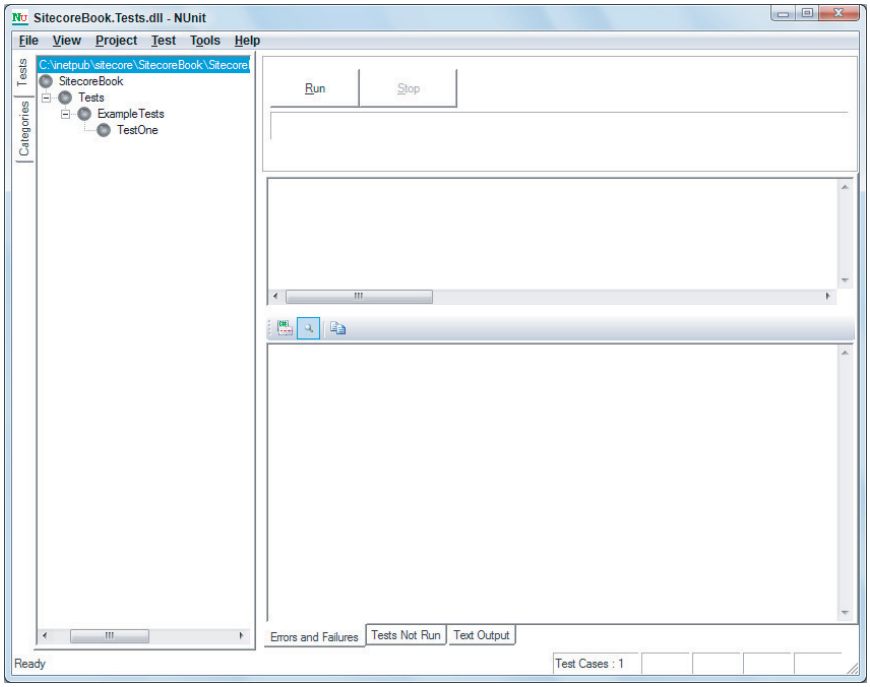


FIGURE 8-6

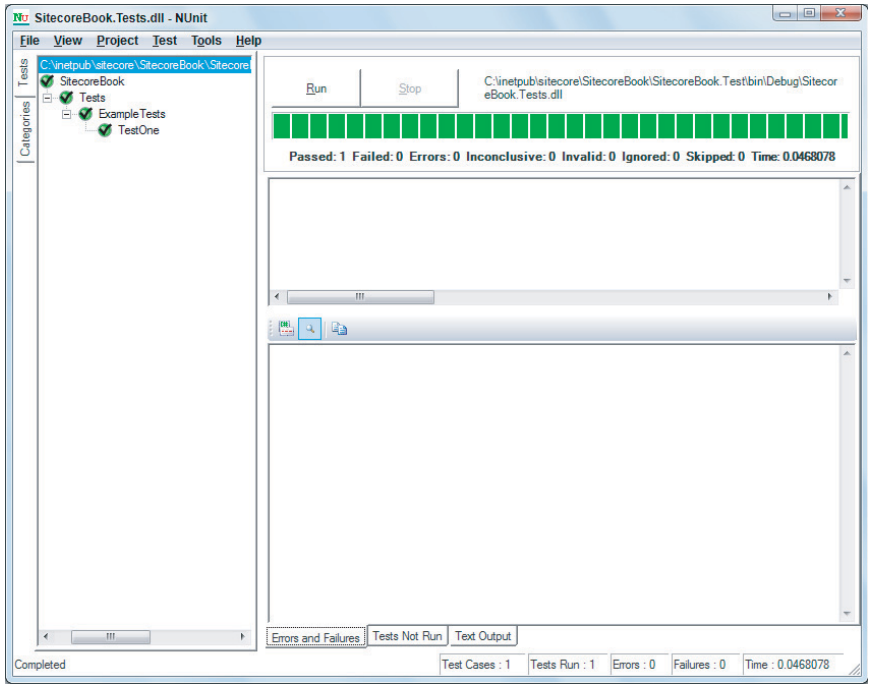


FIGURE 8-7

www.it-ebooks.info

Several types of extensions for Visual Studio enable you to run tests directly inside the integrated development environment (IDE). These include TestDriven.NET (`http://testdriven.net`), Visual NUnit (`http://bit.ly/cbkhrj`) from Ohloh (`www.ohloh.net`), and ReSharper (`http://bit.ly/ybr42J`) from JetBrains (`www.jetbrains.com`). For example, ReSharper exposes testing features in the sidebar at the left when editing test fixture classes that contain NUnit tests as shown in Figure 8-8.

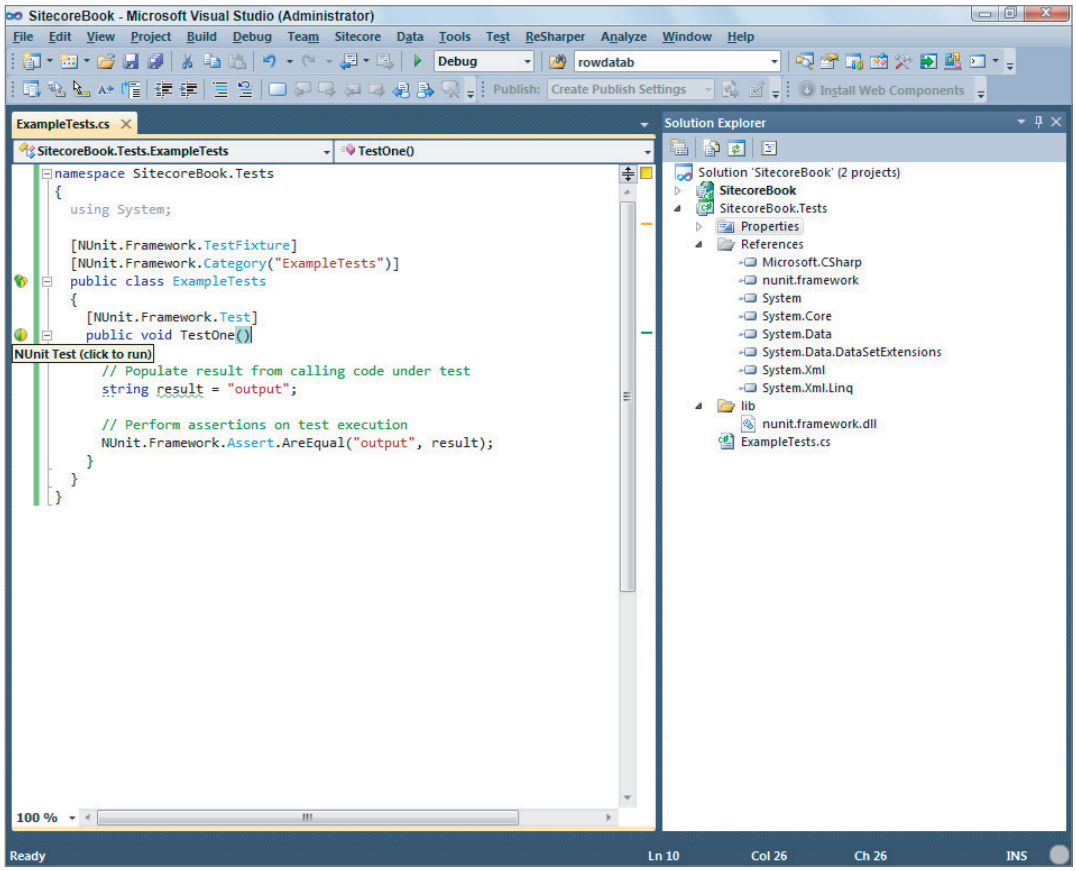


FIGURE 8-8

Alternatively, you can create a custom test runner using additional assemblies available in the NUnit release. For instructions to create a custom test runner that executes within an ASP.NET application, which gives you access to a complete Sitecore context and APIs, including an HTTP context, see the section "Testing with an Embedded Test Runner" later in this chapter.

TESTING WITH THE HTTP PROTOCOL

Almost every web solution provides an interface that enables external applications to request resources using the HTTP protocol. To exercise this interface, you can use a browser or an application such as Wget, or you can construct a custom application that issues simple HTTP requests and returns the result as text. For more information about Wget, see `http://bit.ly/A2k8tE`.

www.it-ebooks.info

The HTTP interface is something like the lowest common denominator for testing web applications. Regardless of the technology used to implement the solution, the application exposes an interface that you can access using HTTP. This section explains how you can use the HTTP interface to test specific aspects of a web solution.

Implementing Tests over HTTP

The basic outline of a process that implements testing over the HTTP protocol is as follows:

1. Request the resource such as a web page — for example, using an instance of the `System.Web.WebRequest` class.
2. Parse the page response — for example, using the HTML Agility Pack (HAP) described in Chapter 7 and later in this section.
3. Verify the results and perform the test assertions.

The web application should respond with text, which your program can examine using techniques such as regular expressions, the HAP, the `Contains()` method in the `System.String` class, or any other technique that meets your requirements. Modern web applications generally output well-formed eXtensible Markup Language (XML), including eXtensible HyperText Markup Language (XHTML), which enables you to use XPath in assertions against the structure of that output. Although it is not impossible, verifying a complex output structure using this approach can be quite difficult.

Your Sitecore solutions should almost certainly generate XML, which is similar to HTML in that it is a markup language using elements and attributes, but applies additional restrictions. Some of the most important characteristics that differentiate well-formed XML and hence XHTML from HTML include the following:

- The markup contains only legal Unicode characters. For more information about Unicode, see `http://bit.ly/xHhd3r`.
- The markup contains a single root element, such as the `<html>` element in an XHTML document.
- The markup uses symbols such as the < (less than) character and `&` (ampersand) character only for markup-delineation purposes, and uses XML entities such as `<` and `&` as alternatives (escape sequences) where they exist in content or other data.
- The markup wraps all attribute values with matching opening and closing single-quote or double-quote characters.
- All elements are properly nested, meaning if you open an element within another element, you close the inner element before you close the outer element (e.g., `<html><body></body></html>` rather than `<html><body></html></body>`).
- Element names are case sensitive, and the case of the opening and closing elements must match (e.g., `<html></html>` rather than `<HTML></html>`).

Not all web applications output well-formed XML. If you attempt to create an object that represents an XML document from malformed markup, you will likely experience exceptions or other

www.it-ebooks.info

errors. You can sometimes use the open-source HAP library mentioned previously to mitigate issues with malformed XML. The HAP can often parse invalid XML and even HTML, and can return valid XML, enabling you to work with the response as you would work with valid XML. The HAP can convert that content to an object using the `System.Xml.XPath.XPathNavigator` class, which enables you to invoke XPath queries against content parsed from HTML or invalid XML.

Sitecore CMS conveniently includes the HAP for its own internal purposes, so you do not have to download anything in order to use it. You do have to reference the `HtmlAgilityPack.dll` assembly in your project in order to use the HAP, and you must copy that file to the subdirectory containing the assembly (`.dll` file) that your test project generates to contain your text fixture classes.

The following code shows how to request the home page of a Sitecore solution at the URL `http://sitecorebook` using the HTTP protocol, and how to retrieve the textual output returned from that application, which represents the content of the home page. This code depends on classes in the `System, System.IO, System.Net`, and `System.Xml.XPath` namespaces, with which I assume you are already familiar:

```
// Request the home page to test using HTTP
WebRequest request = HttpWebRequest.Create("http://sitecorebook");
WebResponse response = request.GetResponse();

// Get the response stream to read the response from
Stream responseStream = response.GetResponseStream();

// Read the output as text
StreamReader reader = new StreamReader(responseStream);
String output = reader.ReadToEnd();

// Close the reader to free up resources
reader.Close();
```

The example code first makes a request to a URL used to access the Sitecore instance and retrieves the response. When implementing your own test based on the example code, be sure to replace `http://sitecorebook` in this code with the actual URL of the page to test.

The code then wraps that response stream in a `StreamReader`, which makes it easy to convert the response to a string. Remember to invoke the `Close()` method on the `System.IO.StreamReader` object after reading the stream, which automatically closes the stream that `System.IO.StreamReader` wraps, implicitly releasing any resources currently held by the stream.



For brevity and readability, all code in this chapter intentionally omits defensive constructs. You should check each variable for `null` *and other potential error conditions in the testing code itself.*

You can now use the HAP to parse the page using an XML document as shown in the following example:

```
// Create a new HtmlAgilityPack document
HtmlAgilityPack.HtmlDocument document =
```

www.it-ebooks.info

```
    new HtmlAgilityPack.HtmlDocument();

// Load the textual output of the page into the HtmlAgilityPack document
document.LoadHtml(output);

// Create an XML document navigator for evaluating XPath expressions
XPathNavigator nav = document.CreateNavigator();
```

After loading the textual output of the page into the `HtmlDocument`, this code uses the HAP to parse the document to create a valid XML document. Then it creates an instance of the `System.Xml.XPath.XPathNavigator` class from that document in order to traverse the parsed content.



Because the HAP representation differs from the original markup, this approach cannot test various specifics of the original response, such as the presence, absence, or volume of whitespace, nor whether the markup contains single quote characters (') or double quote characters (") in specific locations. Remember that the HAP may also correct problems with the markup, preventing you from seeing those issues. Most modern web browsers perform similar corrections. For more information about this approach, see the following section.

Assume that the home page should include the following markup and you want to test only some aspects of only that section of the response (not including the values of the `href` attributes, the depth of the `<nav>` element in the HTML superstructure, whether this structure includes any unexpected markup, or various other aspects of the response):

```
<nav class="primary">
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About Us</a></li>
    <li><a href="#">Partners</a></li>
  </ul>
</nav>
```



Many of the tests used in the remainder of this chapter expect this markup format. If you compile the code provided into a project to develop some experience with testing, you may want to enter this markup as the value of a field of an item in your Sitecore solution and publish that item so that the tests can pass. In a development environment, the `/sitecore/content/home` item may be convenient for this purpose.

The following code demonstrates how you can traverse the XML representation of the markup returned by the Sitecore solution to verify aspects of that markup, such as to find a particular element or set of elements and ensure that they conform to expectations:

```
// Find all list items within the primary nav element
XPathNodeIterator elements = nav.Select(
  "//nav[@class='primary']/ul/li");
```

www.it-ebooks.info

```
// Verify there are only 3 elements found
NUnit.Framework.Assert.AreEqual(
  3,
  elements.Count,
  "primary nav li count");

// Move to the first element
elements.MoveNext();

// Verify the text of the first link in the current element
NUnit.Framework.Assert.AreEqual(
  "Home",
  elements.Current.SelectSingleNode("a").Value,
  "text of first primary nav li");

// Move to the second element and verify the link text
elements.MoveNext();
NUnit.Framework.Assert.AreEqual(
  "About Us",
  elements.Current.SelectSingleNode("a").Value,
  "text of second primary nav li");

// Move to the third element and verify the link text
elements.MoveNext();
NUnit.Framework.Assert.AreEqual(
  "Partners",
  elements.Current.SelectSingleNode("a").Value,
  "text of third primary nav li");
```

The preceding example uses the `System.Xml.XPath.XPathNodeIterator` class to locate elements in the parsed XML using XPath, and then uses NUnit to verify limited specifics of those elements. The first assertion ensures the correct number of matching elements. The next three assertions verify the text of the first link found in each list item element. Note the calls to the `MoveNext()` method of the `System.Xml.XPath.XPathNodeIterator` object before each assertion, which progress that iterator to the next matching element.



Regardless of your testing strategy, tests can take longer to run than you might expect. Specifically, if you have not used the Sitecore solution since you last rebuilt that project, ASP.NET initializes the application when the testing code requests the home page, which adds to testing time.

NUnit reports only the first failed assertion in each tested method. If your class contains multiple test methods, NUnit can report multiple failed assertions, but only the first such assertion in each method. Because unit test methods generally should not accept arguments, this can present limitations when you want to investigate multiple error conditions for a single test.

One benefit of this approach is complete decoupling of the testing infrastructure from the technology used to create the web application. You can use this technique to test solutions built with

www.it-ebooks.info

a technology that does not expose a public API. You can run the test from a separate IIS website without Sitecore, and even from a separate server. Unfortunately, this approach is somewhat tedious and has a variety of limitations as described in the following section.



Regardless of your testing strategy, remember to decorate classes that contain tests with the `[NUnit.Framework.TestFixture]` *attribute, and the testing methods in the tested assembly within those classes with the* `[NUnit.Framework.Test]` *attribute. NUnit automatically invokes all methods with the* `[NUnit.Framework.Test]` *attribute in all classes with the* `[NUnit.Framework.TestFixture]` *attribute.*

Limitations of Testing over HTTP

The technique of testing by requesting pages and performing assertions on the textual response using HTTP presents a number of limitations. This technique does not use a web browser or any type of engine to process the output of the request and perform dynamic behaviors that can affect the Document Object Model (DOM) represented within the client, such as JavaScript and other aspects of a complete client-server interaction. The test runner used for this technique, which functions as a web client, does not evaluate the response from the server using the same mechanisms used by typical web browsers, nor does it address CSS, images, and other resource references embedded within that response. When you use this technique, tests can only assert against the preliminary fundamental structure of the page.

Additionally, this test runner always requests an entire page instead of evaluating a single component or control within a page. This can increase the difficulty of maintaining the tests, especially if they rely on a particular page structure to locate the markup to test.

Imagine that the response contains several instances of a control that you need to test. Assume that you can use XPath similar to the following to locate the markup of that control:

```
div[@class='main']/div/div/ul
```

This XPath expression fails if you remove any of the nested `<div>` elements for any reason. In such cases, the cause of test failure may not be immediately apparent, as other controls, such as a sublayout that someone has added around the control that you intend to test, may output tags that interfere with those tests.

This approach requires that developers understand the entire page structure in order to test a single component within that structure. In short, this technique depends on a relatively static markup structure; if you change that structure, you may need to update all the tests accordingly. Such tests are not resilient, reliable, or completely repeatable.

TESTING USING A WEB BROWSER DRIVER

To overcome the limitations of testing through HTTP, and specifically to realize the dynamic behaviors of the page such as those that result from JavaScript, tests can utilize a web browser to enable functionality of such page elements. You can control a web browser programmatically with a web browser driver such as Selenium (`www.seleniumhq.com`) or WatiN (`www.watin.org`).

www.it-ebooks.info

Implementing Testing with a Web Browser Driver

A web browser driver exposes an API that test code can call. Because this approach uses a real web browser, it realizes all normal browser behaviors, such as running JavaScript code in the client. This approach greatly simplifies testing of HTML forms that require interaction, and the web browser driver can post form entry data back to the server.

The basic process of testing with a web browser driver is as follows:

1. Instruct the browser driver to navigate to the page under test.
2. Use the browser driver to navigate through the page to retrieve elements to verify.
3. Verify such elements against expectations, such as by using NUnit assertions.

The code examples in this book use Selenium to control the Firefox web browser. To use Selenium you must first download it and add it to the test project, much like NUnit. To add Selenium to your test project:

1. Download the Selenium client drivers (`.zip` file) from `www.seleniumhq.com`. Be sure to download the C#/.NET release, as Selenium is also available for other platforms such as Java.
2. Copy the `WebDriver.dll` from the `/net40` subdirectory in the Selenium `.zip` file to the `/lib` subdirectory of your test project.
3. In your test project, add a reference to the `WebDriver.dll` assembly in the `/lib` subdirectory. When you compile your project, Visual Studio will copy this assembly to the directory in which it builds the assembly that contains your test fixture classes (typically the `/obj/debug` subdirectory).
4. Copy the `Ionic.Zip.dll` and `Newtonsoft.Json.dll` assemblies on which the `WebDriver.dll` assembly depends from the `/net40` subdirectory of the Selenium `.zip` file to the subdirectory in which Visual Studio builds your assembly (typically the `/obj/debug` subdirectory). Your project does not need to reference those assemblies.
5. Double-click Properties under your test project in Solution Explorer and configure that project to target .NET 4.0.

To use Firefox, you must install the browser available from `http://firefox.com`. Then you can use Selenium to invoke the Firefox browser to access the page under test. The following code shows an example of using Selenium to drive Firefox to access the home page at `http://sitecorebook`, select elements on the page, and verify that the number of elements matches the number you expect.




```
namespace SitecoreBook.Tests
{
  using System.Collections.ObjectModel;

  [NUnit.Framework.TestFixture]
  class TestHomePageWithFirefox
  {
      [NUnit.Framework.Test]
      public void PageTest()
      {
```

www.it-ebooks.info

```
            // Create a Firefox driver
            using (OpenQA.Selenium.Firefox.FirefoxDriver driver =
              new OpenQA.Selenium.Firefox.FirefoxDriver())
            {
               // Navigate to the page to test
               driver.Navigate().GoToUrl("http://sitecorebook");

               // Locate the list item elements of the primary navigation
               ReadOnlyCollection<OpenQA.Selenium.IWebElement> navElements =
                 driver.FindElements(
                   OpenQA.Selenium.By.XPath("//nav[@class='primary']/ul/li"));

               // Verify only 3 list items exist
               NUnit.Framework.Assert.AreEqual(3, navElements.Count);
            }
         }
      }
   }
```

TestHomePageWithFirefox.cs

The code used to create this test fixture contains a single test that relies on Selenium to drive the Firefox browser to access the home page at `http://sitecorebook`, uses an XPath expression to locate elements to verify within the page, and then uses an NUnit assert against the expected number of elements.

This code instantiates an object of the `OpenQA.Selenium.Firefox.FirefoxDriver` class wrapped in a C# `using` statement. When the application exits the scope of this `using` statement, the .NET Framework automatically closes the driver, which closes the Firefox window used for the tests and frees any resources it used. You might see the page load into that window before the browser closes.

Selenium enables you to locate elements on the page using a variety of techniques. You could alternatively populate the `navElements` variable in this example using a CSS selector instead of using XPath. Both techniques enable you to select zero or more elements in a block of markup such as XHTML. For more information about CSS selectors, see `http://bit.ly/y3uiAk`. The following code demonstrates how to use a CSS selector to locate the same elements selected with XPath in the previous example:

```
ReadOnlyCollection<OpenQA.Selenium.IWebElement> navElements = driver.FindElements(
  OpenQA.Selenium.By.CssSelector("nav.primary ul li"));
```

You can use Selenium to interact with the elements of the page to test dynamic behaviors. For example, if a page includes a form that contains a button with the ID `myButton`, and the page uses JavaScript to change the inner text of a `<div>` element with ID `theText` to `button clicked`, you can use Selenium to locate and "click" the button, which executes the JavaScript action associated with that button. You can then test the inner text of the `<div>`. The following code demonstrates this approach using the page at `http://sitecorebook/form.aspx`:




```
[NUnit.Framework.Test]
public void FormTest()
{
  // Create a Firefox driver
  using (OpenQA.Selenium.Firefox.FirefoxDriver driver =
    new OpenQA.Selenium.Firefox.FirefoxDriver())
```

www.it-ebooks.info

```
  {
    // Navigate to the page to test
    driver.Navigate().GoToUrl("http://sitecorebook/form.aspx");

    // Locate the button and click it
    OpenQA.Selenium.IWebElement button = driver.FindElement(
      OpenQA.Selenium.By.Id("theButton"));
    button.Click();

    // Locate the text to verify
    OpenQA.Selenium.IWebElement text =
      driver.FindElement(OpenQA.Selenium.By.Id("theText"));

    // Verify the text
    NUnit.Framework.Assert.AreEqual("button clicked", text.Text);
  }
}
```

TestFormWithFirefox.cs

This test creates a Firefox driver just as in the previous test. It then uses that driver to navigate the browser to the URL of the page containing the form to test. After loading that page, the test locates and clicks the button with ID `theButton`. In this instance, that simulated click actually occurs inside the Firefox browser controlled by your test method, and causes any JavaScript attached to the button to execute. The rest of the test locates the HTML element with ID `theText` and verifies that the inner text of that element is as expected by accessing the element's `Text` property.

Limitations of the Web Browser Driver Technique

Testing with a web browser driver overcomes one of the most significant limitations of testing over HTTP: Because it uses a real browser to execute the tests, testing with a web browser driver allows tests to account for JavaScript and dynamic behaviors of the page. However, this technique does not address the scope limitation common to employing HTTP directly: The browser loads an entire page to verify a single component of that page. Use a web browser driver when you need to test HTML forms and JavaScript.

TESTING WITH AN EMBEDDED TEST RUNNER

The previous testing techniques have shown how to test entire pages at a time. You will find it much easier to maintain tests with smaller scope. Page-based tests tend to break when they focus on a single component on the page and other parts of the page change. For example, if the component under test was originally inside a sublayout but has moved to another sublayout, the test code may not correctly locate the elements on the page to verify. Such a relatively brittle test would then fail.

To reduce the scope of the tests, you need to call the Sitecore API directly. To call the Sitecore API easily, you can write code that runs inside the Sitecore application, which depends on an HTTP context. Code run inside the Sitecore application has access to all Sitecore APIs, which will function as you expect.

www.it-ebooks.info



Running code inside the Sitecore application does not require placing that code in the same Visual Studio project as your Sitecore solution. You should still maintain a separate project for testing, and avoid deploying that solution to environments that you do not use specifically for testing.

The NUnit test runner used thus far executes one or more tests as a standalone application. To execute NUnit tests inside a Sitecore application, the test runner needs to run inside the Sitecore application. For this task, you can use the NUnit API to create a test runner that runs as an ASP .NET web form inside the Sitecore application.

Implementing an Embedded Test Runner

To implement an embedded test runner, you can create an ASP.NET project in Visual Studio. The previous examples in this chapter used a class library project to hold the tests; create a new project for the test runner in the same manner you use to create your initial Sitecore project (see Appendix B). You should use a new subdirectory for this project rather than the `/Website` subdirectory of your Sitecore installation. For example, create the project under `C:\inetpub\sitecore\SitecoreBook` rather than within `C:\inetpub\sitecore\SitecoreBook\Website`. You then need to copy the required files from this project to the `/Website` subdirectory to use the embedded test runner as described later in this section.

For ease of deployment, the embedded test runner demonstrated in this section loads and executes tests in the assembly that contains the test runner. This embedded test runner enables you to filter the tests by category, which makes it easier to run a small portion of the test suite, rather than run the entire suite each time you test.

The following process description summarizes how to create a Visual Studio project for an embedded test runner as described previously in this section.

1. Create a new ASP.NET web application project inside Visual Studio to hold the test runner. This project will hold the tests written to run inside the embedded test runner.
2. In the new project, add references to the following assemblies in your copy of NUnit as required to create the test runner and to write testing code:
   - `/framework/nunit.framework.dll`
   - `/lib/nunit.core.dll`
   - `/lib/nunit.core.interfaces.dll`
3. Add a new web form to the project and call it **TestRunner.aspx**. This web form will provide access to the embedded test runner.
4. Add the equivalent of the following to the markup file of the embedded test runner web form:




```
<%@ Page Language="C#" AutoEventWireup="true"
  CodeBehind="TestRunner.aspx.cs"
  Inherits="SitecoreBook.Tests.TestRunner" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

www.it-ebooks.info

```
<html>
  <head>
    <title>Embedded NUnit Test Runner</title>
    <style type="text/css">
      body { font-family: Tahoma, Arial, Sans-Serif; font-size: 12px; }
      h1 { font-size: 1.3em; }
      h2 { font-size: 1.2em; }
      .result { font-weight: bold; font-size: 1.1em; }
      .result-pass { color: Green; }
      .result-fail { color: Red; }
      .pass { background-color: Green; }
      .fail { background-color: Red; }
    </style>
  </head>
  <body>
    <form id="Form1" runat="server">
      <h1>Embedded NUnit Test Runner</h1>
      <p>
        <asp:Literal runat="server" ID="ltrStats" />
      </p>
      <p class="result">
        <asp:Label runat="server" ID="lblResult" />
      </p>
      <hr />
      <asp:GridView runat="server" ID="gvResults"
        OnRowDataBound="RowDataBound"/>
      <h2>Category Filter</h2>
      <asp:CheckBoxList runat="server" ID="cblCategories"
        RepeatDirection="Horizontal"/>
      <hr />
      <asp:Button ID="Button1" runat="server" Text="Run" OnClick="RunTests" />
    </form>
  </body>
</html>
```

TestRunner.apsx

To implement the code-behind for the embedded test runner web form you just created:

1. Add the following member variables to the code-behind class. Various methods of the page populate these variables used to report the outcome of the tests. The code snippets for this example depend on classes in the `System`, `System.Collections.Generic`, `System.Data`, `System.Reflection`, and `System.Web.UI.WebControls` namespaces. If they do not already exist, add `using` statements for those namespaces to your class.




```
private DataTable resultsTable;
private int executedCount = 0;
private int failedCount = 0;
private NUnit.Core.TestSuite testSuite = null;
```

TestRunner.aspx.cs

2. Add the following `Page_Load()` event handler method to the code-behind of the new web form, replacing any existing method of the same name. This code calls the `CreateTable()`

www.it-ebooks.info

method that you will create next to initialize the HTML table that will hold the results of the tests. It also initializes the TestSuite declared in the previous code snippet and loads the tests from the assembly that contains this compiled code-behind. If the current request is the initial request for the test runner, the following code also loads the categories from the tests and displays them in the list of checkboxes from which you select the categories of tests to invoke.

```
protected void Page_Load(object sender, EventArgs e)
{
  // Initialize the results table
  this.CreateTable();

  // Initialize NUnit
  NUnit.Core.CoreExtensions.Host.InitializeService();

  // Load test suite from current assembly
  this.LoadTestSuite();

  if (this.IsPostBack)
  {
    return;
  }

  // Populate categories for optional filtering
  List<string> categories = new List<string>();
  this.GetCategories(this.testSuite, categories);
  cblCategories.DataSource = categories;
  cblCategories.DataBind();
}
```

TestRunner.aspx.cs

3. Add the `CreateTable()` method to the code behind to initialize the results table:

```
private DataTable CreateTable()
{
  this.resultsTable = new DataTable();
  this.resultsTable.Columns.Add("Test Name");
  this.resultsTable.Columns.Add("Pass", typeof(bool));
  this.resultsTable.Columns.Add("Message");
  return this.resultsTable;
}
```

TestRunner.aspx.cs

4. Add the `LoadTestSuite()` method to load the tests from the same assembly as the test runner:

```
private void LoadTestSuite()
{
  NUnit.Core.TestPackage package = new NUnit.Core.TestPackage(
    Assembly.GetExecutingAssembly().Location);
  this.testSuite = new NUnit.Core.TestSuiteBuilder().Build(package);
}
```

TestRunner.aspx.cs

www.it-ebooks.info

5. Add the following `GetCategories()` method to read the categories of tests from the assembly (`.dll` file) that contains the test suite as specified by any `[NUnit.Framework.Category]` attributes defined for test fixture classes:

```
private void GetCategories(
  NUnit.Core.TestSuite suite,
  List<string> cats)
{
  // Check if the current suite contains any categories
  if (suite.Categories != null)
  {
    // Add unique categories to the list of found categories
    for (int i = 0; i < suite.Categories.Count; i++)
    {
      if (!cats.Contains((string)suite.Categories[i]))
      {
        cats.Add((string)suite.Categories[i]);
      }
    }
  }

  // Recurse into any nested test suites
  for (int i = 0; i < suite.Tests.Count; i++)
  {
    if (((NUnit.Core.ITest)suite.Tests[i]).IsSuite)
    {
      this.GetCategories((NUnit.Core.TestSuite)suite.Tests[i], cats);
    }
  }
}
```

TestRunner.aspx.cs

6. Add the `RunTest()` method that will handle the Click event for the Run button in the embedded test runner web form:

```
protected void RunTests(object sender, EventArgs args)
{
  // Create a list of any selected categories to filter by
  List<string> selectedCategories = new List<string>();

  foreach (ListItem item in cblCategories.Items)
  {
    if (item.Selected)
    {
      selectedCategories.Add(item.Value);
    }
  }

  // Create the category filter based on the selected categories
  NUnit.Core.Filters.CategoryFilter filter =
    new NUnit.Core.Filters.CategoryFilter(selectedCategories.ToArray());

  // Declare a variable to hold the results of the test run
```

www.it-ebooks.info

```
    NUnit.Core.TestResult result = null;

    // If the category filter is not empty
    // execute the test run with the category filter,
    // otherwise execute the test run with an empty filter
    if (filter.Categories.Count > 0)
    {
      result = this.testSuite.Run(this, filter);
    }
    else
    {
      result = this.testSuite.Run(
        this,
        NUnit.Core.TestFilter.Empty);
    }
    // Bind the collected results to the grid view
    gvResults.DataSource = this.resultsTable;
    gvResults.DataBind();

    // Display statistics
    ltrStats.Text = string.Format(
        "{0} out of {1} tests run in {2} seconds.",
        this.executedCount,
        result.Test.TestCount,
        result.Time);

    if (this.failedCount > 0)
    {
      ltrStats.Text += string.Format(
        "<br/>{0} {1} failed",
        this.failedCount,
        this.failedCount == 1 ? "test" : "tests");
    }

    // Display overall outcome of the test run
    lblResult.Text = "Suite " + (result.IsFailure ? "Failed" : "Passed");

    if (result.IsFailure)
    {
      this.lblResult.CssClass = "result-fail";
    }
    else
    {
      this.lblResult.CssClass = "result-pass";
    }
  }
```

TestRunner.aspx.cs

Before completing the embedded test runner implementation in the following steps, take a few minutes to understand this code. The `RunTests()` that ASP.NET invokes when you click the Run button in the embedded test runner first creates a list of all the names of categories that the user selected to run. If the user selected no categories, then the test runner invokes all the tests in the

www.it-ebooks.info

assembly under test. The code then creates a category filter based on the selected category names. If the user selects at least one category, the code invokes the test suite by calling the `Run()` method of `TestSuite` specifying the category filter to meet the required filter parameter. If the user selects no categories, the code passes an empty filter to the `Run()` method, which indicates that the method should invoke all tests in the assembly.

The `Run()` method requires a reference to an object that can receive feedback from the test runner regarding the outcome of each individual test invoked. To collect such information with ease, the embedded test runner is passed to the test suite. To enable the embedded test runner to receive the feedback from the test suite, it must implement the correct interface, as this section demonstrates further.

The call to the `Run()` method blocks, and will not return until all tests have run. At that point, the `DataTable` named `resultsTable` contains records populated by callbacks from the test suite to the embedded test runner. The code then binds the results table to the `GridView` control `gvResults`, and various statistics about the test run appear in other controls on the page.

To complete the embedded test runner:

1. Add the `RowDataBound` event handler for the `GridView` control in the page to change the background color of the test result cell in that table. This styling provides visual feedback about the state of each test. The code first confirms that the bound row is a data row, in which case it extracts data from the event arguments and applies the appropriate CSS class to the appropriate cell of the markup table to style that cell:




```
protected void RowDataBound(object sender, GridViewRowEventArgs args)
{
  // Ensure the row being bound is a data row
  if (args.Row.RowType == DataControlRowType.DataRow)
  {
    DataRowView data = args.Row.DataItem as DataRowView;

    if (data == null)
    {
      return;
    }

    // Set the appropriate CSS class to allow styling the table cell
    args.Row.Cells[1].CssClass = (bool)data.Row["Pass"] ? "pass" : "fail";
  }
}
```

TestRunner.aspx.cs

2. Update the code-behind class declaration for the embedded test runner web form to implement the `NUnit.Core.EventListener` interface as shown in the following code. This change enables the embedded test runner to collect information about each test as it runs. The `Run()` method of the test suite requires an instance of a class implementing the NUnit `EventListener` interface. The test runner implements this interface to collect information about the running tests.

```
public partial class TestRunner : System.Web.UI.Page, NUnit.Core.EventListener
```

www.it-ebooks.info

The contract for this interface requires many implementation methods, though you really only need to implement the `TestFinished()` method to store the test results:

```
public void TestFinished(NUnit.Core.TestResult result)
{
  // Put results into a data row
  DataRow dr = this.resultsTable.NewRow();
  dr["Test Name"] = result.Test.TestName;
  dr["Message"] = result.Message;

  // Add the stack trace if the test failed
  // to allow easier debugging of the test
  if (result.IsFailure)
  {
    dr["Message"] += result.StackTrace;
  }

  dr["Pass"] = result.IsSuccess;

  // Update statistical member variables
  if (result.IsFailure)
  {
    this.failedCount++;
  }

  if (result.Executed)
  {
    this.executedCount++;
  }

  // Add the result to the data table
  this.resultsTable.Rows.Add(dr);
}

public void RunFinished(Exception exception)
{
}

public void RunFinished(NUnit.Core.TestResult result)
{
}

public void RunStarted(string name, int testCount)
{
}

public void SuiteFinished(NUnit.Core.TestResult result)
{
}

public void SuiteStarted(NUnit.Core.TestName testName)
{
}

public void TestOutput(NUnit.Core.TestOutput testOutput)
```

www.it-ebooks.info

```
{
}

public void TestStarted(NUnit.Core.TestName testName)
{
}

public void UnhandledException(Exception exception)
{
}
```

TestRunner.aspx.cs

In the `TestFinished()` method, a call to the `NewRow()` method creates a new row for the result `DataTable`. The code then populates this row with information about the test that has just completed. The test runner updates the statistics for display at the end of the test run.



Remember to add tests to your project that contains the embedded test runner.

Deploying the Embedded Test Runner

Now that you have completed the test runner implementation, you must deploy it to a Sitecore test environment before you can use it. The implementation described previously makes such deployment quite straightforward: Compile the test runner and copy it to the document root of the Sitecore solution. You must deploy both the web form (the `.aspx` file) and the compiled assembly (the `.dll` file) to the Sitecore solution you use for testing. Later in this chapter, you will write NUnit tests into the same project so that the compiled assembly contains both the test runner and the tests to run.

The following explanation describes how to update the project such that after a successful build, Visual Studio automatically deploys the test runner to the Sitecore instance that you will use to run the tests:

1. To unload the project file, right-click the project in Solution Explorer and select Unload Project from the context menu.
2. To edit the project file, right-click the unloaded project in Solution Explorer and select Edit `[project name].csproj` from the context menu.
3. In the first `<PropertyGroup>` element that has no attributes, add an element to define the `SitecorePath` variable as shown in the following example, and set its value to the path to the document root of the Sitecore solution you use for testing on the local machine:

```
<SitecorePath>C:\inetpub\sitecore\SitecoreBook\Website</SitecorePath>
```

4. Find the commented section near the bottom of the file containing the `<Target>` element named `AfterBuild`. Uncomment that `<Target>` element and add code such as the following to it:

```
<CreateItem Include="**\*.aspx;bin\*.dll;bin\*.pdb">
   <Output ItemName="TestFiles" TaskParameter="Include" />
```

www.it-ebooks.info

```
</CreateItem>
<Copy SourceFiles="@(TestFiles)" SkipUnchangedFiles="true"
  DestinationFiles=
    "@(TestFiles->'$(SitecorePath)\%(RelativeDir)%(Filename)%(Extension)')"  />
```

The `<CreateItem>` task creates a list containing all the web forms (`.aspx` files) in all subdirectories and all assemblies (`.dll` files) and program databases (`.pdb` files, for debugging) in the `/bin` subdirectory of the project. This includes the web form containing the embedded test runner and the assembly containing its code. That assembly will also contain the tests that you write in this project. This `<CreateItem>` task also copies the NUnit assemblies on which the project depends, because when you compile your assembly, Visual Studio copies the referenced assemblies to the output path for the project (the `/bin` subdirectory in this case). The `<Copy>` task contains a transform in the `DestinationFiles` attribute that causes Visual Studio to place the files in the subdirectories in the target filesystem that correspond to their locations in the test project.

Now that you have created the embedded test runner and configured its deployment to the testing environment, you can write additional tests to run with the embedded test runner. If you invoke the embedded test runner without defining any tests, you may experience an error.

INSTANTIATING CONTROLS TO TEST

The testing technique described in this section enables you to instantiate individual controls to test. This approach overcomes some of the limitations of testing through HTTP and testing with a web browser described in previous sections, and addresses the issues of test scope. The two techniques previously explained each request an entire page to verify the output generated by a single component. The technique described in this section limits the scope of individual tests to the output generated by an individual control.

Implementing Tests that Instantiate Controls

You can use the technique of instantiating controls to test inside a Sitecore application by using the embedded test runner created in the previous section. The embedded test runner enables tests to call Sitecore APIs directly.

The basic process involved in the control instantiation technique is quite similar to testing through HTTP in that it collects, parses, and verifies the output of a control. The difference is that it does not collect the output of an entire page request.

The basic process of this technique is as follows:

1. Configure any required aspects of the Sitecore context.
2. Instantiate the control to test.
3. Parse the output of the control.
4. Verify output by asserting conditions.

The control to test likely expects a Sitecore context, including a context item and other static properties of the `Sitecore.Context` class. Your code can set the `Sitecore.Context.Database`,

www.it-ebooks.info

`Sitecore.Context.Item`, and other properties of the Sitecore context using the API methods shown in the following example:

```
Sitecore.Context.Database = Sitecore.Configuration.Factory.GetDatabase("web");
Sitecore.Context.Item = db.GetItem("/sitecore/content/home");
```

After setting the required Sitecore context properties, you can instantiate the control and invoke its `RenderAsText()` method to retrieve the output of a presentation control. The following example shows how to instantiate an eXtensible Stylesheet Language (XSL) rendering control, set the `Path` property of that control to specify which `.xslt` file to process, and retrieve the output that results from rendering that control under current processing conditions:

```
Sitecore.Web.UI.WebControls.XslFile xsl =
  new Sitecore.Web.UI.WebControls.XslFile();
control.Path = "/xsl/Child List.xslt";
string output = control.RenderAsText();
```

As when testing through HTTP, you can use the HAP to parse the output of the control as shown in the following example. Remember that the HAP attempts to represent the markup as valid XML.

```
HtmlAgilityPack.HtmlDocument document = new HtmlAgilityPack.HtmlDocument();
document.LoadHtml(output);
```

The remainder of the test follows the same process as testing through HTTP. The code creates a `System.Xml.XPath.XPathNavigator` object to traverse the output of the control and perform assertions against that output as shown in the following code:

```
XPathNavigator nav = document.CreateNavigator();
XPathNodeIterator elements = nav.Select("//li");
Assert.AreEqual(3, elements.Count);
elements.MoveNext();
Assert.AreEqual("Home", elements.Current.SelectSingleNode("a").Value);
elements.MoveNext();
Assert.AreEqual("About Us", elements.Current.SelectSingleNode("a").Value);
elements.MoveNext();
Assert.AreEqual("Partners", elements.Current.SelectSingleNode("a").Value);
```

The output processed in this case is that of the individual control; the test focuses on this limited scope rather than the markup of the entire page. This technique insulates tests from other controls that could affect the markup superstructure of the page.

Limitations of Testing by Instantiating Controls

Because this technique instantiates controls directly rather than using Sitecore to create controls, the behavior of the control may differ from that exhibited in a true Sitecore context. For an HTTP request, Sitecore instantiates the control and adds it to the page control tree during a specific ASP .NET page life cycle event. In that context, additional page events fire after ASP.NET creates the control, and the control may respond to those events. If the control under test relies on any of these events, it may not behave as expected when tested with this technique, leading to invalid results. If the control relies on events, then you can test either with HTTP or by using a web browser driver.

The lack of a full ASP.NET page lifecycle when you instantiate individual controls affects contextual information from Sitecore on which those controls may depend. If a Sitecore pipeline normally

www.it-ebooks.info

sets data used for a control under test, you need to set that data explicitly from code in order to test the control accurately.

Because this technique makes assertions against only the static output of the control, much like assertions made when testing through HTTP against the static output of a page, this technique does not allow you to test dynamic behaviors that affect the output of the control.

INVOKING THE SITECORE API DIRECTLY

The preceding testing techniques focus on testing presentation components and controls by parsing the markup they produce. Most controls generate markup that depends on calls into the Sitecore API to retrieve data or interact with the Sitecore context or other components.

A well-designed solution extracts the data collection logic from the controls themselves and places it into a series of utility classes and methods. It is easier to test such a solution than to test the markup produced by individual controls. This additional API layer also can help to centralize logic and make the solution easier to maintain, regardless of testing.

For example, suppose a control renders a list of an item's children, where each of those items contains a checkbox field named Include in List, and the control only processes those children that have that checkbox field selected. You can use code such as the following in the code behind of a sublayout to locate the items to display and bind them to the `System.Web.UI.WebControls.Repeater` control named `selectedChildren`:

```
private void Page_Load(object sender, EventArgs e)
{
  selectedChildren.DataSource = this.GetSelectedChildren();
}

private Sitecore.Data.Items.Item[] GetSelectedChildren()
{
  return Sitecore.Context.Item.Axes.SelectItems("*[@include in list = '1'"");
}
```

This version of the `GetSelectedChildren()` method has several issues. First, this implementation uses a private method of a sublayout, meaning that no other classes (including tests) can use that method. Second, the method uses the context item. Other components may need to access the selected children of an item other than the current context item. Finally, using the context item makes methods harder to test. When using the embedded test runner to invoke tests, the context item is likely to be `null`. It would be easier to test this method if it accepted the item under which to locate selected items.

Because other components may need to find the selected children of an item, you should move the `GetSelectedChildren()` method to an external class and make it `public`. Additionally, you may want to add additional signatures of the same method to allow developers to specify the name of the checkbox field and the item that contains that field. You would then update the existing implementation of the `GetSelectedChildren()` method to call one of the alternate signatures, passing `Sitecore.Context.Item` as the first argument to specify the item that contains the checkbox field and `include in list` as the value of the second argument to specify the name of the checkbox field

www.it-ebooks.info

to evaluate in the children of that item. The original method would then return the value returned by that nested method call to another signature of the same method name.

The following methods show a more testable version of the original method. These signatures include overloads that provide the capability to specify the item containing the children and the name of the checkbox field, which default to the context item and the original field name.

```
public static class ItemUtil
{
  public static Sitecore.Data.Items.Item[] GetSelectedChildren(
    Sitecore.Data.Items.Item parent,
    string fieldName)
  {
    string query = "*[@" + fieldName + " = '1']";
    return parent.Axes.SelectItems(query);
  }

  public static Sitecore.Data.Items.Item[] GetSelectedChildren(
    Sitecore.Data.Items.Item parent)
  {
    return ItemUtil.GetSelectedChildren(parent, "include in list");
  }

  public static Sitecore.Data.Items.Item[] GetSelectedChildren()
  {
    return ItemUtil.GetSelectedChildren(Sitecore.Context.Item);
  }
}
```

This data collection method is more testable and enables you to limit the scope of the test to include only the logic that locates the items to display, rather than parsing output to infer what the control did.

Now you can call the method that calls the Sitecore API directly inside a test as shown in the following example. This code depends on the `System.Linq` namespace:

```
[NUnit.Framework.Test]
public void SelectedChildItems()
{
  // retrieve the web database by name
  Sitecore.Data.Database web =
    Sitecore.Configuration.Factory.GetDatabase("web");

  // retrieve the /sitecore/content/home item from the web database
  Sitecore.Data.Items.Item home =
    web.GetItem("/sitecore/content/home");

  // retrieve the selected children of the home item
  Sitecore.Data.Items.Item[] selected = ItemUtil.GetSelectedChildren(home);

  // confirm the expected number of children
  NUnit.Framework.Assert.AreEqual(3, selected.Length);

  // retrieve the names of those children
  string[] names = (from child in selected
```

www.it-ebooks.info

```
        select child.Name).ToArray();

      // confirm the names of the children match expectations
      NUnit.Framework.Assert.Contains("child 1", names);
      NUnit.Framework.Assert.Contains("child 2", names);
      NUnit.Framework.Assert.Contains("child 3", names);
    }
```

This example first retrieves the item containing the selected children to test from the Web database, and then passes that item to the `GetSelectedChildren()` method that accepts only one argument. The first assertion ensures that the application returned the correct number of items. The subsequent code uses Language INtegrated Query (LINQ — see `http://bit.ly/woQJ2a`) to create a list containing the names of those items. The final assertions operate against that list to validate that the names of the selected items match expectations.

USING SITECORE APIS WITHOUT AN HTTP CONTEXT

Because Sitecore is a complete web application, to function properly, Sitecore APIs often depend on aspects of the solution configuration, which links together components of the system to create an application. You can use some, but not all, Sitecore APIs without a Sitecore configuration. Some aspects of Sitecore, such as the Sitecore context, depend on an HTTP context; you cannot test some APIs without a Sitecore context and hence an HTTP context.

Considering Calling Sitecore APIs Without an HTTP Context

To utilize this technique, the test project needs access to many Sitecore components. The following process describes how to configure everything you need to use this technique. You will need a `.zip` distributive (as opposed to the setup program) matching your version of the CMS, which you can download from the Sitecore Developer Network (`http://sdn.sitecore.net`). Because this technique does not run tests inside an HTTP context, you should perform the following process with the test project created as described for the two techniques presented previously in this chapter: testing through HTTP and testing using a web browser driver.

1. Ensure that the `.zip` archive matches the version of Sitecore used in the Sitecore solution to test.
2. Create a new subdirectory named `Sitecore` under the `/lib` subdirectory of the test project to hold the Sitecore binaries.
3. Expand all the files from the `/bin` subdirectory inside the Sitecore `.zip` archive to the `/lib/Sitecore` subdirectory you just created.
4. Create a new subdirectory named `/App_Config` under the test project to hold the Sitecore configuration files if that subdirectory does not already exist.
5. Expand all the files from the `/App_Config` subdirectory inside the Sitecore `.zip` archive to the `/App_Config` subdirectory in the test project.
6. To add an `/App.config` file to the test project, right-click the test project in Solution Explorer and select Add ➪ New Item from the context menu. The Add New Item dialog shown in Figure 8-9 appears.

www.it-ebooks.info

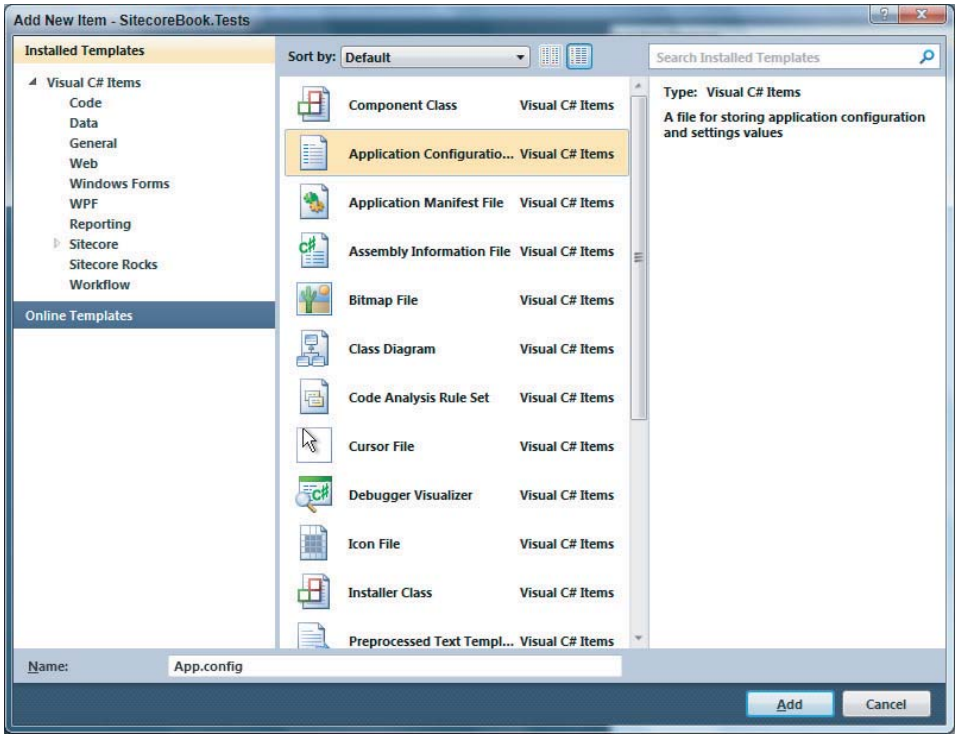


FIGURE 8-9

7. Select the General category and highlight the Application Configuration File item. Leave the content of the Name field as the default, `/App.config`, and click OK.
8. Expand the `/Website/web.config` file from the Sitecore `.zip` archive to a temporary location and open it in a text editor.
9. Copy the following sections from the temporary `web.config` file into the `/App.config` file in the test project:
   - The `/configuration/configSections/section` elements named `sitecore` and `log4net`
   - The `/configuration/connectionStrings` element
   - The `/configuration/appSettings` element
   - The `/configuration/sitecore` element
   - The `/configuration/log4net` element
10. Locate the `/configuration/sitecore/sc.variable` element named `dataFolder` and update the value to `data`.
11. Create a `/data` subdirectory under the test project and copy a valid Sitecore license file into that subdirectory.

www.it-ebooks.info

The test project now has all the files it needs to use the Sitecore API outside an HTTP context, but the files are not in the correct locations. You can use MSBuild (see `http://bit.ly/zhxx90`) to copy the files to the correct locations so the tests will work. The following process describes how to configure MSBuild on the test project to copy the files to the correct locations:

1. Right-click the project in the Solution Explorer and select Unload Project from the context menu. This step enables you to edit the MSBuild components of the test project file using a text editor in the IDE.
2. Right-click the unloaded test project and choose Edit `[project name].csproj` from the context menu. The test project will open in the editor.
3. Locate the `<Target>` element named `AfterBuild`. Uncomment that `<Target>` element.
4. Add the following XML to locate all the configuration files inside the `/App_Config` subdirectory that you created previously and copy those files to the output subdirectory while retaining the subdirectory structure:

```
<CreateItem Include="App_Config\**\*.*">
   <Output ItemName="ConfigIncludes" TaskParameter="Include" />
</CreateItem>
<Copy SourceFiles="@(ConfigIncludes)" SkipUnchangedFiles="true"
  DestinationFiles=
    "@(ConfigIncludes->'$(OutputPath)%(RelativeDir)%(Filename)%(Extension)')" />
```

   The `CreateItem` task locates the files indicated by the pattern in the `Include` attribute, which in this case locates all files inside the `App_Config` subdirectory (recursively). The `<Output>` element nested inside the `CreateItem` task defines the `ConfigIncludes` variable to contain the list of files, which the `<Copy>` element instructs Visual Studio to copy to the output subdirectory. The `DestinationFiles` attribute uses a transform to alter the destination files and subdirectories for the files.

5. Add a similar XML fragment to copy the Sitecore assemblies from the `/lib` subdirectory to the output subdirectory as shown here:

```
<CreateItem Include="lib\sitecore\*.*">
   <Output ItemName="SitecoreBinaries" TaskParameter="Include" />
</CreateItem>
<Copy SourceFiles="@(SitecoreBinaries)"
  DestinationFolder="$(OutputPath)" SkipUnchangedFiles="true" />
```

   This XML fragment follows the same structure as the previous fragment. The `Include` attribute of the `CreateItem` task locates all the files inside the `/lib/sitecore` subdirectory (not recursively). Because this is a flat list of files, you can use the simple `DestinationFolder` attribute of the `<Copy>` element to copy all the files specified by the `SourceFiles` attribute to that location.

6. Add a similar XML fragment to copy the `/data` subdirectory to the output subdirectory as shown here:

```
<CreateItem Include="data\*.*">
   <Output ItemName="DataFiles" TaskParameter="Include" />
</CreateItem>
<Copy SourceFiles="@(DataFiles)"
  DestinationFolder="$(OutputPath)\data" SkipUnchangedFiles="true" />
```

www.it-ebooks.info

This XML fragment is the same as the previous one, but locates a different set of files and stores them in a different variable.

The last step is to determine how to handle Web.config include files. Sitecore automatically processes Web.config include files located in the `/App_Config/Include` subdirectory. Other than requiring the `.config` extension, the name of the file is irrelevant to Sitecore, although file names should describe what type of information each configuration file contains.

The Web.config include files apply after Sitecore loads initial configuration from the actual `/web.config` file. The configuration patch utility tries to locate the patch files by using the `System.Web.HttpServerUtility.MapPath()` method. If the current application does not run in the context of a web server, which is the case for tests invoked using the method described in this section, then the `System.Web.HttpServerUtility.MapPath()` method returns the path passed to the method unmodified. When Sitecore tries to load the configuration patch files, it passes the path of the `/App_Config/Include` subdirectory to the `System.Web.HttpServerUtility.MapPath()` method. This call returns the value passed to it, in this case `/App_Config/Include`. The leading slash (`/`) character causes the patch utility to search for the subdirectory from the root of the disk where the application is running, such as `c:\App_Config\Include`.

There are two options to mitigate this issue:

- Do not use configuration patch files in the test environment and ensure that any configuration required appears within the `/App.config` file.
- Use MSBuild to copy the `/App_Config` subdirectory from the test project to the root of the current disk.

The first option is straightforward. The second option requires an additional copy task in the test project file. Locate the tasks in the `AfterBuild` target that copy the configuration to the output subdirectory and add an additional copy task to copy the configuration files to the root of the current disk:

```
<CreateItem Include="App_Config\**\*.*">
   <Output ItemName="ConfigIncludes" TaskParameter="Include" />
</CreateItem>
<Copy SourceFiles="@(ConfigIncludes)" SkipUnchangedFiles="true"
  DestinationFiles="@(ConfigIncludes->'\%(RelativeDir)%(Filename)%(Extension)')" />
```

Now the configuration patch files will work. You can reload the test project and run tests that call the Sitecore API directly.

This technique is somewhat more intensive than traditional unit testing. When calling the Sitecore APIs from the test code, Sitecore interrogates its configuration to determine which system components to create and connect, which includes the Sitecore databases. Tests that call Sitecore data APIs to read or write data actually read and write to and from the Sitecore database as defined by the configuration.

You can trim the configuration down by removing components not required in the testing context. For example, the test code may not need connections to all three default Sitecore databases; a single database connection may suffice for tests that read and write Sitecore items. Remember that the more the configuration of the test project differs from the configuration used in production, the greater the chance that the environment in which the tests run does not reflect that production

www.it-ebooks.info

environment. Although such tests may pass, the solution may not behave as expected in production. Cull the configuration of the test environment carefully.

Normally, a Sitecore test project exists only to contain tests for a Sitecore project. To ensure the tests run in an environment similar to production, use the Sitecore configuration actually used by the Sitecore project.

Limitations of Using Sitecore API Outside an HttpContext

Although you can access a number of Sitecore APIs using this technique, several APIs and classes depend heavily on an HTTP context and will only work properly when executed inside such a context.

When calling Sitecore APIs directly without an HTTP context, page life cycle events do not fire and the various Sitecore pipelines do not run. If the code under test uses Sitecore data in such cases, then the test code must create or initialize that data.

For example, the `SetGlobals` processor in the `initialize` pipeline initializes the `Sitecore.Globals` static object when the Sitecore application runs in an HTTP context. With no pipelines running, tests that depend on that object must initialize that component with an API call such as the following:

```
Sitecore.Globals.Load();
```

WORKING WITH TEST DATA

Good tests are resilient and repeatable. Small changes in the environment should not affect the outcome of a test, unless the specific intention of the test is to detect such conditions. The outcome of one test should not affect the outcome of any other tests. Tests should be isolated from each other to ensure predictable results.

You should run tests in a known environment. In Sitecore terms, the content tree makes up part of that environment. Many tests against Sitecore solutions involve methods and classes that access the content tree and interact with content in various ways, including reading, writing, and deleting content.

Writing automated tests requires more effort from developers than writing only functional code. If test results are not repeatable or seem to be random, developers may quickly lose heart with the practice, may not write sufficient tests, and may not maintain those tests.

To ensure that tests are repeatable, do not rely on the expectation that the content tree in the test environment will not change. For example, if you manually create an item for testing, and write your tests to depend on that item and the field values it contained when you created it, your tests may fail if another user moves, renames, deletes, updates, or otherwise affects that item.

To ensure a predictable and consistent state for the data used by your tests, create the test data at the commencement of the test, whether you create artificial data or copy actual data from elsewhere in the system. One test should not affect another test or leave the test environment in a state that differs from that at commencement of the test run. This means that each test should create and destroy any data that it requires. Creating and destroying test data prevents litter in the test environment and other potential issues.

www.it-ebooks.info



On one real-world project consisting of more than 300 tests, most tests accessed the content tree, working with the same five items as most of the other tests. Some tests modified these test items. The need to isolate tests became apparent when vast numbers of tests failed because an unrelated test failed and left one of the test items in a state other than what the subsequent tests expected. This made it much more difficult to identify the culprit and the actual cause of the failure than if the tests had been isolated from one another, created test data when required, and removed that data when complete.

NUnit provides several methods that automatically run at the commencement and conclusion of each individual test method, or at the commencement and conclusion of an entire test fixture class. You must attribute these types of methods in your text fixture classes to execute automatically in each of these cases so that NUnit can determine when to execute each such method.

Table 8-1 shows the attributes that you can apply for test data setup and destruction. These attributes exist in the `NUnit.Framework` namespace.

TABLE 8-1: Attributes Used for Test Data Setup and Destruction

ATTRIBUTE	PURPOSE
`[TestFixtureSetUp]`	The test runner calls these methods before any tests inside the test fixture class. In these methods, create test data used by all tests.
`[TestFixtureTearDown]`	The test runner calls these methods after completing all tests inside the test fixture class. In these methods, destroy all test data.
`[SetUp]`	The test runner calls these methods before each test in the test fixture class. In these methods, recreate common test data used by each individual test if any of the tests might alter the test data.
`[TearDown]`	The test runner calls these methods after each test in the test fixture class. In these methods, destroy any volatile data that any tests might alter.

You may encounter cases for which the lifetime of the test data should not exceed a single test. In those cases, the test that requires the data should create and destroy that data. To ensure destruction of the test data, even in the event of an error or exception in a test, wrap the test code logic in a `try...finally` block, and destroy the data in the `finally` block.

Location of Test Data

You can create temporary or permanent Sitecore items to use solely for testing purposes. Most items used for testing are temporary. You can create items for testing in any Sitecore database. While

www.it-ebooks.info

you should create items not intended for test in the Master database rather than a publishing target database so that publishing does not delete those items, you typically want to remove test items shortly after you create them. Therefore, you can create test data in a publishing target database such as the default Web database, in which case publishing can clean up the database if your testing infrastructure fails to delete those temporary items. If you use the technique described earlier for testing with an embedded test runner, the test runner operates in the context of the public website, which defaults to the Web database. There is a slight chance that publishing operations concurrent with testing could remove items in the publishing target database while tests run; if you create items in a publishing target database, avoid publishing while testing.

Creating Test Data through HTTP

The method used to create the test data may depend on the testing technique used for the test. If the test uses one of the techniques that run the test code outside an HTTP context, such as testing through HTTP or testing using a web browser driver, then you can use the standard Sitecore web service to create and destroy the test data.

The standard Sitecore web service resides at `/sitecore/shell/WebService/Service.asmx` on the Sitecore server. For example, assuming the home page of the Sitecore solution is `http://sitecorebook`, the address of the standard Sitecore web service on that instance is `http://sitecorebook/sitecore/shell/WebService/Service.asmx`.

To utilize the web service, you must add a service reference to the test project in Visual Studio. You can use the following procedure to add such a reference:

1. Right-click the test project in Solution Explorer and select Add Service Reference from the context menu. The Add Service Reference dialog shown in Figure 8-10 appears.

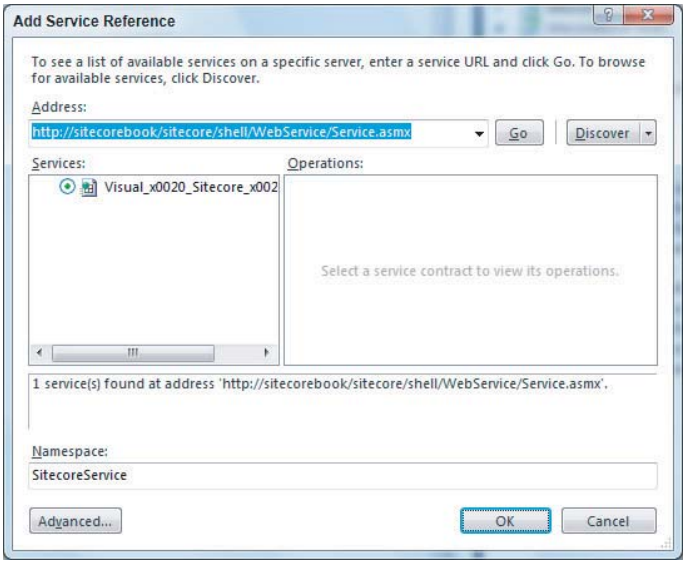


FIGURE 8-10

www.it-ebooks.info

2. Enter the URL for the Sitecore web service on the Sitecore instance on which you need to create data in the URL field.
3. Click Go. Visual Studio accesses the web service and retrieves the Web Services Description Language (WSDL) data for the service. The IDE uses the WSDL information to create a service proxy class that abstracts the web service methods available at the address you entered.
4. (Optional) Supply a namespace for the service in the Namespace field. Enter an appropriate namespace such as `SitecoreService`.
5. Click OK. The Add Service Reference dialog disappears and you return to Visual Studio.
6. Use the Sitecore standard web service to create test data as described in the remainder of this section.

Each call to a Sitecore web service method must include authentication details in the form of a username and password. You must populate an instance of the `Credentials` class in the namespace that contains the service proxy class, and supply that to the web service methods. The following code samples assume that the Sitecore service proxy class exists in the namespace `SitecoreService`. To create an instance of the `SitecoreService.Credentials` class, use code such as the following:

```
SitecoreService.Credentials credentials =
  new SitecoreService.Credentials();
credentials.UserName = @"sitecore\admin";
credentials.Password = "b";
```

You must provide a fully qualified username, which includes both the Sitecore security domain and the name of the user. When you create an item in Sitecore, you must specify four things:

- The database to contain the item
- The item under which to create the new item
- The data template to associate with the new item
- The name of the new item

Sitecore web methods expect IDs rather than instances of the `Sitecore.Data.Items.Item` class. The following code demonstrates how to create an item named `test item` under an existing item. This code depends on classes in the `System.Xml.Linq` namespace:

```
// Define the ID of the item under which to place the test item
string parentId = "{0DE95AE4-41AB-4D01-9EB0-67441B7C2450}";

// Define the ID of the template upon which base the new item
string templateId = "{76036F5E-CBCE-46D1-AF0A-4143F9B557AA}";

// Create the proxy instance
SitecoreService.VisualSitecoreServiceSoapClient client =
  new SitecoreService.VisualSitecoreServiceSoapClient();

// Create the new item
XElement response = client.AddFromTemplate(
  parentId,
```

www.it-ebooks.info

```
    templateId,
    "test item",
    "web",
    credentials);
```

The response from the call to the web method represents XML structured in the following format:

```
<sitecore>
   <status>ok</status>
   <data>
      <data>{953C7E7C-7019-40DD-9CB3-CC8F87935D5E}</data>
   </data>
</sitecore>
```

The response includes both a `<status>` element and a `<data>` element. You can use the `<status>` element to determine whether the web service call succeeded. The information returned in the `<data>` element and the structure of that data depends on the web method you call. In the case of the example code, the data returned by the `AddFromTemplate()` method contains the ID of the item created. You can use this ID in other method calls, such as to set field values in the new item.

You can set field values using the `Save()` method of the web service proxy class. This method accepts XML in the following format, which specifies the item to update, the fields to set, and the values for those fields:

```
<sitecore>
   <field itemid="{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}" language="en" version="1"
     fieldid="{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}">
      <value>The Field Value</value>
   </field>
</sitecore>
```

You can include multiple `<field>` elements to update multiple field values with a single call.

The following code demonstrates how to parse the ID returned from the `AddFromTemplate()` method call, construct XML to set the Title field of the newly created item, and then pass that XML to the `Save()` method to set the value of that field:

```
// retrieve the ID of the item created by the previous web service call
string itemid = response.Element("data").Value;

// Define the ID of the field to set
string fieldId = "{75577384-3C97-45DA-A847-81B00500E250}";

// Create the XML containing the field values to set
XDocument fieldXml = new XDocument(
   new XElement("sitecore",
      new XElement("field",
         new XAttribute("itemid", itemid),
         new XAttribute("language", "en"),
         new XAttribute("version", "1"),
         new XAttribute("fieldid", fieldId),
         new XElement("value", "the title")
      )
   )
);

// Save the field value
XElement fieldResponse = client.Save(fieldXml.ToString(), "web", credentials);
```

www.it-ebooks.info

At the completion of each test run, you should destroy test data created during that run. You can use the `Delete()` method of the class that represents the web service to delete a single item and all of its descendants. For example:

```
XElement response = client.Delete(id.ToString(), false, "web", credentials);
```

Pass the ID of the item to delete as the first argument to the method. The second argument is a Boolean value that indicates whether to move the item to the recycle bin or destroy it. The third argument specifies the database from which to remove the item. The final argument contains credentials used to authenticate before executing the command.

Creating Test Data through the API

If the testing technique used allows you to call the Sitecore API directly, then you can use the Sitecore API to create and destroy test data. Accessing the Sitecore API directly is easier than using web services because you can use all the classes in the Sitecore assemblies, you do not need to specify authentication credentials in code, and you do not have to parse an XML response to determine the outcome of each operation. The Sitecore API is more comprehensive than the web service API, providing many more options. You can use most of the Sitecore data APIs with or without an HTTP context.

The following code demonstrates how to use the Sitecore API to create items in the default publishing target database named Web by creating the `/sitecore/content/test` item using the `Sample/Sample Item` data template:

```
// retrieve the database named web
Sitecore.Data.Database web =
  Sitecore.Configuration.Factory.GetDatabase("web");

// retrieve the /sitecore/content item in the web database
Sitecore.Data.Items.Item parent =
  web.GetItem("/sitecore/content");

// grant this block of code rights to perform any operation
using (new Sitecore.SecurityModel.SecurityDisabler())
{
  // retrieve the sample item template
  // from the /sitecore/templates/sample folder
  Sitecore.Data.Items.TemplateItem template =
    web.GetTemplate("sample/sample item");

  // create the /sitecore/content/test item item
  // using the sample item template
  Sitecore.Data.Items.Item item =
    parent.Add("test item", template);
}
```

First, the application must retrieve the database in which to create the item. If you run this code inside an embedded test runner, which runs in the context of the published website, then the context database would be the Web database and you could update the code to use `Sitecore.Context.Database` instead of referencing a named database explicitly. Second, the application retrieves from

www.it-ebooks.info

that database the item under which to create the new test item. Third, the application retrieves the template with which to create that test item. Fourth, the application creates the test item based on that template as a child of the `/sitecore/content/home` item by calling to the `Add()` method of the `Sitecore.Data.Items.Item` class.

Note the use of the `Sitecore.SecurityModel.SecurityDisabler` class, which is required to ensure that the test code has sufficient access rights to create new items under the parent. If you run the example code with an HTTP context, and that context includes a Sitecore context that specifies a user with sufficient access rights, then you can move the instantiation of the `template` variable outside and above the `Sitecore.SecurityModel.SecurityDisabler`, and possibly remove the `Sitecore.SecurityModel.SecurityDisabler` entirely. If you run this code without an HTTP context, Sitecore cannot determine the context user, and assumes an anonymous user. In that case, without the `Sitecore.SecurityModel.SecurityDisabler`, this code would throw an exception because the anonymous user should not have access rights to perform the required operation. When using a `Sitecore.SecurityModel.SecurityDisabler`, Sitecore does not apply security for the context user, and the code does not fail due to insufficient access rights. For more information about the `Sitecore.SecurityModel.SecurityDisabler` class, see Chapter 4.

Setting the fields of an item using the Sitecore API is much easier than assigning field values using XML through a web service. The following code demonstrates how to set the value of the field named Title in the `Sitecore.Data.Items.Item` object represented by the `item` variable created previously:

```
using (new Sitecore.SecurityModel.SecurityDisabler())
{
   item.Editing.BeginEdit();
   item["title"] = "the title";
   item.Editing.EndEdit();
}
```

You can use the `Delete()` method of the `Sitecore.Data.Items.Item` class to destroy the test data, as shown in the following example. To ensure that you delete the correct item, store the ID of the item that you created previously (the following example assumes that ID exists in the variable named `id`).

```
Sitecore.Data.Database db =
  Sitecore.Configuration.Factory.GetDatabase("web");
Sitecore.Data.Items.Item item = db.GetItem(id);
item.Delete();
```

Alternatively, you could store a reference to the actual `Sitecore.Data.Items.Item` that you created, and then destroy it using the `Delete()` method of that object. Either method permanently deletes the item.

Creating Test Data from XML

Some tests can require large data sets and complex content tree structures. You can use the Sitecore API with a looping construct to create repetitive test data efficiently. For example, within a loop, you can call the `Add()` method of the `Sitecore.Data.Items.Item` class to add items and then set field values in those items using logic appropriate for your purposes. If the test data is complex, you may find it easier to create the test data using Sitecore user interfaces such as the Content Editor or

www.it-ebooks.info

the Page Editor, and then export those items as XML. You can then use that XML to create data required for tests.

There are many options for exporting the XML of an item. For example, you can use the `GetOuterXml()` method of the `Sitecore.Data.Items.Item` class to export items as XML, or you can use the free Sitecore Rocks (`http://sitecorerocks.net`) Visual Studio extension for Sitecore developers. You can also use third-party tools to export items as XML, but these techniques are beyond the scope of this book.

To retrieve the XML of a branch of items from the content tree using Sitecore Rocks, right-click the root item of that branch in Sitecore Explorer, and then select tools ➪ XML ➪ Copy. The resulting Get Item XML dialog allows you to copy the XML representation of the branch to the Windows Clipboard or save it to a file (see Figure 8-11).

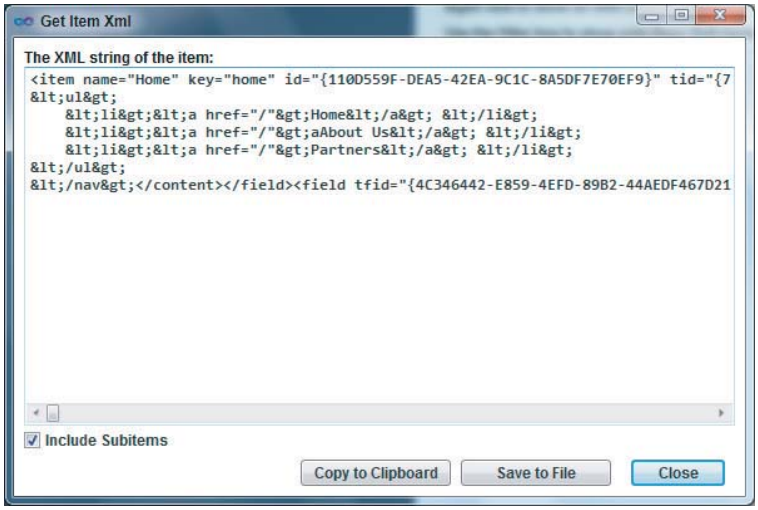


FIGURE 8-11

The XML returned by the `GetOuterXml()` method of the `Sitecore.Data.Items.Item` class or using the Sitecore Rocks technique described in the previous section provides an exact copy of the item as it exists in the content tree. Sitecore does not require all the attributes and field values in that XML when pasting it to create new items. You may need only the structure of the branch nested within that root branch, or particular field values from those items. At a minimum, Sitecore requires that pasted XML contain the components described in Table 8-2.

TABLE 8-2: Nodes Required in the item XML

NODE TYPE	NAME	PURPOSE
attribute	`@name`	Name of the item
attribute	`@tid`	ID of the data template associated with the item

continues

www.it-ebooks.info

TABLE 8-2 *(continued)*

NODE TYPE	NAME	PURPOSE
element	`version`	Each version of the item in each language
attribute	`version/@language`	Language of the version
attribute	`version/@version`	Number of the version
element	`version/fields`	Contains fields of the version
element	`version/fields/field`	Each field of the version
attribute	`version/fields/field/@tfid`	ID of the data template field
element	`version/fields/field/content`	Value of the data template field

For an item to exist in a given language, it must contain at least one field. The specific field that exists is not important; what is important is that the item contains at least one field. The following markup provides an example of the minimal amount of XML required to represent an item with a single version in the default English language:

```
<item name="min" tid="{8B1787D2-99A1-4688-ABA1-B60EFBA91BC3}">
   <version language="en" version="1">
      <fields>
         <field tfid="{001DD393-96C5-490B-924A-B0F25CD9EFD8}">
            <content>&lt;r /&gt;</content>
         </field>
      </fields>
   </version>
</item>
```

You can use the `Paste()` method of the `Sitecore.Data.Items.Item` class to create new items and entire branches of items in the content tree from an XML representation of an item that meets the requirements defined in the previous section. The following code demonstrates how to use the `Paste()` method to paste item XML to create new items. This example retrieves item XML from an external file:

```
string xml = File.ReadAllText("../../test data/tree.xml");
parent.Paste(xml, true, Sitecore.Data.Items.PasteMode.Merge);
Sitecore.Data.Items.Item testItem = parent.Axes.GetChild("test data");
```

The second argument to the `Paste()` method of the `parent` variable, which represents a `Sitecore.Data.Items.Item` object that you retrieved before invoking that method, controls whether Sitecore applies new IDs to items created from the XML instead of using the IDs that XML contains. Most often, you should apply new IDs to the new data, which avoids potential conflicts with existing items in the database that have the same IDs as the items in the XML. Such items may exist if you copied the XML from the same database into which you paste that XML (or a database that publishes to that database, as both could contain items with the same IDs). If field values in the imported XML reference other items in that XML by ID, you may wish to maintain the existing IDs when you paste the XML. If you do not instruct Sitecore to change the IDs, then you can create only a single instance of the test data in the content tree at any one time.

www.it-ebooks.info

The `Paste()` method of the `Sitecore.Data.Items.Item` class does not return an item. To retrieve an item from the test data created by pasting XML in this manner, use common Sitecore APIs. This example uses the `Axes` property of the `Sitecore.Data.Items.Item` object represented by the `parent` variable under which it pasted the data to select one of the imported items by name, where that name corresponds to the name of an item in the XML markup.

Once you have created the test items by pasting XML, you can treat them like any other items in a Sitecore database. Use the standard Sitecore APIs to destroy the test items as you would to destroy items created using the Sitecore API. For example, to delete the item created previously:

```
testItem.Delete();
```

TAKE HOME POINTS

This chapter detailed numerous techniques covering a wide range of Sitecore components and showed how you can effectively test each. If you are a seasoned developer with a test-driven mindset, then the idea of implementing testing for an entire Sitecore project may seem reasonable. If you are new to testing, the task of developing unit tests may seem quite daunting and tedious. Regardless of your experience with testing, any project can benefit from automated testing.

Inevitably, time and cost constrain the scope of any project, and hence the comprehensiveness of testing and the eventual quality of the solution. Investing excessive resources into testing could reduce the amount of development time available to implement functionality, visually appealing features, or other aspects of the solution from which you could derive more value regardless of code quality. If time is of the essence, then you need to take care when selecting which components to test and which aspects of those components to test automatically.

The tests that achieve the greatest value are those that cover functional aspects of the system under development. This is where defects that have the most impact typically manifest. These defects can be more difficult and time consuming to identify by testing throughout the entire site, which typically requires numerous different sets of input data to cover all cases. Developers often identify defects that present visually by interactively inspecting the website under development using common web browsers.

A well-written solution ensures the separation of the functional components that contain logic from presentation components. This allows you to design the functional components in a manner that simplifies testing, such as passing all parameters rather than using static properties of the Sitecore context. These components benefit most from automated testing.

This is not to say that the presentation components do not benefit from automated testing. Some presentation components may contain logic intrinsic to the component. You may find that the best way to test such aspects of the solution is to invoke such components individually or in the context of a page and test the resulting output. Your test architecture can use a combination of the techniques described in this chapter.

www.it-ebooks.info

www.it-ebooks.info

9

Managing Implementations

WHAT'S IN THIS CHAPTER?

- Managing projects
- Publishing content
- Understanding workflow
- Deploying changes
- Supporting multiple sites

After you configure your Sitecore solution, you want to get the most value from it. This chapter provides the information you need to manage Sitecore projects. After an overview of traditional project management, this chapter continues to cover the concepts of publishing and workflow, which separate work-in-progress content from that visible on your public sites and enforce a combined manual and automated process for transferring approved changes to your production content delivery environment. Once you understand content publishing, you can consider deployment, which involves the migration of developer assets, such as data templates and code, from one environment to another. You can also use Sitecore to manage multiple logical sites with a single Sitecore instance.

Sitecore is very true to the core principles of the ASP.NET application server, so you can easily leverage your existing expertise with the CMS platform. Once you truly understand Sitecore, the toolset should improve your ability to meet any web objectives that you can define, and reduce your time to market, as Sitecore eliminates some of the steps in the typical implementation process. You can use Sitecore to prototype solutions and then add features to the framework in iterations. Sitecore does not limit the output that you can produce in any way; you can implement any web solution in Sitecore. Sitecore also makes it easy to modify any component of your solution, such as by adding data to existing pages or by adding entire features to the solution.

www.it-ebooks.info

This chapter describes the Team Development for Sitecore (TDS — see http://bit.ly/xIx33R) product from Sitecore Technology Partner Hedgehog Development (http://hhogdev.com), which you can use to optimize and eliminate risk in your deployment procedures.

APPROACHING SITECORE PROJECTS

Many organizations that engage in web development projects employ a formal project methodology to their initiatives. Whether they use agile, waterfall, rapid application development (RAD), or some combination of techniques, methodology selection is a core component of the development process. At times, a content management system (CMS) can interfere with project methodologies and coding practices due to factors such as a framework closed to development and extension, coding restrictions, and proprietary scripting language limitations. The Sitecore CMS framework helps you avoid many of these pitfalls while actually enhancing the web development process. Working with Sitecore should not affect the project plan in a negative manner if you spend sufficient time on requirements analysis before choosing the underling platform and carefully chart your course. Unless directed otherwise by Sitecore, follow standard best practices for working with the underlying platform of ASP.NET. To achieve the greatest return in the least time, Sitecore customers should consider engaging an experienced Sitecore partner to assist with at least the initial phases of their first Sitecore projects. For more information about Sitecore partners, see Appendix A.

Modern web development projects often leverage agile project management methodologies such as the Scrum incremental implementation model that involves a number of iterative development sprints. For more information about Scrum, see http://bit.ly/xfaVK8. With Sitecore, you can use RAD methods to create a hybrid of project methodologies. This approach leverages RAD during the planning and requirements gathering phase of the project to validate assumptions quickly and to help identify any requirements missed during the initial analysis. The project can then move into an agile Scrum process for the remainder of the project. Throughout this entire life cycle, it is critical that the technology and platform components not hinder the developers or the application development process. The following sections of this chapter focus on how Sitecore enhances the developer's ability to prototype rapidly, especially early in the development process, and how Sitecore supports common coding techniques already established for object-oriented design processes.

Prototyping with Sitecore and RAD

Projects that use Sitecore do not interfere with the hybrid project management methodology described in the previous section. During the planning and requirements-gathering phases, such projects employ Sitecore to test assumptions within a development environment rapidly. This is one example of Sitecore providing a boost to developers that many other CMS platforms do not offer. Developers can quickly design basic data templates, common layouts and sublayouts, and other major components, which greatly increases productivity, enabling developers to focus on core requirements much more than would be possible using only .NET with a database server, or in comparison with various other CMS platforms. In addition, Sitecore provides a mechanism to elicit

www.it-ebooks.info

client feedback using the functional prototype, which aids in shortening the requirements-gathering phase and helps to validate the development team's initial ideas and plans. Furthermore, by employing this approach, project managers can segment the development effort into logical sprints, resulting in improved efficiency and concurrency. Upon completion of each sprint, the team can assess progress and the remaining sprints to determine whether to apply course correction based on lessons learned.

Prototyping with Traditional RAD

Typically, a web development team using standard Microsoft technologies (ASP.NET with Microsoft SQL Server and Visual Studio) uses the following procedure to build a simple data structure with storage persistence, along with a grayscale functional front end:

1. Create the database instance.
2. Create database tables that represent application objects and relationships.
3. Configure the application project in an integrated development environment (IDE) such as Visual Studio.
4. Create custom classes to instantiate, get, and set application objects.
5. Add custom code to add, modify, and delete objects in the database.
6. Create front-end shell HTML using ASP.NET master pages (see `http://bit.ly/wKeLF8`).
7. Create web forms to add, modify, and delete objects in the database.
8. Create front-end web controls and user controls to display objects from the database in accordance with project requirements.

Prototyping with Sitecore RAD

The Sitecore CMS framework enhances the traditional RAD process while reducing time to market. Sitecore automates many of the manual, labor-intensive steps that compose the traditional RAD process. Sitecore RAD prototype steps include the following:

1. Install Sitecore.
2. Create data templates to represent application objects and relationships.
3. Configure an application project in an IDE such as Visual Studio.
4. (Optional) Create custom classes to instantiate, get, and set application objects.
5. Create front-end shell HTML using layouts.
6. Create front-end presentation controls (renderings) to display objects from the database in accordance with project requirements.

www.it-ebooks.info

Sitecore can help you automate or eliminate the following steps in the traditional RAD process:

- Create database tables that represent application objects and relationships.
- Add custom code to add, modify, and delete objects in the database.
- Create web forms to add, modify, and delete objects.

The Sitecore CMS framework automates these tasks and enhances your ability to prototype in several areas. For example, suppose the development team implements a data template to represent products, and creates three renderings that perform the following functions:

- List all products.
- Display details about a product.
- Highlight a featured product.

If requirements emerge that indicate the need for an additional field to describe a new attribute available to all products, the developer can implement this change to the data template through a graphical user interface (GUI) without changing the database schema, and then update the corresponding presentation controls to use that field. In addition to requiring a new column in a database table, in a traditional web development process, such a change would require modification of the data entry forms used to manage product data, possibly changes to stored procedures in the database, and potentially even more development and testing effort (for example, to define a default value for the new field). Sitecore automatically and dynamically generates consistent, user-friendly data entry forms based on data template definitions.

To provide additional benefits for developers, the Sitecore CMS platform supports rapid front-end interface modifications by using placeholders in layouts and sublayouts. Placeholders enable the layout engine to bind any number of presentation controls to a page dynamically. Placeholders also enable users and developers to change the list of presentation components bound to each placeholder at any time using a browser-based GUI, rather than requiring updates to code, configuration, database records, or other technical solutions. Continuing with a real-world example using products, as illustrated in Figure 9-1, the development team can prototype the product detail page to include a product description and a matrix of product features.

Knowing that other pages have very similar designs, a developer would insert a placeholder into the layout or sublayout where the product features appear, and use layout details to bind a presentation control to that placeholder to render product features when needed, and bind other presentation controls to that placeholder for other types of pages. By using a placeholder, the developer can replace the presentation control for product features easily with an alternate version and see the impact on the entire site without changing a single line of code. The prototype shown in Figure 9-1 represents the exact structure as outlined by the information architecture team, but with sample content in place, which makes it clear to designers that there is not enough room to fit product reviews within the first fold of the web page. Assuming the presentation control implementations support flexible visual widths, a developer can rearrange presentation objects by binding them to alternate placeholders (such as a placeholder in the right column) through a

www.it-ebooks.info

browser-based user interface without updating any code or configuration files, resulting in output as depicted in Figure 9-2.

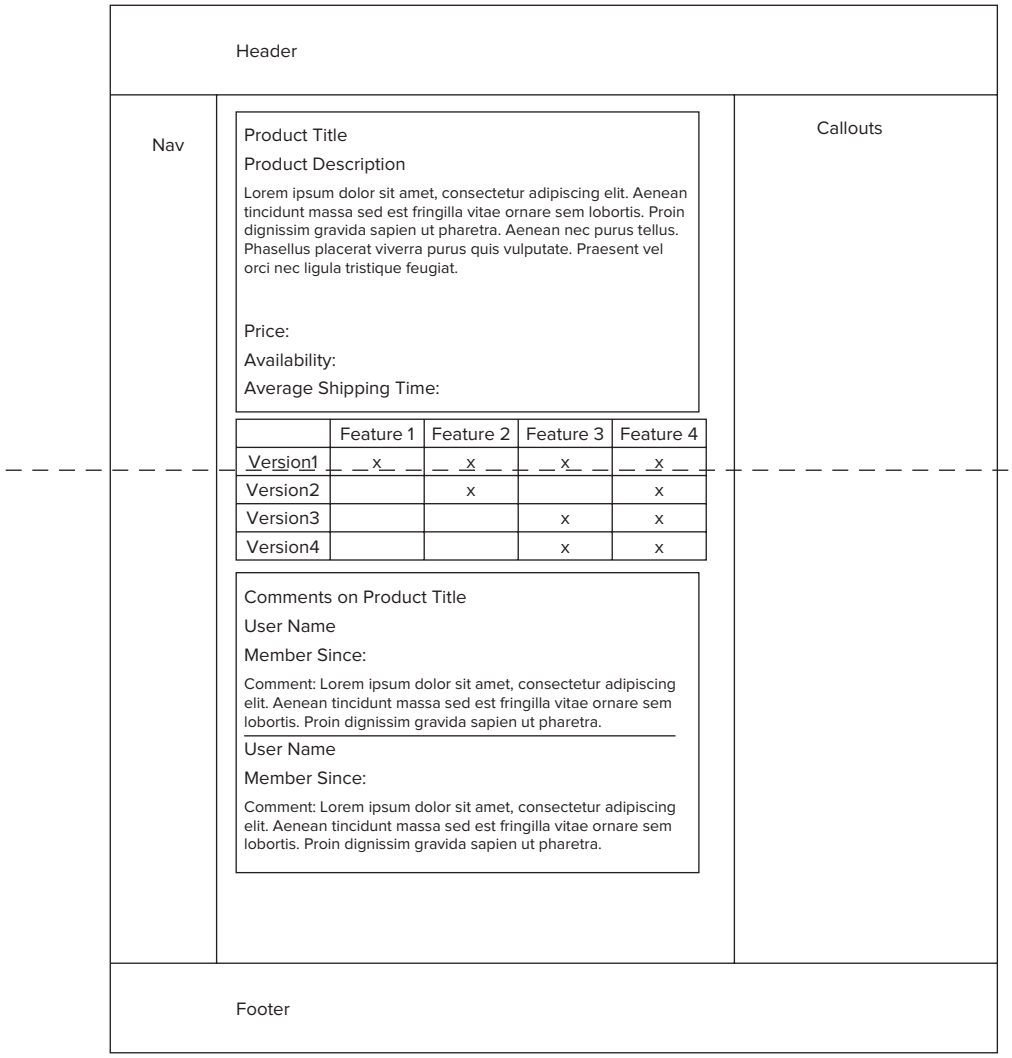


FIGURE 9-1

You can achieve these types of modifications using traditional ASP.NET web development, but that approach requires more effort and direct manipulation of code to see results. This manual effort can increase the amount of time required to implement such a modification, reduce resources available for testing the modification and implementing additional features, and even discourage essential design changes.

www.it-ebooks.info

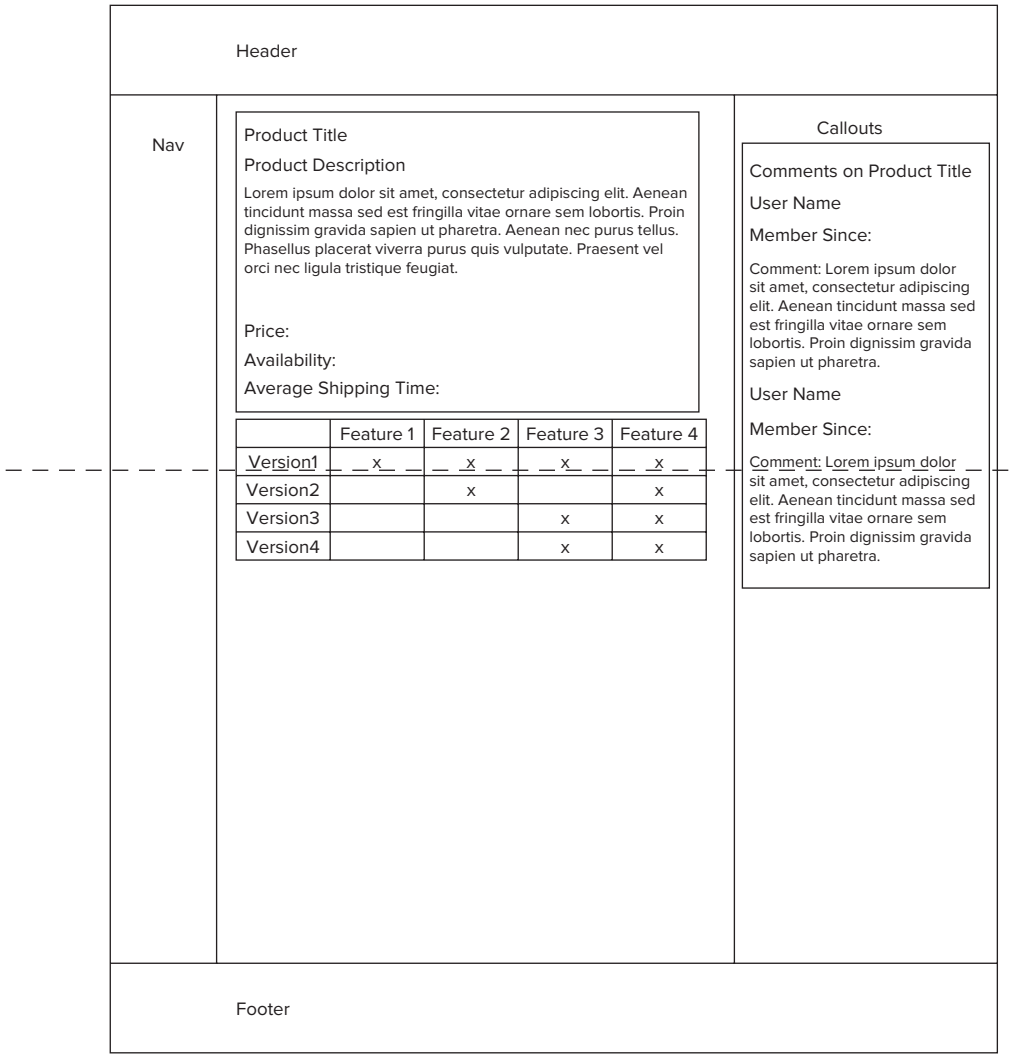


FIGURE 9-2

Facilitating Object-Oriented Coding Techniques

Upon completion of the requirements-gathering phase with rapid prototyping, you can leverage techniques from the agile Scrum methodology for the remainder of the development process. After implementing the basic data templates, common layouts, and superstructure sublayouts required for the project during a rapid prototyping phase, you could identify core features to develop in Scrum sprints. You can then check changes into a source code management system (SCMS) and deploy completed code using standard integration and deployment tools.

Most content management systems force developers to learn and manage a coding syntax unique to the product, which can negatively affect the development learning curve and the implementation

www.it-ebooks.info

process. The open nature of the Sitecore development framework, its use of standard ASP.NET components, and its adherence to the underlying principles of ASP.NET simplify the development process. The Sitecore framework enables you to follow a typical ASP.NET web application development approach while reducing effort and increasing speed. The development team follows the same steps to create custom objects, business logic, and front-end markup as they would follow without a CMS. This similarity in the process reduces the amount of learning required of developers before approaching the platform and project. Developers do not need to learn product-specific code syntaxes, work with incomplete application programming interfaces (APIs), or change the way they design and develop front-end markup in order to fit the requirements of the product framework. The following sections demonstrate how Sitecore seamlessly integrates itself with the existing development process.

Integrating Front-End Markup

Sitecore developers, including those working with Sitecore partners, can work directly with Sitecore clients and creative agencies that provide their own front-end markup, images, and other visual resources. These deliverables may consist of layered `.psd` (Adobe Photoshop Document) files or complete HTML or other markup, Cascading Style Sheets (CSS), JavaScript, and additional file types. The CMS must not dictate or shape the direction of the creative team or the development style of markup. The inherent segmentation of the Sitecore layout engine supports standards-driven markup using layouts and sublayouts, which provide a clear path for unobstructed front-end implementation and further integration.

As an example, consider the requirement to implement a navigation drop-down menu. You can develop and implement markup to meet this objective in several different ways. The Sitecore framework does not control whether you use unordered lists (HTML `<ul>` and `<li>` elements) with CSS and jQuery, a pure CSS approach (such as HTML `<div>` elements with supporting styles), or any other technique deemed appropriate for the project. Sitecore focuses on structuring and managing the content; you develop presentation components to render that content in any manner that you like. In fact, you can implement multiple presentation components for a shared set of content — for example, to use different markup elements for different types of devices (such as browsers and mobile phones). Sitecore leaves the markup implementation completely to the developer, which is the most flexible infrastructure possible. This single advantage supports broader organizational freedom, allowing interchangeable skill sets above the Sitecore foundation, reducing the level of effort required, eliminating bottlenecks, and delivering speed to market, organizational efficiency, and hence return on investment (ROI).

Integrating Custom Business Logic/Objects

Nearly every project requires integration with a third-party system or legacy business objects. The pluggable design of Sitecore and its adherence to standard ASP.NET coding techniques streamlines such integration efforts. Because the foundation of a Sitecore site is an ASP.NET web application, the options for integration and custom logic available to that platform apply to any Sitecore solution. Sitecore provides facilities such as events and pipelines to simplify system integrations.

You can use Sitecore APIs, any of the various techniques described in Chapter 7, and standard ASP.NET techniques such as web services to integrate custom logic and third-party systems into your Sitecore solutions. The coding required to use standard ASP.NET techniques does not vary due to the fact that you use Sitecore. For example, if a Sitecore solution requires integration with a legacy .NET application that uses custom classes, the Sitecore Visual Studio project can reference the assemblies containing those classes, and use those classes for native access to the methods contained therein.

www.it-ebooks.info

PUBLISHING WITH SITECORE

Sitecore sites offer a simple but flexible paradigm for publishing content from a content management environment accessed by CMS users to one or more content delivery environments accessed by visitors to the public (or internal, or extranet) sites. Content editors typically edit and preview content in a production content management environment, and then publish that content to a content delivery environment. Content becomes visible to visitors to the managed site in the content delivery environment only after publication. You must publish changes made by CMS users to items in the content management environment for those changes to appear in the content delivery environment. Specifically, Sitecore publishes items from the Master database to one or more publishing target databases.

You can publish an individual item, an item and its descendants, or an entire database. You can publish in one or more languages to one or more publishing target databases. When you publish, you can choose from incremental, smart, and complete republishing modes as described in the following section.

Follow these steps to publish an individual item or branch from the Content Editor:

1. Select the item from which to begin publishing.
2. Click the Publish tab, click Publish in the Publish group, and then click Publish Item. The Publish Item Wizard appears. If the Publish Item Wizard displays a welcome screen, click Next. The Settings page appears as shown in Figure 9-3.

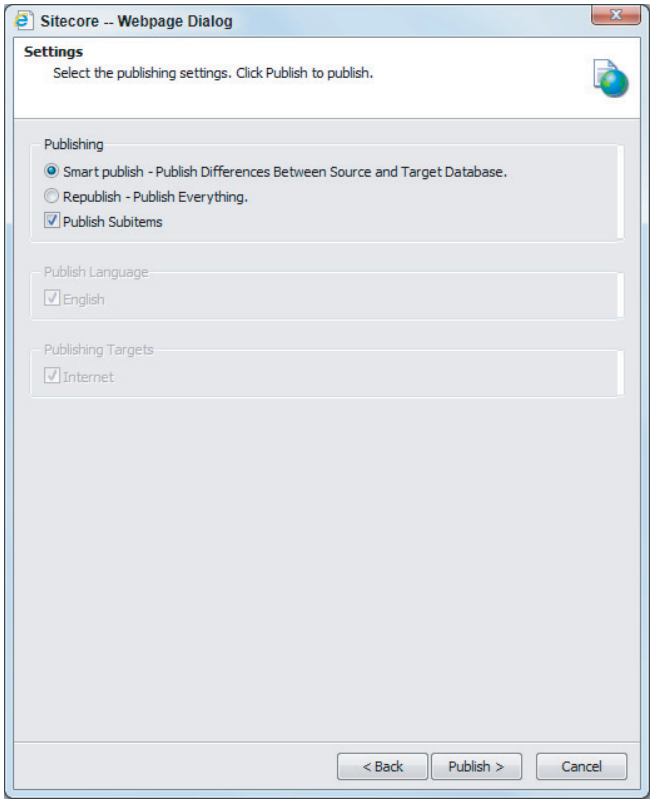


FIGURE 9-3

www.it-ebooks.info

3. Select publishing options based on your requirements and as described in the following sections. You can select either Smart or Republish mode, one or more languages to publish, and one or more publishing targets. Choose the Publish Subitems checkbox to publish an entire branch instead of only the selected item. Then click Publish. When Sitecore completes publishing, click Finish. The Publish Item Wizard disappears and you return to the Content Editor.

To publish the entire Master database from the Sitecore desktop, follow these steps:

1. Click the Sitecore button, and then click Publish Site. The Publish Wizard appears. If the Publish Wizard displays a welcome screen, click Next. The Settings page appears as shown in Figure 9-4.

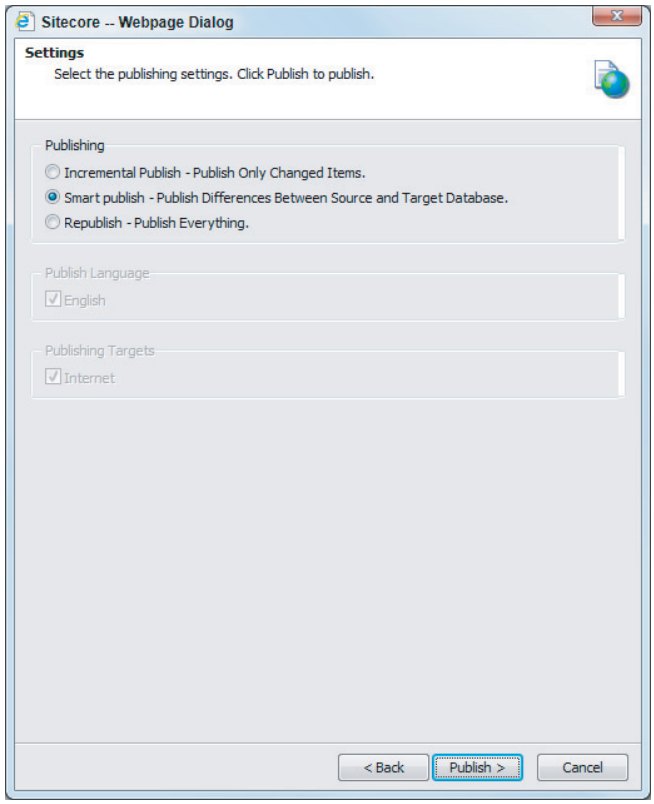


FIGURE 9-4

2. Select publishing options based on your requirements and as described in the following sections. You can select the Incremental, Smart, or Republish publishing mode, one or more languages to publish, and one or more publishing targets. Then click Publish. When Sitecore completes publishing, click Finish. The Publish Wizard disappears and you return to the Content Editor.

www.it-ebooks.info

For more information about publishing with Sitecore beyond that provided in the remainder of this chapter, see the publishing operations article at `http://bit.ly/r6Phna` on the Sitecore Developer Network (SDN). For ideas about controlling the publication of deleted items, see my blog post at `http://bit.ly/ol09C8`. You can also find several Sitecore shared source modules relevant to publishing at `http://bit.ly/vacX7d`.

Publishing Modes

Whether you publish an individual item or the entire Master database, you can select a mode of publishing for the operation. The following sections describe the publishing modes summarized in this list:

- **Republish** — Publishes all items in the database or only those items you select to publish, with no exceptions in either case
- **Incremental** — Publishes all items in the database or only those items you select to publish, in either case excluding items that Sitecore has not added to an internal list of items that may require publication
- **Smart** — Publishes all items in the database or only those items you select to publish, in either case excluding items for which the revision identifier in the source database matches that in the publishing target database



Regardless of which publishing mode you choose, Sitecore does not transfer items and versions of items to the target database if publishing restrictions (including workflow status) indicate that Sitecore should not publish the item or the version. Subsequent sections in this chapter describe publishing restrictions and workflow.

Republishing

When you choose to republish, Sitecore publishes each of the items that you indicate to publish (or all items) from the Master database to the publishing target database(s), with no exclusions. Republishing is the least efficient publishing mode. You can use republishing to force Sitecore to publish all items, such as when you bring a new publishing target database online or after you add a new language and corresponding content.

Incremental Publishing

When CMS users add and update items, Sitecore adds the IDs of those items to an internal list of items that may require publication. When you select incremental publishing mode, Sitecore publishes the items that you select to publish that also exist in this list, or all items in that list if you select to publish the entire database. Incremental publishing is typically more efficient than republishing or smart publishing.

Smart Publishing

Whenever a user updates an item, Sitecore updates its internal revision identifier for that item in that database to a unique value. When you select the smart publishing mode, Sitecore compares its

www.it-ebooks.info

internal revision identifiers for each version of each item that you select to publish in the Master database against the revision identifiers of the corresponding items in the publishing target database to determine which items to publish. Smart publishing is more efficient than republishing, but less efficient than incremental publishing. You can choose the smart publishing mode when incremental publishing does not transfer a change that you expect, without resorting to republishing.

Publishing Restrictions

You can specify publishing restrictions for each item and each version of each item in each language. Publishing restrictions control whether Sitecore can publish that item or version, as well as the dates during which Sitecore can publish the item or version. Additionally, you can specify publishing targets for an item, which prevent Sitecore from publishing that item to any other publishing target databases.



Regardless of publishing restrictions, Sitecore does not publish versions associated with non-final workflow states.

Follow these steps to set publishing restrictions for an item in the Content Editor:

1. Click the Publish tab, and then click Change in the Restrictions group. The Publishing Settings dialog appears as shown in Figure 9-5.

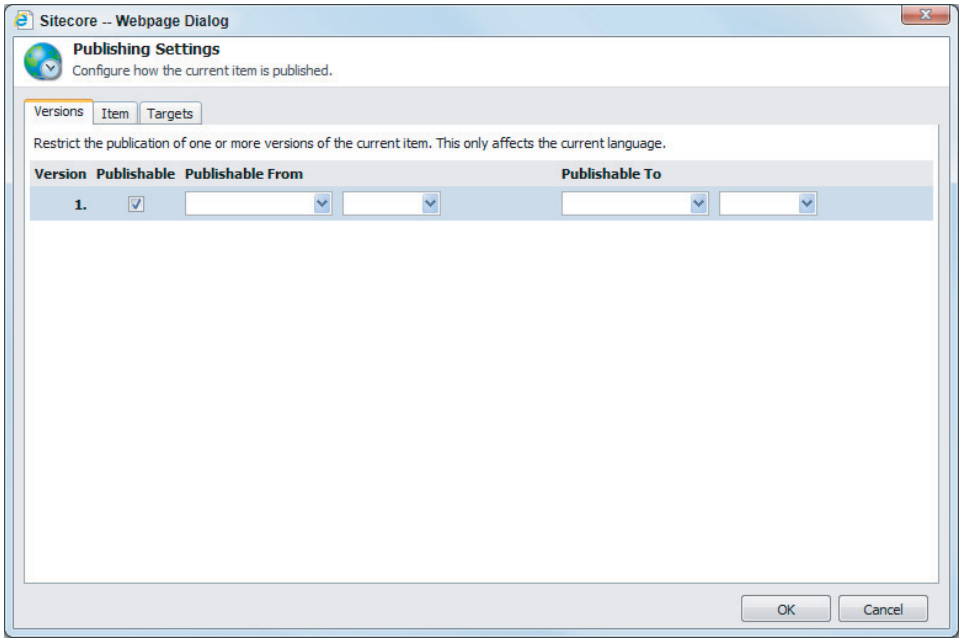


FIGURE 9-5

2. Click the Item or Versions tab to define publishing restrictions for the item or individual version according to your requirements.

www.it-ebooks.info

3. To prevent Sitecore from publishing the item or version, clear the Publishable checkbox. To restrict the dates during which Sitecore allows the item or version to appear in publishing target databases, enter values in the Publishable From and Publishable To fields. Then click OK. The Publishing Settings dialog disappears and you return to the Content Editor.



If Sitecore cannot publish an item, it cannot publish any descendants of that item.

Publishing Targets

Publishing target databases represent Sitecore content databases to which you can publish content from the Master database in the content management environment. Publishing target databases typically support content delivery environments. A publishing target database involves an actual database and a publishing target definition item. Whereas the Master database contains all versions in all languages of all items, including unapproved changes, publishing targets contain at most one version of each item in each language, and they do not contain unapproved content.

Most customers want to publish all items to all publishing targets, which is what Sitecore does by default. After you add a publishing target, publishing operations can publish all items to that new publishing target. Publishing restrictions can associate items with publishing targets, allowing Sitecore to publish those items only to the publishing targets selected.



It may not be intuitive, but if you do not select any publishing targets (as by default) on the Targets tab in the Publishing Settings dialog, Sitecore publishes the item to all publishing targets. As soon as you start selecting publishing targets for an item, you actually limit the publishing targets to which Sitecore can publish the item to those targets that you select. This is important because you do not need to update all your items after you add a publishing target; Sitecore automatically publishes every item to every publishing target if you select no targets. If you really do not want to publish an item to any publishing targets, clear the Publishable check box on the Item tab.

The name of the default publishing target is Internet. This publishing target specifies the Web database. This name could be confusing when using Sitecore for an intranet or extranet. You can rename this default publishing target, but you may need to update any code or configuration that references that publishing target by name.

You can configure additional publishing targets for various purposes, including the following:

- To support high-traffic volumes (each bank of content delivery instances accesses a separate publishing target database)
- To support geographic distribution and failover (publish to content delivery environments on different continents)
- To address redundancy and other scalability considerations

www.it-ebooks.info

- To allow publishing to a preproduction environment before publishing to the production content delivery environment (see the SDN forum thread at `http://bit.ly/o9c6Wo` for information about addressing this requirement)
- To host the databases for some of the managed sites in one publishing target database and other sites in another database, which can reduce publishing time and database storage requirements
- To complete publishing to a database and then switch the production configuration to use that new database

For more information about publishing target databases, see The Sitecore Content Reference manual (`http://bit.ly/qd6lUO`) and my blog post at `http://bit.ly/mSeQtc`.

Publishing to Preproduction Sitecore Environments

While the Sitecore publishing paradigm offers a robust user experience for content editors, developers often need additional options to deploy updates to data templates and other technical resources managed as Sitecore items. You may need to separate multiple different types of deployment assets. Apart from content deployment, which you can achieve with publishing alone, a typical "code" deployment consists of file assets and Sitecore items. For example, a sublayout consists of at least a web user control (`.ascx` file) and a sublayout definition item in a Sitecore database, and may involve a .NET assembly (`.dll` file). The functionality of the site depends on both types of assets becoming available simultaneously. If you deploy either of these assets without the other, the site might not function as you expect. For example, if you deploy the sublayout definition item without the file(s), content items that use that sublayout may generate errors.

Because the public Internet should not have access to the content management environment, you might be able to schedule downtime during which content editors should not update content while you migrate updates from development and test environments to that environment. After you have deployed both the code and the Sitecore items, you can further test the site internally in that environment.

Deploying updates to the content delivery environment presents a different challenge. Due to specific customer requirements, some published sites cannot tolerate any downtime. Such sites require more complex deployment procedures, especially considering the machine time required to deploy and publish new and updated items and files. Typically, file assets deploy more quickly than Sitecore items publish, which can open a window during which the content delivery environment does not contain all necessary components.

Many large Sitecore implementations share a single publishing target database with a bank of content delivery instances. This arrangement provides some flexibility for solutions that cannot tolerate downtime. You can copy or replicate the publishing target database to create a temporary publishing target database, and then configure some of the instances in the bank to use that temporary database instead of the original. Then you can remove the other content delivery instances from the bank, apply the deployment (or Sitecore upgrade), publish to the original publishing target database, and then test those instances. After you confirm functionality of the updated instances, you can add them back to the content delivery bank, remove the remaining servers from the bank, upgrade the remaining servers, and reconfigure them to access the original publishing target database.
After you complete the deployment to all instances, they all use the same publishing target database

www.it-ebooks.info

and service requests with the latest code; you can then remove the temporary publishing target database. You can follow a similar process even when multiple banks of content delivery instances depend on multiple publishing target databases.

Publishing Media

When you upload a file into the media library, Sitecore encodes its binary content and stores that data as a record in the Master database. Before you publish content items that reference a media item, you should publish that media item.



Excluding the File Drop Area field type, Sitecore does not automatically prevent a user from publishing content that references unpublished media.

There are multiple potential strategies for publishing media. You could require users to publish their media before publishing their content, in which case you might implement validation to alert the user about items that reference unpublished media (see my blog posts at `http://bit.ly/xVdkT0`). If you use workflow for content (which you should, even if users simply publish their own changes), then you can use a workflow action to publish related media (see my blog post at `http://bit.ly/ouU37j`). Otherwise, you can use event handlers or other features to publish media immediately after users upload new files or update existing media items. This last approach works whether you use workflow or not, but some organizations may not want to publish media immediately. Therefore, the most comprehensive solution may be to intercept content publishing to publish media referenced by that published content. In that case, see my blog post at `http://bit.ly/ouU37j`.



Before you publish a media item, you must first publish the folder that contains that media item to ensure that the parent item exists in the target database before the child.

Scheduling Publication

There are two aspects to scheduling publication: configuring publishing restrictions to control when Sitecore can publish the item (or versions of that item), and triggering a publishing operation after that time on that date.

Configuring publishing restriction does not cause Sitecore to publish or unpublish the item or version at the date and time specified; publishing restrictions merely control whether Sitecore can publish the item or version. You must initiate a publishing operation to publish the changes after reaching the date and time configured in publishing restrictions. You can publish manually; you can configure the `/configuration/sitecore/scheduling/agent` element in the Web.config file with a value of `Sitecore.Tasks.PublishAgent` for the `type` attribute to publish periodically; or you can implement a more advanced solution such as the Automated Publisher (`http://bit.ly/A6BHyX`) Sitecore Shared Source project. For information about scheduling processes with Sitecore, see Chapter 7.

www.it-ebooks.info

The publishItem Pipeline

Sitecore invokes the `publishItem` pipeline for each item that it publishes, including the publication of deleted items. You can implement a processor in the `publishItem` pipeline to intercept item publishing and apply custom logic, such as to add media referenced by the item to the list of items to publish. For an overview of the `publishItem` pipeline and an example processor, see my blog post at `http://bit.ly/n0Wsn6`.

WORKFLOW

A CMS workflow typically prevents content from reaching the production content delivery environment without passing through some combination of manual and automated process. Such a process can include any number of editorial and review phases, sometimes delineated by email notification to the parties involved, as well as automatic content validation and publishing.

Primary components of the Sitecore workflow system include the following:

- **Workflows** — Define overall processes
- **States** — Represent steps in workflow processes
- **Commands** — Allow users to transition content items from one state to another
- **Actions** — Automate functions in the workflow process



Keep your workflow implementation as simple as possible. Minimize the number of workflows, states, commands, and actions (especially actions that send email). Use access rights to allow the same workflow to work for multiple departments within the organization.

Before delving into complex workflow requirements, remember that one primary goal of the CMS is to facilitate changes to the site, not to hinder such changes or discourage CMS users from updating content. You cannot always design and implement a CMS workflow process to enforce every possible variant of the publishing process. I recommend that you use the fewest number of workflows possible, and make those workflows as simple as possible. At the same time, it is important that all changes except those made by developers and administrators flow through at least a simple workflow consisting of no less than two states, such as one "editorial" and one "published" state. The next level of workflow complication adds a review state between the editorial state and the published state. For more information about workflow, see The Sitecore Workflow Reference (`http://bit.ly/w5r81k`) and The Sitecore Workflow Cookbook (`http://bit.ly/AaVJoR`).

The Sitecore Workbox

The Workbox is a browser-based application for managing workflows in the Sitecore ASP.NET CMS. You can use the Workbox to approve, reject, edit, or otherwise manage content in the workflow system. To access the Workbox from the Sitecore desktop, click the Sitecore button, and then

www.it-ebooks.info

click Workbox. To access the Workbox from the Content Editor, select the Workbox tab at the bottom of the window. To access the Workbox from the Page Editor, click Workbox in the Edit group on the Ribbon. Figure 9-6 provides an example of the Workbox.

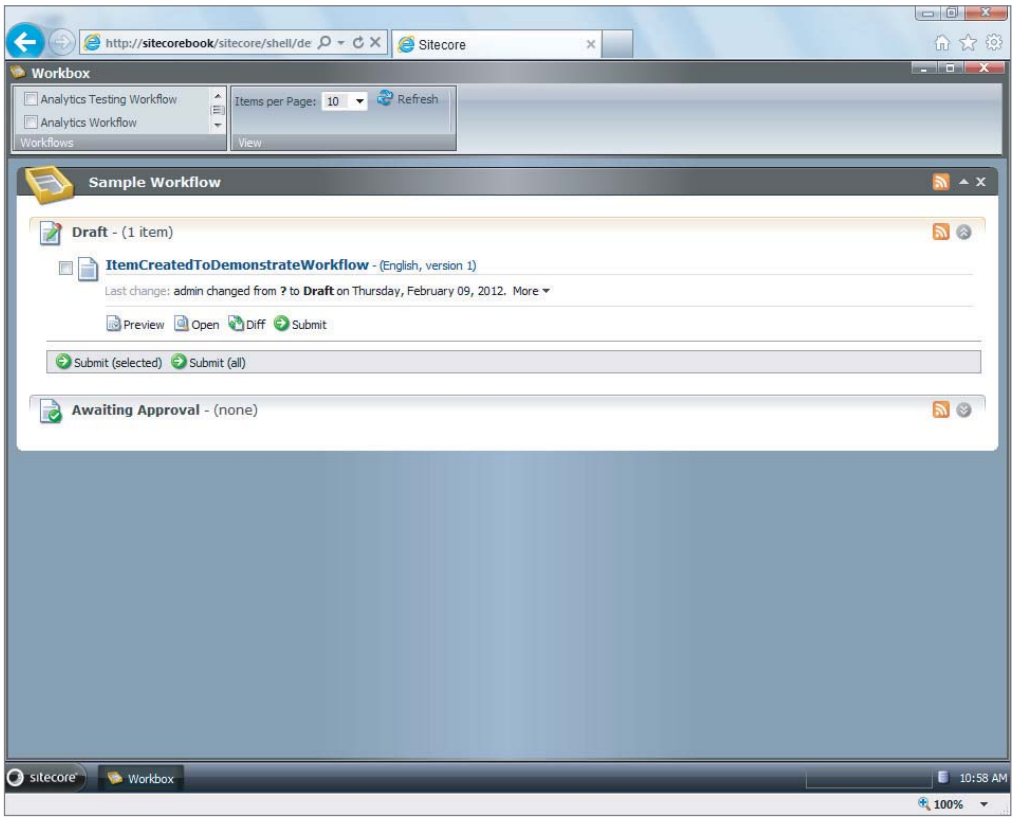


FIGURE 9-6

Locking and Workflow

Sitecore maintains locks to control which users can edit versions of items. Only one user can lock a version at any time. Excluding Sitecore administrators, no other users can edit a version while you own the lock for that version. When you create an item, Sitecore assigns you a lock to the initial version of that item in the current working language.



Locks do not prevent Sitecore from publishing version of items. Placing a version of an item in a workflow state that is not final prevents Sitecore from publishing that version.

www.it-ebooks.info

Workflows

Sitecore workflows define publishing processes for content items as a sequence of states. When a user creates or edits an item based on a data template, Sitecore initiates the workflow that you designate as the initial workflow in the standard values for that data template.

Sitecore follows a specific procedure when you create an item. If the data template for that item specifies an initial workflow, Sitecore puts the first version that you create in a language into the initial state of the workflow specified in the standard values of the data template associated with that item. When a user that is not a Sitecore administrator edits an item associated with a final workflow state, Sitecore creates a new version of that item in that language, and puts that version in the initial state of the initial workflow associated with that data template.



Specify the initial workflow in the standard values for each data template (Review ➪ Workflow ➪ Initial).

By default, each complete workflow cycle creates a new version. For example, when a CMS user creates an item, Sitecore creates version 1 of that item in the language selected by the user. After that version completes the workflow, when a CMS user that is not an administrator edits the item in that language again, Sitecore creates version 2, which must complete a workflow cycle before becoming a candidate for publication.

Workflow States

Workflows consist of some number of states representing different phases in the publishing process, such as editorial, review, and published states. A user's access to a content item depends on the union of their rights to the content item and its current workflow state. For example, even if you have write access to an item, if that item is in a workflow state to which you do not have write access, you do not have effective rights access to the item.

Initial Workflow State

You must specify an initial workflow state for each workflow. The initial workflow state specifies the default state in which to place items when they enter that workflow.



The initial workflow state of a workflow is not the same as the initial workflow for a data template. The initial workflow selected in the standard values for each data template specifies the workflow to activate for items based on that data template. The initial workflow state selected for each workflow specifies the state in that workflow in which to place items when they enter that workflow.

www.it-ebooks.info

Final Workflow States

You should specify at least one final workflow state in each workflow. Final workflow states serve at least two purposes:

- Sitecore can publish versions associated with final workflow states.
- When a user that is not a Sitecore administrator edits an item that is in a final workflow state, Sitecore creates a new version and puts that version in the initial state of the initial workflow specified in the standard values of the data template associated with the item.



Final workflow states have no correspondence to initial workflow states. A workflow can have any number of final workflow states, but designates only one initial workflow state.

Workflow Commands

Workflow commands provide a mechanism for users to move a version of an item from one workflow state to another. For example, from an editorial workflow state, an author could select a command such as Submit to move that content to a review state.

Workflow Actions

Workflow actions invoke .NET code when a version of an item reaches a specific workflow state or when a user invokes a workflow command. For example, you can associate an action with the final states of your workflow to publish the updated content automatically when it reaches that state, and you can associate an action with a rejection command to send an email message to the author.



You can associate workflow actions with both workflow states and workflow commands.

Sitecore provides default workflow actions that do the following:

- Publish the version that completed workflow, and optionally its descendants (associated with final workflow states)
- Send email notification (associated with workflow states and commands)
- Prevent workflow progress until the user resolves validation issues (associated with workflow commands)
- Move the version to a different workflow state when a user who is a member of a specific role saves the version

For an example of a custom workflow command intended to prevent reviewers from approving their own changes in cases where users serve as both editors and reviewers in a single workflow, see my blog post at `http://bit.ly/qFztYA`.

www.it-ebooks.info



By default, Sitecore does not apply a workflow when you delete an item or a version of an item. Any subsequent publishing operation can publish the deletion. For a discussion of using workflow to delete items, see my blog post at `http://bit.ly/ol09C8`.

You can assign the same initial workflow (the landing workflow) in the standard values of all data templates and use a workflow action to determine the actual workflow to apply to each item at runtime. For example, a content item might flow through different workflows depending on the site, branch, creator, category, or other properties. The initial state of the landing workflow contains a workflow action to determine which actual workflow to apply. Assigning this same landing workflow to all items can reduce administrative effort. More important, you can trigger specific workflows based on your specific requirements, meaning you can use landing workflows when you do not know in advance which workflow to apply to an item or all items based on a data template. You may want to add the landing workflow action to the initial state of all workflows, to move the item into a different workflow if conditions have changed since its last workflow assignment.



The landing workflow need not have a final state, but you should assign permissions and someone responsible for checking this workflow in the Workbox to ensure that your workflow action has not failed, leaving items in the initial state of the landing workflow.

In your landing workflow action, after you determine the ID of the definition item of the actual workflow to apply to the item, you can call a method based on the following example to set the workflow for the item:

```
protected void TransferToWorkflow(
  Sitecore.Data.Items.Item item,
  string workflowId)
{
 Sitecore.Workflows.IWorkflow workflow =
   Sitecore.Context.ContentDatabase.WorkflowProvider.GetWorkflow(workflowId);

 using(new Sitecore.Data.Items.EditContext(item))
 {
   item[Sitecore.FieldIDs.Workflow] = workflow.WorkflowID;
 }

 workflow.Start(item);
}
```



You could use the rules engine to determine which workflow to trigger. For an example that uses the rules engine in a custom context, see my blog post at `http://bit.ly/n2X3Pz`.

www.it-ebooks.info

MANAGING SITECORE DEPLOYMENTS

You will likely need to migrate three types of updates from development through test and production content management and content delivery environments:

- Content updates, which developers may initiate in a development environment or users may initiate in the production content management environment
- Code updates, which developers initiate in a development environment
- Sitecore CMS updates, released by Sitecore and deployed to all environments

In relation to deployment, the distinction between files on the filesystem and items in a Sitecore database is important. In addition to using definition items to contain metadata, Sitecore depends on files for specific features, including layouts, sublayouts, XSL renderings, and web controls. In many cases, you can use Sitecore publishing to deploy changes to content items from the production content management environment to the production content delivery environment. You should always use a separate release management process to migrate files from the development environment through one or more testing environments and eventually to production content management and content delivery environments. That release management process must account for items that developers change in development environments as opposed to items that CMS users change in the production content management environment, such as presentation component definition items and data template definitions, and you should typically synchronize the deployment of both types of assets. Sitecore automatically versions items in the database, but you should use an SCMS to version your files.

In some cases, you may have to choose between storing an asset in Sitecore or on the filesystem. For example, if a presentation component references an image, and CMS users never update that reference, you can put that image on the filesystem or in the Sitecore media library. If you want the capability to change the image, and potentially version that change using the CMS, without a code deployment, you can store the image in the media library. If you want a change to that image to be part of a code deployment, you can store that image on the filesystem. The same holds true for video and other types of resources.



In this context, I do not mean file-based storage of Sitecore media items, which is an option that I do not recommend (store your Sitecore media items in the database whenever possible). Rather, I mean that you can store such files on the filesystem without any corresponding media item in Sitecore, and reference those assets from code and markup as you would using a standalone ASP.NET solution rather than the Sitecore CMS.

Sitecore Item Serialization

You can use Sitecore serialization to serialize items and branches of items in a Sitecore database to `.item` files on a filesystem, and to import the data in such serialization files to a Sitecore database. You can check serialization files into an SCMS, which has advantages for sharing resources between developers as well as versioning and defining release branches. You can manipulate serialization

www.it-ebooks.info

files in ways that you cannot manipulate items, such as by using visual comparison tools. To access serialization features in the Content Editor, right-click a tab and then click Developer in the context menu to show the Developer tab. Then use the commands in the Serialization group to serialize items and entire branches to files and to update items from serialization files. For more information about serialization, see The Sitecore Serialization Guide (`http://bit.ly/pA9Cbm`).

Creating a Deployment

To create a deployment, first compile your Visual Studio project(s). Next, create a folder structure that contains all the new and updated files to deploy. This folder structure can include layout files (`.aspx`), sublayout files (`.ascx`), eXtensible Stylesheet Language Transformation files (`.xslt`), compiled assemblies (`.dll`), and other new and updated resources, including JavaScript (`.js`) and Cascading Style Sheet (`.css`) files to deploy to the document root of the Sitecore solution. Then create a Sitecore package to contain the Sitecore items changed since the previous deployment. A Sitecore package is a `.zip` file that contains meta data and it can contain Sitecore items from any database and files from the document root. For more information about Sitecore packages, see The Sitecore Package Designer Administrator's Guide (`http://bit.ly/zDvwnv`).

To access the Package Designer from the Sitecore desktop, click the Sitecore button, and then click Development Tools ➪ Package Designer. To create a package in Visual Studio using the free Sitecore Rocks extension for Sitecore developers (`http://sitecorerocks.net`), select the items to include in the package using the Sitecore Explorer, right-click one of those items, select Tools from the context menu that appears, and then click Create Package. Alternatively, with Sitecore Rocks, you can use the `package` keyword to create a package in Query Analyzer. You can include files in your package. Alternatively, when the time comes, you can simply copy the file assets manually or using a release management tool.

Depending on the complexity of the deployment, you can use two techniques to determine which items to include in the package:

- Package only the items that you have changed.
- Package all the data templates for the solution, ensuring you include all changes without having to pick each individual item.

Each of these alternatives presents potential problems. When you package items selectively, you may forget to include an item, which can cause delays or worse problems for the deployment. If you instead package all the templates, you may include work in progress not intended for the current deployment. You may also experience issues if your environment does not include the latest changes from all developers working on the project. It is very difficult, if not impossible, to merge changes from multiple developers by packaging all data templates comprising the solution. A more reliable approach requires a central repository for the current versions of all Sitecore items managed by all developers on the project, similar to a source control system for files.

Deploying to the QA Environment

To install the deployment in a quality assurance (QA) environment, run the Package Installation Wizard in that environment. To run the Package Installation Wizard from the Sitecore desktop, click the Sitecore button, click Development Tools from the menu that appears, and then click Installation

www.it-ebooks.info

Wizard. This task sounds simple and straightforward, but you may need multiple iterations due to the complexity of the packaging process, the number of packages created, and the order of dependencies controlling the package installation order.

Once you deploy the updates to QA, test in that environment to identify code that you need to update to address technical issues and anything that you may have forgotten to package, including both code and items.

Deploying Additional Changes

If you have to make some code changes in your local development environment to address issues that you identified by testing in the QA environment, you need to decide how to deploy those changes. You can follow the same process described previously, especially for the code assets. To reduce the size and number of update packages to install, you might want to create one or more packages containing only a subset of the items contained in the package(s) that you already deployed, especially if you updated a small number of items after that deployment.

If you create an entirely new package containing only the items changed since the initial deployment, you risk forgetting something important that changed from the original package(s). Careful tracking of all changes reduces this risk but adds overhead for developers. If you create a new package the deployment becomes more complex, as you need to deploy the packages to each environment in the proper order. This scenario gets even more complex when multiple developers contribute Sitecore item changes to the deployment.

Deploying to the Production Environments

After you deploy all changes to the QA environment and those changes pass QA testing, you can deploy those changes to the production environments (both content management and content delivery). Ideally, you can follow the same deployment steps that you used for the QA environment to deploy the same changes to production. Because production environments have different configurations than development, QA, and potentially other environments, you must take special care to ensure the preservation and deployment of a proper configuration.

Upgrading Sitecore

For current CMS releases, Sitecore provides update packages (`.update` files) that you can use to upgrade your Sitecore instances through a browser-based user interface. Unfortunately, Sitecore does not provide a user interface that allows you to create such update packages yourself. Always follow all instructions from Sitecore to install each update.



Update packages (`.update` files) differ from packages that you can create with Package Designer and Sitecore Rocks (`.zip` files). Use Package Designer and Sitecore Rocks for your changes; Sitecore uses update packages for its own system changes.

www.it-ebooks.info

Team Development for Sitecore

Sitecore Technology Partner Hedgehog Development (`http://hhogdev.com`) provides the optional Team Development for Sitecore (TDS — see `http://bit.ly/xIx33R`) software product, which enables developers to treat Sitecore items as code assets. Using TDS, you create items in one or more Sitecore environments and then synchronize those changes using a TDS project in Visual Studio. After a Sitecore item becomes part of a TDS project, it has all the features available for files in a source control system. Through integration with Sitecore Rocks, you can automatically include items in your TDS project. In other words, with Sitecore Rocks, TDS does not require any manual effort to include these items in your project.

When developers get the latest code from the source control system, they automatically get the latest Sitecore items within the TDS project. By simply deploying your solution, you get the most current Sitecore items in the target environment.

TDS removes many of the issues that exist with standard Sitecore serialization. Developers check in only those items that they have created or modified (not including all items provided by Sitecore itself), resulting in substantially fewer items in source control. This can reduce potential problems from inadvertently overwriting changes to items with old data from the source control system. For example, if a Sitecore update changes an item and that change did not make it into source control, overwriting that item from source control would lose the change.

TDS provides a complete audit trail on items, including deletion and renaming trails. You can easily use TDS to work on Visual Studio projects without storing your files within the document root on your development instance. TDS addresses path length limitations imposed on the filesystem by the operating system, and it automatically escapes characters in item names that are invalid on the filesystem.

MANAGING MULTIPLE SITES WITH SITECORE

You can manage multiple logical sites with a single instance of the Sitecore ASP.NET CMS. For each HTTP request, Sitecore sets the context site based on the first `/configuration/sitecore/sites/site` element in the Web.config file where the following attributes match the request:

- `hostName` — Matches the hostname part of the URL (remember that the N is uppercase)
- `virtualFolder` — Matches the path part of the URL (the subdirectory or virtual directory)
- `port` — Matches TCP/IP port (defaults is 80)

For more information about these attributes, read the comments above the `/configuration/sitecore/sites` element in the Web.config file. For example, the `hostName` attribute supports wildcards using the asterisk (`*`) characters, and multiple values separated by pipe (`|`) characters.

Additional attributes of each managed site (`/configuration/sitecore/sites/site` element in the Web.config file) include the following:

- `mode` — Enables or disables the site
- `name` — Name of the site, by which you may refer to it in code or other configuration elements

www.it-ebooks.info

- `physicalFolder` — The subdirectory containing files for the site
- `rootPath` — Path to an ancestor of the home item of the site (`/sitecore/content`)
- `startItem` — Path to the start item of the site, relative to `rootPath` (`/home`)
- `language` — Default content language for the site
- `database` — Default context database for the site
- `content` — Default content database for the site (the database to edit within CMS user interfaces)
- `contentLanguage` — Default content language for the site (the language to edit within CMS user interfaces)
- `contentStartItem` — Path to default home item for CMS users relative to the value of the `rootPath` attribute (`/home`)
- `masterDatabase` — Name of database to set as context database in Preview and Web Edit modes
- `device` — For this site, the layout engine sets the context device to the named device if layout details in the context item exist for that device
- `filterItems` — Controls whether the site shows the current version of an item without requiring publishing (for live mode configurations, which I recommend against)
- `filteredItemsCacheSize` — Size of the cache used to store items filtered according to the `filterItems` attribute
- `cacheHtml` — Enable or disable output caching for the site
- `htmlCacheSize` — Maximum size of output cache for the site
- `cacheMedia` — Enable or disable media caching for the site
- `mediaCachePath` — Subdirectory for caching media for the site
- `domain` — Default security domain for the site
- `requireLogin` — Require users to authenticate before accessing the site
- `loginPage` — Relative URL of login page for site



The `loginPage` *attribute must specify the URL of a physical file or an item that does not require Sitecore authentication.*

- `enableDebugger` — Enable or disable Sitecore debugging features, including tracing, profiling, and rendering information, and the debugging ribbon
- `enablePreview` — Enable or disable Preview for the site
- `enableWebEdit` — Enable or disable the Page Editor for the site

www.it-ebooks.info



WebEdit is a legacy name for what is now the Page Editor.

- `enableWorkflow` — Enable or disable workflow for the site (for live mode configurations)
- `enableAnalytics` — Enable or disable analytics (the Digital Marketing Suite) for the site
- `allowDebug` — Controls whether presentation components that inherit from `Sitecore.Web.UI.WebControl` collect profile and trace information, affecting whether the debugger renders information about those controls, and whether to show tracing and profiling commands in the debugger ribbon
- `browserTitle` — Value for HTML `<title>` element for pages on this site (used by the site that hosts the CMS user interfaces)
- `disableClientData` — Enable or disable storage required for CMS user interfaces for the site
- `disableXmlControls` — Enable or disable Sitecore user interface technology for the site
- `defaultDevice` — Default device for this site
- `inherits` — Name of another site from which to inherit all attributes not overridden by attributes of this site
- `targetHostName` — The hostname to use in URLs for the site (useful when the `hostName` attribute contains an asterisk (`*`) character or a pipe (`|`) character)

Some of these attributes, such as `content`, `contentLanguage`, and `contentStartItem`, are relevant only to the site named `shell` used by Sitecore user interfaces.



The distinction between the `device` and `defaultDevice` attributes of each managed site definition is subtle and not commonly understood, but it can be important when you need to apply different default devices for different managed sites. To determine the context device for an HTTP request, Sitecore first checks the query string parameter named `sc_device`. If the requested URL does not specify that query string parameter, Sitecore sets the context device to the device specified by the `device` attribute used in the site definition associated with the context site. If the `device` attribute does not exist for that site, Sitecore sets the context device to the first device definition item under the `/sitecore/layout/Devices` item in the context database containing field values that match the query string or user agent specified in the request. If no device definition item matches either of these values, Sitecore sets the context device to the first device definition item with the Checkbox field named Default selected, unless the `defaultDevice` attribute exists for the context site, in which case Sitecore sets the context device to the device named by that attribute.

www.it-ebooks.info

Each site has a start item that represents the home page for that site. To determine the start item for a site, Sitecore appends the value of the `startItem` attribute of the value of the `rootPath` attribute. For example, the default site named site defines `rootPath` as `/sitecore/content` and `startItem` as `/home`, leading to a start item at `/sitecore/content/home`.



Instead of using the Web.config file to define properties of the managed sites, you can use the Multiple Sites Manager (`http://bit.ly/xrDFyD`) Sitecore Shared Source project, which manages site definitions as definition items.

Sitecore defines ten managed sites by default:

- `shell` — Host Sitecore user interfaces including the desktop and the Content Editor (components under the `/sitecore/shell` subdirectory)
- `login` — Represents the Sitecore authentication system (components under the `/sitecore/login` subdirectory)
- `admin` — Represents Sitecore administrative pages (components under the `/sitecore/admin` subdirectory); some versions of the Sitecore installer may configure IIS to deny anonymous access to this subdirectory
- `service` — Hosts Sitecore error service pages (components under the `/sitecore/service` subdirectory)
- `modules_shell` — Represents Sitecore modules running within Sitecore user interfaces (components under the `/sitecore modules/shell` subdirectory)
- `modules_website` — Represents Sitecore modules running in the context of a published site (components under the`/sitecore modules/web` subdirectory)
- `website` — The default published site
- `scheduler` — Used for scheduled processes
- `system` — Used for system administration
- `publisher` — Used for publishing

HTTP requests in the content delivery environment, as well as CMS interfaces such as the Page Editor, Preview, and the browser-based debugger, activate the site named `website`. HTTP requests in Sitecore user interfaces such as the desktop and the Content Editor activate the site named `shell`.

You can add as many managed sites as you need, but you must use criteria such as those described previously to activate each site. You can use different managed sites for different devices, different languages, or for whatever purposes you need. The `/configuration/sitecore/sites/site` elements in the Web.config file function something like a fall-through mechanism: The `SiteResolver`

www.it-ebooks.info

processor in the `httpRequestBegin` pipeline sets the context site to the first `<site>` element with attributes that match the request.

Use the following steps to create a new site:

1. Create the home item and any descendant items, including any required data templates, presentation components, and any other required items.



Remember to publish the items for the new site from the Master database to the publishing target database(s), and consider that you can apply different languages, devices, and other properties to items in an existing site.

2. Copy the existing `/configuration/sitecore/sites/site` element in the Web.config file named `website` to create a new `<site>` element before the existing `<site>` named `website`.
3. Update attributes of the new `<site>` element, especially the attributes used to determine the context site (typically at least `hostName`, but potentially `virtualFolder` and `port` as well) and the `startItem` and possibly `rootPath` attributes.
4. (Optional) Set the `device` attribute of the new `<site>` element to the name of the device to associate a device with the site.
5. Duplicate one of the existing `<handler>` elements listed under the `/configuration/sitecore/events/event/handlers` element for each of the `/configuration/sitecore/events/event` elements named `publish:end` and `publish:end:remote` in the Web.config file, and update the values of those new `<event>` elements to the name of the new managed site.

If CMS users create internal links between managed sites, set the `alwaysIncludeServerUrl` attribute of the `/configuration/sitecore/linkManager/providers/add` element named `sitecore` in the Web.config file to `true` so that Sitecore includes the protocol and domain in links. You can set the `targetHostName` attribute of each `/configuration/sitecore/sites/site` element to the value to use as the hostname in URLs for that site.

For more information about managed sites with Sitecore, including a sample Web.config include file to define a site, see my blog posts at `http://bit.ly/qSAXyI`, `http://bit.ly/o9FcfG`, and `http://bit.ly/rl9HcG`.

TAKE HOME POINTS

You can use any project methodology you like to implement Sitecore solutions, and you can use a hybrid approach that combines the best features from multiple disciplines. You can use the Sitecore layout engine to construct any markup or other output format that you need.

www.it-ebooks.info

Before you roll your Sitecore solution into production, define a publishing process that should almost certainly involve workflow. You should separate that publishing process from the procedure that you use to deploy updates to the Sitecore software and your custom solution.

You can use multiple logical sites on individual Sitecore instances for a variety of purposes. For example, with multiple managed sites you rely on a single content tree, where the actual differences between the sites involve the hostnames used to access the server, the content language served, and the device-specific formats used to render that content.

www.it-ebooks.info

10

On Beyond CMS

WHAT'S IN THIS CHAPTER?

- Engaging your audience with the Digital Marketing System (DMS)
- Managing web forms with Web Forms for Marketers (WFFM)
- Organizing email campaigns with the Email Campaign Manager (ECM)
- Employing standalone Sitecore products
- Accessing the App Center
- Using optional modules

This chapter introduces Sitecore products beyond its core ASP.NET content management system (CMS). Sections in this chapter provide additional information about products in the following list.

- **Digital Marketing System (DMS)** — Leverage web statistics, engagement analytics, marketing automation, and much more
- **Sitecore Intranet Portal (SIP)** — Deploy intranet solutions with ease
- **Sitecore Foundry** — Allow multiple groups of users to manage numerous web properties with similar characteristics
- **Sitecore App Center (SAC)** — Integrate third-party online applications into your CMS
- **Adaptive Print Studio (APS)** — Reuse content and apply personalization in print campaigns
- **Calendar** — Publish calendars of events from your CMS
- **dtSearch** — Expose site search results using the dtSearch search engine
- **Sitecore E-Commerce Services (SES)** — Implement electronic commerce features

www.it-ebooks.info

- **SharePoint Integration Framework (SPIF)** — Integrate Microsoft SharePoint with your Sitecore solution
- **Active Directory (AD)** — Authenticate users against Microsoft Active Directory
- **Search Engine Optimization Toolkit (SEO)** — Analyze and optimize your web solutions to improve Internet search engine ranking
- **Sitecore Azure** — Host your Sitecore solution in the Microsoft Windows Azure Internet cloud
- **Web Forms for Marketers (WFFM)** — Simplify web form maintenance and form results analysis for non-technical CMS users

This is not a complete list of Sitecore products, but instead includes what I consider the most important components available for use with Sitecore CMS 6.5. Sitecore has discontinued some of its older products. For example, Sitecore 6.3 incorporated features from the Staging module, eliminating the need for that product. Sitecore continuously develops new software. For comprehensive information about current Sitecore offerings, see `http://www.sitecore.net`.

THE DIGITAL MARKETING SYSTEM

The Sitecore Digital Marketing System (DMS) pre-integrates functionality to automate customer engagement, monitor, manage web statistics, and support a wide variety of complex digital marketing activities as described in this section.



DMS replaces the Online Marketing Suite (OMS), which was available in Sitecore CMS 6.1 through 6.4 and adds a variety of improvements and new features not available in the OMS.

DMS extends CMS to support the Customer Engagement Platform (CEP). CEP is simply CMS with the DMS product installed. DMS provides a combination of features that enable marketing staff to track and analyze visitor behavior; launch, test, and optimize various marketing campaigns: and manage, measure, and optimize visitor engagement and experience. CEP provides a foundation for integration with products that provide additional capabilities including Email Campaign Manager (ECM) and Web Forms for Marketers (WFFM).

The Digital Marketing System supports the following major features:

- **Engagement analytics** — At the heart of DMS, engagement analytics tracks and collects all visitor engagement data across multiple channels and measures the quality of each interaction. Data written to the analytics database is available through Sitecore user interfaces as well as external reporting tools. DMS APIs make analytics data available to drive personalization in the content delivery environment.
- **Engagement management and automation** — You can think of engagement management and automation as workflow for visitors to the published websites and through other channels rather than content items in the CMS. With DMS, you can implement

www.it-ebooks.info

engagement plans that define a number of states, and manage visitors through those states. Time-based and event-based triggers move users from one state of an engagement plan to another, and you can transition users between states using a graphical user interface. In each engagement plan, you can specify system conditions to control which users can transition from each state to any of the possible subsequent states in that plan. You can associate any number of actions with conditions to automate specific functionality. By implementing actions that, for example, activate based on user clicks in marketing email messages sent to visitors in a state of a plan, you can coordinate digital engagement across marketing channels. Each channel can leverage insights gleaned from customer interactions through other channels, suggesting strategies to tailor the most relevant content to various audiences.

- **Testing and optimization** — These tools enable marketers to test the effectiveness of content in various web pages, email, and mobile interactions, enabling them to determine optimal content to achieve the greatest visitor engagement. Specifically, CMS users can apply rules to show and hide presentation components, change the data source of each such component, and replace entire visual components with alternative renderings.
- **Visitor profiling** — Profiling enables marketers to associate values with various content elements and functionality in order for the system to build knowledge about visitors based on their behavior on the site. As site visitors view items, Sitecore uses these profile values to determine profile scores for the visitor, which DMS can build from multiple visits and channels. CMS users can configure testing rules to drive personalization based on visitor profile characteristics, which are also available in reports.
- **Real-time personalization** — Personalization enables customization of each page view based on behavioral profiles, explicitly contributed profile values, logic, and other data such as geographic location.
- **Universal profile management** — Universal in this context refers to enabling marketers to use customer intelligence knowledge stored in third-party systems such as databases, customer relationship management systems (CRMs), and social networks to influence the user's unique experience on the site, through email interactions, and via other channels including print.
- **Campaign management** — Campaign management enables marketers to create, track, and manage various email, advertising, and search placement campaigns.
- **Dynamic segmentation** — Segmentation enables the placement of visitors groups dynamically based on profile attributes and user behavior. Segmentation enables marketers to target email campaigns and other features to different types of visitors.
- **Traffic type analysis** — DMS automatically classifies traffic sources into categories based on aspects of the visit such as whether the user accessed the site directly, through referral by an external site and the type of content on that site, or if a search engine brought the user to the site.
- **Visitor classification** — DMS determines visitor classification from a combination of automated logic and user settings, such as to identify search engine crawlers and other robots and existing customers. DMS users can override automatic visitor classification.

www.it-ebooks.info

In addition to these features, DMS uses the following terminology:

- **Visitor** — Visitors access any number of pages in the content delivery environment over any number of visits. Visitors differ from authenticated users of the published site and the anonymous user of the published site. Sitecore issues a cookie to all visitors regardless of whether they log in, enabling DMS to record information about each visitor over sequential page requests without requiring the user to authenticate.
- **Visit** — A visit consists of a number of page views by a visitor in a single browsing session. A visitor can return for any number of visits.
- **Page** — A page represents an HTTP request for a content item that contains layout details.
- **Page event** — Activity within a page can trigger any number of page events that DMS can report as associated with the page view.
- **GeoIP** — GeoIP records contain information about the owner of the Internet Protocol (IP) address associated with a visit, such as the geographic location associated with that IP address.



If you are familiar with Sitecore's Online Marketing Suite (OMS), a prior software release with a subset of the capabilities provided by DMS, the term visit *replaces the term* session*, and the term* visitor *replaces the term* global session.

In addition to the preceding terms and properties such as visitor profiles, DMS records information about many other attributes associated with each visit. Tracked data includes details about the marketing campaign(s), browser and user agent (the specific version of the browser), screen resolution, operating system, referring site, and the Internet search keywords that brought each visitor to the site, as well as the visitor classification, traffic type, multivariate test(s) executed, and test results associated with that visit.

You can configure DMS to use GeoIP lookup services such as MaxMind to expose information about the owner of an IP address block in web analytics reports. Such information can include the following attributes of the visit:

- Country
- City
- Postal code
- Latitude
- Longitude
- ISP (Internet Service Provider) name
- Organization name

For more information about DMS, see `http://bit.ly/u3Gmlq` and `http://bit.ly/wfb4LL`.

www.it-ebooks.info

Engagement Analytics

As mentioned previously, engagement analytics is at the heart of DMS. Engagement analytics consists of a variety of reports and visualization tools that provide marketers with information about website and campaign performance, visitor behavior and visit history, as well as a number of traffic-based reports containing traditional web analytics data. Engagement analytics differs from most other web analytics systems in two significant ways: it provides rich, detailed, visit-level statistics and a qualitative metric termed *engagement value*.

Traditional web analytics tools generally compress historical data into aggregate statistics such as total page visits for a given period of time. In contrast, engagement analytics stores detailed, visit-level data, such as a record of the individual pages viewed by each visitor. This extra information enables detailed analysis of visitor behavior patterns and other features such as predictive personalization.

Measuring engagement value provides a means to assess the quality of interaction with each visitor instead of merely collecting statistics such as the number of interactions that occur on the site. For example, a visitor reading product reviews might view only five pages. The specific focus of content on those pages may suggest that visitor is more likely to become a customer in the near term than another visitor who views more pages containing information about a greater number of products. A visitor who views information about a small number of expensive products might be a more valuable prospect to an organization than a visitor who views more pages about a greater number of inexpensive products. Investigating only the number of page views leads to a different perspective about the site visitor than measuring the quality and the quantity of those interactions. With engagement analytics, you can measure the effectiveness of a marketing campaign and various other traffic sources as well as the performance of various marketing channels.

DMS measures engagement value by applying a weighted score to the crucial conversion events that occur in visitor interactions. CMS users predefine engagement value points to activities such as a subscribing, signing up for a newsletter, registering for an event, downloading an asset, requesting additional information, and purchasing a product. DMS user interfaces indicate the accumulated value of the activities associated with visits, campaigns, and other system constructs. Engagement value provides a common key performance indicator (KPI) to help marketers make important decisions regarding how to focus their resources toward specific channels and campaigns. Sitecore offers a complete methodology for applying optimal engagement values to conversion events, as well as detailed workshops about best practices for approaching this activity. For more information about optimizing engagement value, see the Sitecore Engage website at `http://engage.sitecore.net`.

DMS provides the following reporting and visualization tools targeted at different types of CMS users:

- **Engagement reporting —** The simplest of all the engagement analytics tools, engagement reports contain information about visitor behavior, campaign summaries, and a variety of traditional web statistics. Engagement reporting targets content contributors and marketing users who need detailed information about website traffic for entire sites and individual pages.
- **Engagement intelligence —** Engagement Intelligence Analyzer enables web analysts and marketing managers to analyze current website engagement activity across various channels, campaigns, and interactions, a capability that traditionally required data analysts. This feature enables integration of site engagement information with data from external systems including

www.it-ebooks.info

Customer Relationship Management (CRM) and loyalty management applications, giving marketers a more complete view of the customer across all channels. Engagement intelligence dashboards provide traditional web statistics and marketing performance analytics.

- **Marketing Performance Quickview —** DMS users can access the Marketing Performance Quickview application (previously known as the Executive Insight dashboard) from the Sitecore desktop to evaluate the performance of traffic sources and campaigns using the engagement value KPI. The Marketing Performance Quickview application targets marketing leaders who want a quick overview of performance of various channels, campaigns, and traffic sources.

The Marketing Performance Quickview application provides access to the following types of dashboards:

- **Traffic —** Traffic dashboards offer a view into traffic sources, as well as how different traffic sources perform, such as branded and nonbranded (organic) search keywords, entry pages, site search keywords, assets consumed, and goal conversions. With this insight, marketers can optimize their search engine placement and other campaign efforts.
- **Campaigns —** Campaign dashboards utilize engagement value scores to determine and report the level of visitor engagement associated with each of the campaigns that brought visitors to the site. Marketers can use this information to obtain insight and optimize campaigns accordingly.

DMS provides the Engagement Intelligence Analyzer as an online analytical processing (OLAP) datamart that condenses data from the Sitecore CEP database(s) into a highly optimized reporting and business intelligence repository. The Engagement Intelligence Analyzer includes a comprehensive set of tools that allow marketers to access information in the datamart quickly and easily. These tools provide visualizations of common web statistics and marketing performance data, and enable ad hoc data analysis. To present data, the Engagement Intelligence Analyzer provides clear, dynamic tables and diagrams including the following:

- More than 120 interactive reports encompassing traditional web statistics along with engagement value
- More than 300 printable quick reports that you can export to a spreadsheet for further analysis
- Prototypes for users to create their own dashboards and reports quickly
- Built-in simple trending and mathematic capabilities, such as linear trends and standard deviations
- The capability to trigger email alerts after meeting or exceeding thresholds
- The capability to schedule reports and their distributions via email
- The ability to create slides for presentation dynamically

DMS users can filter and segment data on every dimension in the OLAP cube, which enables them to partition the data set in numerous ways and investigate anomalies, find causal relationships between various events, and identify latent trends and patterns in user behavior.

www.it-ebooks.info

Engagement Automation

Engagement automation enables marketers to nurture customers and prospects by automating interactions based on the visitor's implicit and explicit behavior, or with timed triggers (for example, if no interaction occurs within a given time period, send an email message to the visitor). You can think of engagement automation as functioning like workflow as described in Chapter 9, but involving site visitors rather than content items. Engagement automation enables marketers to create and execute cross-channel campaigns for site visitors, increasing marketing efficiency through automation and presenting the audience with relevant content based on their actions or their state in a given engagement plan.

For example, when users register for access to the website, you could automatically enter them into an engagement plan that triggers a welcome email. The system can then check whether the visitor has opened the email and logged in to the site, and can perform actions based on that response. If the visitor does not access the site within three days, the system can automatically send a reminder email. If they access the site, DMS can track the visitor's activity and begin to personalize content based on their behavior. For example, if a visitor focuses on a specific area of the site, the system can automatically show an offer inviting them to download a white paper about that subject, but requiring the visitor to complete a web form that asks for different information than they provided when they originally registered for access to the site. This type of approach enables the marketer to learn more about the visitor and continue to provide even more relevant content.

DMS users can configure engagement automation through the Marketing Center application in the Sitecore desktop. In this application, the user can design a process flow by dragging states and conditions onto the canvas, then connect those states, and add actions to execute when users move from one state to another. Engagement automation conditions and actions leverage the Rules Engine, which provides numerous predefined conditions to evaluate a visitor's profile and GeoIP information, and supporting personalization of the visitor's experience across all pages. DMS actions can integrate with other channels and systems, such as to update a record in an external customer relationship management (CRM) system or to send an email message. Like almost everything that Sitecore provides, if the default conditions and actions available in DMS do not meet your requirements exactly, you can implement your own.

DMS users can monitor the progress of automation plans by seeing how many visitors are in each stage of an engagement plan. They can determine the number of visitors in each state of the plan, as well as the success rate and value of a campaign, and then adapt the campaign and engagement plan based on that information. DMS users can also add visitors to, remove visitors from, and move visitors between the states in an engagement plan, and can simulate conditions and actions to test how various visitors will flow through different engagement states, conditions, and actions.

Testing and Optimization

Today's marketers need the ability to test various types of content, offers, features, and other types of interaction. This testing can mean the difference between guessing about the optimal message and ensuring that selection is based on concrete data. DMS provides testing and optimization capabilities aimed specifically at marketing users, providing the ability to test various pieces of content and functionality against each other, and to review precise feedback about the performance of each piece of content, without involving development staff.

www.it-ebooks.info

Sitecore offers two levels of testing: simple tests of individual elements on a page, known as A/B/N split testing, and the capability to test multiple variables on a page simultaneously, known as multivariate testing. Marketers can access both at the page level through an intuitive user interface.

DMS provides A/B/N tests that enable marketers to create a test on any page by selecting a component in the Page Editor and then choosing options from the Testing menu. This enables the user to create any number of variants of the selected component, providing the marketer with an option to select an existing data source item for that component, create a new data source item, or clone an existing item. Marketers have the option to test variations of content as well as presentation and styling of that content. After configuring the variations in the content management environment, the marketer elects when to deploy the test configuration to the content delivery environment.

After starting a test, DMS automatically calculates how long to run the test before reaching statistical significance, basing that duration on the number of variants under test and the volume of traffic the site receives. While the test runs, the marketer can see how each variant of a test scenario performs in terms of engagement value gained by each affected visitor, enabling the marketer to measure the complete impact of a variation of content or functionality. For example, the marketer can test two images and compare the results to determine whether visitors who see variation A of the content are more inclined to register for an offer than visitors who see variation B. Once a test reaches statistical significance, DMS automatically stops the test and prompts the marketer to select the winning scenario. At that point, the other variants deactivate and the system applies only the winning variant. Marketers can also stop tests manually at any time.

To configure a multivariate (MV) test, you simply configure more than one test for a single page. MV tests work exactly like A/B/N tests, but test a combination of variations across all of the tests on the page. For instance, with two tests, one with four variations and one with five, you would have a combined 20 permutations to test ($4 \times 5 = 20$). If you add a third test with five variations, you have 120 permutations to test ($4 \times 5 \times 6 = 120$). The testing and optimization feature automatically creates all the required combinations of tests, and reports on each permutation individually as well as in combination. Similar to A/B/N testing, MV tests run until they reach statistical significance, and then enable the marketer to pick the winning combination. A greater number of permutations requires more time to reach statistical significance.

Visitor Profiling

You can learn a great deal about people based on their behavior in the digital world just as you can in the physical world, for example by observing clothing, body language, and where they focus their attention. Sitecore DMS provides a rich profiling engine that enables marketers to understand visitor interests, tendencies, buying behavior, and other persona characteristics and audience segments to which the visitor belongs. You can use the profile data collected from visitor interactions in a variety of ways, such as to target content to each visitor in order to increase relevancy. Additionally, you can share this information with other systems, such as CRM solutions, so that it is immediately available to sales teams and call center agents. You can also use profile data to filter, segment, and otherwise evaluate campaign performance in reports and dashboards. These features allow marketers to make better decisions about how to manage their marketing programs.

www.it-ebooks.info

Sitecore provides a profiling system that enables nontechnical users to create and manage profile attributes, and then associate those attributes with content and functionality that visitors experience in various interactions across channels. As visitors consume content and functionality, their profile grows. DMS can aggregate profile data across many visits into a single data set. DMS users manage profiling in the Marketing Center, which they access from the Sitecore desktop.

To understand profiling, you must first understand some relevant terms:

- A *profile* defines a category used to specify the type of information by which to track visitor behavior, such as skill level, product preferences, and so forth.
- A *profile key* defines an individual attribute related to a specific profile. For example, if the profile tracks visitor behavior on a website for a car manufacturer site, a marketer might implement a driver type profile with profile keys for safety consciousness and exhibitionist tendencies.
- A *profile card* defines a preset collection of profile key values that DMS users can apply to content, simplifying the application of profiles during the content creation process. For example, a DMS user could apply the predefined "Interested in Family Cars" profile card to content items to set a high score for profile keys that indicate safety consciousness.
- A *profile card – persona* defines a profile card used to assign profile information to personas (fictional characters/archetypes designed to be similar to real people, with details describing their lives, habits, backgrounds, interests, and professions).
- *Content profiling* is the assignment of profile values to new or existing site content using either profile cards or specific values.
- A *real-time visitor profile* contains the unique profile of a visitor based on the content that visitor consumes on a site in real time.
- A *pattern* defines an easy way to recognize how similar one visitor is to other typical visitors. DMS matches patterns against each visitor's profile in real time, and automatically display relevant, personalized content based on similar behaviors and interests.

After you configure profiles, keys, and cards, you can begin profiling your content. Once you profile your content, DMS automatically builds a profile for each site visitor as they access content and site features. DMS automatically classifies visitors into predefined persona groups, or segments, but also expresses profile information visually to help marketers reclassify visitors if desired.

Real-Time Personalization

Personalization targets content and features at visitors to one or many pages using known information about that visitor. Real-time personalization reflects the capability to observe visitor behavior and attributes and dynamically adjusts content, presentation, and functionality based on that changing data. As discussed in the preceding section, you can learn a lot by observing visitors in your various channels. In addition to profiling data (equivalent to digital body language), DMS can react to visitors based on things they have informed the system about themselves intentionally or unintentionally. For instance, you can determine the approximate location of a visitor from the

www.it-ebooks.info

GeoIP record associated with their visit, you can investigate attributes of the phone or other device they use to browse your site, and you can account for the gender information provided when they registered for access. You can use the Page Editor to configure personalization for any element of any page.

Much like the testing and optimization capabilities, CMS users can personalize any component on any page without the need for development resources. To personalize a presentation component in the Page Editor, the CMS user selects that component on the page and then selects an option from the menu for that component Selecting that option exposes a dialog that allows the CMS user to create as many data source items for that component as needed, and to assign rules that dictate when the component should apply each data source. Users can personalize just the data source item, or vary the presentation used to render that content, or both.

Configuring personalization involves selecting rules within the Rules Editor and entering parameters for the conditions and actions in each rule, similar to configuring Microsoft Outlook's Out of Office Wizard. DMS provides dozens of predefined rule conditions and actions, and you can also develop custom implementations of each. Built-in rules can select content to render based on GeoIP properties, handle mobile device displays, target content to various user types or personas, and even vary content when users match certain behavioral patterns. In addition to configuring personalization at the page level, you can also configure personalization for all pages on the site, for each section, and for each page type using personalization theming.

Universal Profile Management

In modern organizations, customer information resides in many places. Customer databases such as loyalty management systems and customer relationship management (CRM) systems are commonplace. Social networking sites such as Facebook, LinkedIn, and others expose a tremendous amount of information, with new sites of these type appearing constantly. The capability to tap into those data repositories, use that information, and even feed information back to those repositories is becoming critically important in creating a consistent and cohesive user experience for your visitors across multiple web properties.

The universal profile management technology utilizes a combination of Sitecore's security architecture, profile providers, and CRM and social integration features to enable bidirectional data sharing with these third-party customer databases. This enables you to use any information located in these external repositories as if it were native Sitecore data, such as to drive personalized experiences, to feed customer experience data back to those external systems, and to create follow-up tasks and more.

Campaign Management

Marketing campaigns are the lifeblood of marketing. DMS supports many types of campaigns, including online advertising campaigns, offline campaigns, and email campaigns. DMS natively supports tracking campaigns. Marketers create campaigns in the Marketing Center, which they can access from the Sitecore desktop. As described in the following paragraph, you can trigger campaigns in several ways.

www.it-ebooks.info

For online advertising campaigns, embedding the unique ID of a campaign in links back to the website triggers the campaign and associates the visit and all goal conversions and engagement value derived from that visit with that campaign. For offline campaigns and other situations that do not involve direct links, you can implement landing pages to trigger the campaign. When a visitor accesses a page, that visitor's visit becomes associated with that campaign, including all conversion goals and engagement value for that visit.

To support email campaigns, Sitecore DMS provides the Email Campaign Manager (ECM) application described further in this chapter, enabling DMS users to conduct bulk and individualized email campaigns integrated with the content management system. Marketers can personalize email content based on a visitor's known interests, and you can personalize content on the website based on the visitor's actions in the email, such as which link they click. Because CMS users access the Page Editor to create content for email, including images and other media, they work in a consistent, familiar, user-friendly environment. Once composed, ECM automatically encodes links within an email to trigger campaigns if a visitor clicks. The links in an email not only trigger the campaign, but also support individualization for each visitor, so that you know when each visitor clicks each link for any campaign.

The engagement automation engine can access ECM to trigger email distribution based on conditions defined in an engagement plan. CMS users can monitor email campaign results in the Marketing Performance Quickview, which indicates how a campaign performs based on the engagement associated with it. CMS users can also compare the results of email campaigns with each other and with other campaigns and traffic sources.



For more information about Email Campaign Manager, see `http://bit.ly/ur50cW` *and* `http://bit.ly/tUQ3ne`.

Dynamic Segmentation

A fundamental component of marketing campaign management involves managing lists of members in the audience, or audience segments. Marketers use these lists to target offers to specific visitors to the website or recipients of email, and to follow up with those visitors. Traditionally, marketing systems created lists and segments based on static attributes of profile data such as mailing address, gender, and age. While marketers can create and administer visitor lists manually, DMS can also define segments dynamically and reclassify visitors based on their behavior. As such, lists and segments support dynamic inclusion or exclusion of members based on their activities and profile attributes.

Marketers create and manage lists and segments within the Marketing Center, which you can access from within the Sitecore desktop. The Dynamic Segmentation tool allows marketers to configure lists in a manner similar to using personalization rules. Audience segmenting rules define parameters that determine which visitors to include in each segment. After creation, you can use dynamic segments for purposes such as directing email newsletters, marketing campaigns using engagement automation, and targeting web page content to visitors in specific segments.

www.it-ebooks.info

Web Forms for Marketers

The Web Forms for Marketers (WFFM) module provides browser-based user interfaces to help non-technical CMS users create and manage web forms associated with Sitecore items. Each form consists of a number of fields with various properties. Forms can also include buttons that can invoke any number of predefined actions, such as to create an item from the field values and save them to a database, or send an email message. You can also develop custom actions. WFFM forms automatically support internationalization and localization. WFFM includes a Form Data Viewer for viewing information from forms submitted to the database. Using the viewer, you can export that data to a spreadsheet for analysis. Sitecore implements WFFM forms as `.ascx` files that work like any other presentation controls. You can export WFFM forms as sublayouts that you can update in Visual Studio.



For more information about the Web Forms for Marketers (WFFM) module, see `http://bit.ly/tOhbic` *and the documentation at* `http://bit.ly/vqn9ql` *appropriate for your version of WFFM.*

Email Campaign Manager

Sitecore's Email Campaign Manager (ECM) provides a solution that allows marketers to create, manage, and deliver email messages to targeted lists of individuals. ECM allows marketers to track the rates by which recipients open each message and click contained links, and to monitor subsequent site visitor activity. You can configure ECM to work with Sitecore's enterprise-grade Message Transfer Agent (MTA) and other third-party MTAs to maximize the delivery rate of emails. MTAs deliver email messages, maintain relationships with Internet Service Providers (ISPs) on your behalf, assist with whitelisting, monitor and resolve blacklisting issues, implement real-time feedback loops (RTFLs) to prevent spam reports, and provide additional services to maximize delivery without adversely affecting your reputation.

In addition to MTA services, you can use email services through the Sitecore App Center described in this chapter, which integrates with third-party providers in Internet clouds. For example, you can use a service that analyzes how major spam filters are likely treat a message under composition, to preview a message as it appears in different email clients, and to view email campaign landing pages in various browsers.

You can use ECM to send email campaigns in bulk and to deliver iterative marketing campaigns in conjunction with engagement automation. ECM users can create and assemble emails using the same Page Editor they use to build web pages. Additionally, ECM users can configure personalization rules for targeted content delivery as well as A/B tests in the email subject line and content body. ECM is ideal for email newsletter distribution, ad hoc email distribution, and marketing campaigns to grow your business.



For more information about ECM, see `http://bit.ly/tUQ3ne` *and* `http://bit.ly/x5fBaS`.

www.it-ebooks.info

STANDALONE SITECORE PRODUCTS

Sitecore provides the standalone Intranet Portal (SIP) and Foundry products that include the Sitecore CMS. You can use Sitecore Intranet Portal to manage intranets. You can use Foundry to simplify the management of a large number of websites with similar characteristics, often managed by different groups of CMS users.

Sitecore Intranet Portal

SIP is a standalone solution for managing full-featured intranets, which are internal websites intended for the members of an organization. Intranets facilitate internal communication, information sharing, and collaboration within an organization, and can help to ensure compliance with government regulations and other standards. You can also use your intranet to implement processes such as data entry and maintenance forms.

SIP includes data templates, presentation components, and other features to help you deploy your intranet as quickly as possible. Because SIP uses the Sitecore CMS as a foundation, everything that you learn about building public sites with Sitecore applies to your intranet solutions, so you can easily extend these components and add features. While most SIP users access the published website using an in-context editing environment that functions like a simplified Page Editor, given appropriate access rights, they can also access the Content Editor, the Sitecore desktop, and all the applications available in the desktop. In addition to relevant features provided by Sitecore CMS, such as translation and versioning, SIP includes the following:

- Presentation components, including web and print layouts as well as numerous individual presentation controls
- Personalized content
- Wiki-like in-context editing
- Discussion forums
- Blogs
- Image slideshows
- Intranet search
- Employee database and phone book
- Interactive organization charts
- Activity calendar
- Support for web parts (SharePoint presentation components)
- Cafeteria (canteen) menu
- Polls
- FAQs

www.it-ebooks.info



For more information about Sitecore Intranet Portal, see `http://bit.ly/vTZph5` *and* `http://bit.ly/vwVMdE`.

Sitecore Foundry

Sitecore Foundry enables you to centrally provision and manage large numbers of websites, potentially in the thousands. In addition to relevant capabilities provided by Sitecore CMS, Foundry provides predefined data templates, presentation components, and other features to make it easier to deploy new websites. These include functional components such as discussion forums, event calendars, newsletters, photo galleries, and ad rotators, as well as site configuration wizards and styling skins. Further, Foundry automates the process of creating users and roles, as well as assigning rights, to simplify the process of bringing up a new site for a new department or other organization. Foundry can also assist in ensuring visual consistency throughout a large number of websites, including instantaneous global updates to those components, while providing the owners of each site with an appropriate level of control.



For more information about Sitecore Foundry, see `http://bit.ly/tsuhi8` *and* `http://bit.ly/trLSKg`.

SITECORE APP CENTER

The Sitecore App Center (SAC) enables Sitecore customers to access an online marketplace of pre-integrated applications and Internet cloud-based services to extend the value of their Sitecore solution.

SAC consists of a user interface available on the Sitecore menu within the Sitecore desktop. Using this interface, you can discover, activate, and manage software and services complementary to CMS. You can use the SAC to add, reconfigure, and remove services at your will and on your schedule. The SAC currently supports email delivery services, spam, and client previewing applications, as well as page previewing tools. Over time, the App Center exposes additional pre-integrated services to include social media monitoring, Sitecore Azure management, search engine optimization (SEO) services, translation for content internationalization and localization, standards and additional forms of compliance, social media monitoring, digital print, device detection, and more.



For more information about the Sitecore App Center, see `http://bit.ly/sHgrFZ`.

www.it-ebooks.info

OPTIONAL MODULES

This section describes the following optional Sitecore modules that you can license and use in your Sitecore solution:

- **Active Directory (AD)** — Manage user authentication, user profiles, and roles in Active Directory
- **Adaptive Print Studio (APS)** — Apply personalization to CMS content to generate print materials
- **Calendar** — Expose event calendars on your managed websites
- **dtSearch** — Integrate the dtSearch crawler for searching your managed websites
- **E-Commerce** — Implement e-commerce solutions
- **SharePoint Integration Framework (SPIF)** — Integrate between SharePoint and your Sitecore solutions
- **Search Engine Optimization (SEO) Toolkit** — Analyze and improve your search engine index ranking
- **Sitecore Azure** — Deploy your Sitecore solutions to the Microsoft Windows Azure Internet cloud

Active Directory

The Sitecore Active Directory (AD) module is a set of ASP.NET security providers that allow Sitecore to access users, roles, and user profiles defined in Microsoft Active Directory. While you can configure the AD module for CMS users, users of one or more published websites, or both, most organizations use AD for their internal users in the CMS environment. Resources defined in AD appear as native functionality, replacing the default implementation of these features.

The AD module supports single sign-on (SSO). If the operating system user running the browser has authenticated against the AD security domain configured in Sitecore, the instance can automatically authenticate the browser as a CMS user without requiring the user to enter a username and password at the Sitecore login screen. This works even if the user did not select the Remember Me check box on the Sitecore login screen when they previously logged in from that browser.



For more information about the Active Directory module, see `http://bit.ly/rMAcPB`.

Adaptive Print Studio

The Adaptive Print Studio (APS) module brings the sophistication of web content management to the print world by integrating Sitecore CMS with Adobe InDesign. APS is an important component of Sitecore's ability to bridge online channels such as the web, email, and social networks with offline channels, including print.

www.it-ebooks.info

With APS, you can create targeted and personalized print-grade assets from content stored in the CMS. For example, you can generate catalogs that emphasize certain product lines based on a customer's order history and site browsing history. APS can generate QR codes (Quick Response codes, `http://bit.ly/HmgCZO`) with embedded campaign tracking codes that link back to a website, allowing you to track conversions across channels.

APS is just one example of how you can reuse content in a Sitecore repository across a number of channels. With APS, users can easily share content between websites and printed materials. APS enables Sitecore's standard content management features for printed documents, including versioning, workflow, support for multiple languages, access control, and collaborative authoring. These features can significantly reduce duplication and inconsistencies that occur with a copy-and-paste approach commonly used for repurposing content.

APS provides native connectors to Adobe InDesign and InCopy. Print designers and editors who work in these tools can edit and use content directly from Sitecore. In addition, users of these external tools can save documents directly to the Sitecore content repository. Once the content exists in Sitecore, other designers and editors can access them without the need to manually copy or email files, fostering a more collaborative environment for building and editing these documents.

APS provides a marketer-centric application for building printed documents, providing functionality similar to InDesign for marketers. The browser-based Print Studio Editor application provides basic features similar to those of InDesign, but exposes those facilities to marketers. Documents created with the Print Studio Editor are InDesign-compatible, allowing marketers and designers to collaborate in new ways.



For more information about Adaptive Print Studio, see `http://bit.ly/AcMzqM`.

Calendar

The Sitecore Calendar module facilitates the creation of event calendars within Sitecore solutions. You can apply access rights to each event definition item to control who can see the event. The Calendar module provides day, week, and month views, with both read-only and read-write access. Site visitors can filter the events listed on the calendar, and Sitecore provides components that enable them to select the day, month, and year to display.



For more information about the Sitecore Calendar module, see `http://bit.ly/vWE73o`.

dtSearch

The Sitecore dtSearch module provides preconfigured integration with the third-party dtSearch search engine so that you can easily expose search features on the websites that you manage with Sitecore. The dtSearch indexer generates a search index for the solution, and the dtSearch module provides presentation controls and APIs to access that index.

www.it-ebooks.info



For more information about the dtSearch company and its products, see `http://dtsearch.com`. *For more information about the dtSearch module from Sitecore, see* `http://bit.ly/rwRdwz`.

E-Commerce Modules

Sitecore provides two optional products that can help you implement solutions that engage in electronic commerce:

- **E-Commerce Services (SES)** — SES provides a framework for building e-commerce solutions on the Sitecore platform. SES brings the benefits of CEP — visitor tracking, segmentation, and behavior-based content targeting — to commerce sites. SES includes an e-commerce API that supports integration with external systems such as ERPs (Enterprise Resource Planning systems). SES also includes a number of dedicated interfaces for tasks such as merchandising and order management.
- **Insite for Sitecore** — Built on SES, Insite for Sitecore supports all of the features of SES along with more advanced, rule-based commerce features such as pricing, promotions, shipping, and taxation. Insite for Sitecore also provides an integration framework supporting bi-directional communication and synchronization with ERP and other external systems.



Sitecore E-Commerce Services replaces Sitecore E-Commerce Fundamentals Edition (SEFE). For more information about Sitecore E-Commerce products, see `http://bit.ly/uH4CML`.

SharePoint Integration Framework

The SharePoint Integration Framework (SPIF) provides advanced, bi-directional content sharing capabilities between Sitecore and SharePoint. SPIF can expose data in SharePoint as items in Sitecore, which enables you to use existing presentation components to present that data. Sitecore can also present entire SharePoint objects such as lists within the CMS, exposing collaboration components within pages delivered by Sitecore.



For more information about the SharePoint Integration Framework, see `http://bit.ly/sRgwYO`.

Sitecore Search Engine Optimization Toolkit

The Sitecore Search Engine Optimization (SEO) Toolkit provides content editors with information they can use to improve ranking values assigned to web pages by Internet search engines. The SEO Toolkit incorporates relevant information directly into the Page Editor, making it available to content authors in the context of the page.

www.it-ebooks.info

The SEO Toolkit separates information into the following tabs:

- **Page Information** — Displays basic page metadata, including the page title, description, keywords, and language. This tab also provides a summary of information detailed on other tabs, including headings, images, links, and meta tags.
- **Text Only View** — Renders content as it appears to search engines, without formatting, styling, images, or media. You can use this tab to analyze how content appears to search engines.
- **Keywords** — Analyzes the words and phrases to determine those most prevalent in the page. This tab provides links to perform searches for each word, and word combinations, using popular search engines.
- **Search Engines** — Displays inbound links as reported by popular search engines. You can use this tab to investigate how other sites link to the page.
- **Headings** — Renders the heading tags embedded in the page as a hierarchy. When you click the text of a heading tag, the Page Editor scrolls to that tag and displays the type of the tag, such as heading1 or heading2.
- **Images** — Lists all images on the page, along with a thumbnail of the image, its alternate text, location, and dimensions.
- **Links** — Lists all links on the page, including the text of each link, and indicates whether the link references a resource that does not exist.
- **Meta Tags** — Lists the meta tags included in the page header, including the name and value of each tag.



For more information about the SEO Toolkit, see `http://bit.ly/w4wApM`.

Sitecore Azure

Sitecore Azure enables you to automate deployment of your Sitecore solutions to the Microsoft Windows Azure Internet cloud, using SQL Azure for storage rather than Microsoft SQL Server or Oracle. For more information about the Microsoft Windows Azure cloud, see `http://bit.ly/Znsdo`.

Sitecore Azure provides at least the following benefits:

- Unlimited, flexible, near-instantaneous scalability
- Real-time database backup and failover
- Automated deployment of content and code from content management to content delivery
- Distribution of redundant content delivery infrastructure across geographies
- Reduced total cost of ownership

www.it-ebooks.info

Sitecore Azure can help to improve solution performance by placing content delivery servers closer to users in disparate locations, and increase reliability by providing redundant infrastructure within and between locations, which also facilitates disaster recovery.



For more information about Sitecore Azure, see `http://bit.ly/utriC2` *and* `http://bit.ly/suA3zb`.

TAKE HOME POINTS

Using the Sitecore products described in this chapter, you can implement solutions to the most common types of challenges in CMS implementations more rapidly than if you had to build the infrastructure yourself. These tools range in scale from relatively simple components such as calendars to the full-featured Digital Marketing System (DMS), which extends the CMS with engagement and analytics. You can also use the standalone Sitecore Intranet Portal (SIP) and Foundry products, which use the CMS as a foundation, to deliver your Sitecore solutions sooner, with lower maintenance costs and total cost of ownership (TCO).

www.it-ebooks.info

www.it-ebooks.info

11

Sitecore Best Practices, Tips, and Tricks

WHAT'S IN THIS CHAPTER?

- Optimizing the information architecture
- Getting the most from the media library
- Applying security
- Managing projects
- Taking advantage of the layout engine
- Enforcing workflow
- Increasing performance
- Maximizing usability
- Generating simple reports
- Using the browser-based debugger

This chapter describes best practices and tips for working with the Sitecore ASP.NET content management system (CMS). You can use these suggestions to maximize the usability, performance, and security of your Sitecore solutions, as well as increase your productivity. This chapter includes information on a wide variety of subjects including web project and CMS program management, hardware requirements estimation, system administration, data infrastructure, presentation, workflow, security, the media library, configuring Sitecore user interfaces (UIs) including the Rich Text Editor (RTE), ad hoc report generation, keyboard shortcuts, the browser-based debugger, and a number of other topics.

www.it-ebooks.info



To be honest, I found it difficult to determine which content defines best practices as opposed to opinion. As such, this chapter contains guidance that you can follow to get more value and achieve greater productivity and usability with your Sitecore solutions, not hard and fast rules that you must obey or face catastrophe. I also had trouble separating best practices from tips and tricks, especially for subjects such as the Content Editor that span both areas. Many of the topics discussed in this chapter do not fit nicely under a single heading; I recommend that you read the entire chapter, even if some of the content seems irrelevant. A number of the suggestions provided in this chapter may appear duplicated elsewhere in this book, while others did not fit neatly within any of the previous chapters.

I have spent much of my eight years with Sitecore conducting research into the CMS product's capabilities. Even after all this time, I continue to learn things about the CMS (often from the Sitecore developer community), and Sitecore continues to implement new features; as such, this chapter can never be complete. While some of the content in this chapter derives from that experience and my four years implementing CMS solutions prior to joining Sitecore, I also scoured the Internet, solicited input from parties internal and external to Sitecore, conducted interviews, and encouraged peer review of this work in order to provide the most comprehensive resource possible.



If you have questions, comments, suggestions, or other feedback regarding the information in this chapter or this book, please comment at `www.wrox.com`.

SITECORE BEST PRACTICES

This section contains suggestions that I consider best practices for working with the Sitecore ASP .NET CMS. A number of existing resources describe Sitecore best practices; some of these may differ from or add to my perspective.

The Sitecore Best Practices Blogs (`http://bit.ly/qWllih`) provide expert advice on a number of topics, including the following:

- Accessibility and compliance
- Community and social media
- Electronic commerce (e-commerce)
- Landing pages and experience testing
- Lead management and email marketing
- Managing multiple websites
- Marketing and sales automation

www.it-ebooks.info

- Mobile web solutions
- Multilingual websites
- Persona and personalization
- Search engine optimization (SEO) and marketing
- Taxonomy and information architecture (IA)
- User experience (UX) and usability
- Web analytics
- Web implementation and integration

See also the Sitecore Recommended Practices document (`http://bit.ly/opaCUF`).

Managing Sitecore Implementations

While no project team can possibly fully understand, let alone document, all project requirements in advance, implementing a solution without clear requirements can result in mid-project shifts to the technical architecture, which can put the entire solution at risk. In my opinion, requirements analysis should account for at least half of the project timeline, including a variable amount of time after implementation begins. When you migrate an existing website to Sitecore, you can consider some portion of its prior existence as a portion of the requirements analysis phase for that replacement system. Avoid inflexible solutions that could be difficult to adapt as unforeseen requirements emerge.

Follow best practices for securing, optimizing, scaling, and otherwise applying all technologies underlying your Sitecore solutions, including the following:

- Microsoft Windows, Internet Information Services (IIS), and ASP.NET
- Web standards, including HTTP, HTTPS, HTML, XHTML, XML, XSL, and CSS
- Client technologies, including JavaScript and Adobe Flash
- Underlying relational database management platforms, such as Microsoft SQL Server

Identify stakeholders within the organization to sponsor the project, and allocate sufficient resources to fill all the required roles throughout the overlapping phases of the solution life cycle. These phases, which often overlap, include requirements analysis, platform selection, system administration, developer and eventually CMS user training, implementation, development of custom training and documentation materials, content migration and maintenance, testing, user acceptance, production support, and other aspects of the project. Plan sufficient time for training, documentation, various types of testing, and other components frequently omitted from web solution project plans.

Before you determine your technical approach to achieve various project objectives, attend Sitecore developer training, especially the Sitecore CMS Developer Foundations (SDF), Sitecore CMS Website .NET Developer (SND), Sitecore CMS .NET Developer Day of Prototyping (DOP), and Sitecore CMS Advanced .NET Developer (AND) courses. For more information about Sitecore training, see Appendix A. Additionally, review available Sitecore documentation. Participate in

www.it-ebooks.info

online forums about Sitecore, as well as local user groups when possible. Make use of Sitecore support, which has likely seen requirements similar to yours previously.

Involve CMS users early in the requirements analysis, CMS selection, and implementation processes. CMS users can provide valuable insight that helps to define and refine requirements, and can help build user confidence and foster eventual acceptance of the system.

Develop and optimize the solution before attempting to determine hardware requirements or deploy Sitecore on the production hardware. You can use a single machine for both the web server and the database server in development and some test environments, but you should use production-class hardware in the load-testing environment, which should include one or more separate database servers.

Implement appropriate exception management as described in Chapter 5, and monitor the Sitecore logs for errors and exceptions as described in Chapter 6. You should be able to resolve or at least understand every error message that Sitecore generates. In addition to Sitecore error management facilities, consider ASP.NET application and error page functionality, as well as IIS error pages and logs.

Avoid excessive and unnecessary customization, and remember that product extensions can interfere with Sitecore and with each other. New Sitecore developers frequently welcome Sitecore's flexibility and capabilities without understanding the ramifications of altering components of such a complex system. You can often satisfy your users by meeting most requirements and approximating others without significantly enhancing Sitecore.

In many cases, you can achieve an objective that you think might require Sitecore-specific customization by instead using existing features of the ASP.NET Framework itself. Just because you can extend something using a Sitecore-specific approach does not mean that approach is necessarily the best way to achieve that goal. In some cases, decoupling a feature from Sitecore has advantages over implementing a solution that depends on the CMS. Use Sitecore features where they facilitate your objective, but remember that you have the power of the entire ASP.NET Framework behind your Sitecore solutions.

Use Web.config include files to contain your configuration settings, class overrides, and other changes to the `/configuration/sitecore` section of the Web.config file. For more information about Web.config include files, see my blog post at `http://bit.ly/qp0vps`.

You can use the Sitecore Recycle Bin if you accidentally delete something, but try to avoid relying on it.

Estimating Hardware Requirements

Because numerous variables affect a solution's capacity, you cannot accurately estimate hardware requirements without first developing the solution and testing it on hardware equivalent to what you intend to use in production. These variables include hardware performance, such as web servers, database servers, load balancers, and the network, as well as any dependencies on external databases and other systems.

To estimate hardware requirements for a Sitecore solution:

1. Implement the solution (or a close approximation thereof).
2. Optimize the solution.

www.it-ebooks.info

3. Monitor resource usage while load testing the solution on hardware equivalent to that you will use in the production content delivery environment to determine the peak traffic supported by a single instance.
4. Monitor resource usage on the database server while adding content delivery instances to determine how many a single publishing target database can support.
5. Estimate the number of content delivery instances and database servers required to meet projected peak traffic requirements.

Administering Sitecore Solutions

At the time of this writing, I recommend 64-bit Microsoft Windows Server 2008 R2 for all web servers and database servers. Unless instructed otherwise by Sitecore, always patch Windows, including Internet Information Services and ASP.NET, as well as additional supporting technologies, including Microsoft SQL Server, client operating systems, and web browsers.

Whenever possible, use the Sitecore setup program to configure Sitecore. If you instead install Sitecore from a `.zip` distributive or from an archive of an existing Sitecore solution, secure the solution completely as described in The Sitecore Installation Guide (`http://bit.ly/pv7X3B`). Always review product release notes (`http://bit.ly/pyPUPV`), including known issues for the versions you use as well as issues fixed in subsequent releases that might affect your solution.

For performance and capability, I recommend integrated mode for the Internet Information Services application pool associated with the IIS website hosting the Sitecore solution.

The speed of network connections between web servers and database servers can have a fundamental impact on the performance of Sitecore solutions. Configure Sitecore web servers to access database servers in the same location rather than remote database servers, and optimize network connections between those machines.

For security, scalability, resiliency, performance, and potentially other benefits, separate the content management environment from the content delivery environment. Do not implement content delivery features that depend on content management functions, such as attempting to access the version history of an item or write to a Sitecore database.

Separate development environments from production and implement at least one test environment between the two. Clearly differentiate content assets from code assets, including both items and files. Use source code management software and release management processes and tools to deploy code to all environments after development, and use Sitecore publishing to deploy content from the production content management environment to the production content delivery environment.

Implement maintenance plans to clean up and defragment filesystems, to defragment databases and indexes, and to back up the entire solution. Test the system recovery plan, which includes testing the backup system and any cold, warm, and hot failover solutions.

When feasible, rather than introduce settings and external configuration files, use the configuration factory and Web.config include files to set properties on instantiated types. For more information about the configuration factory, see my blog post at `http://bit.ly/A7Ae1s`. Only when multiple components share configuration values should you consider adding settings to the Web.config file or

www.it-ebooks.info

using external configuration files that all such components can access. Such approaches are preferable to using the configuration factory to pass the same values to multiple individual components.

Test each presentation component under a variety of conditions, such as with varying amounts of valid and invalid data on the published site, in the Page Editor, as anonymous and authenticated users, and under any other conditions specific to your solution. Require that each build of the solution pass automated regression testing before deployment to production. Always test using accounts with access rights configured similar to those of end users, rather than test with administrative accounts. For more information about testing, see Chapter 8.

Architecting Data Infrastructure

For performance and usability, avoid creating items with hundreds of children. Use a narrow, deep information architecture that helps CMS users focus on proper data categorization. In presentation components and CMS user interfaces such as the content tree, performance and usability degrade as the number of children under a single item increases, similar to performance and usability with a large number of subdirectories and files on a filesystem. Prevent any item from having hundreds of children, and consider whether you will likely add items under existing items in the future. Avoid structures that could require you to apply logic to a large number of items, especially in the content delivery environment, such as any construct that uses of the descendant axis. When necessary, use alternative techniques such as search indexes to evaluate a significant amount of items at once.

Like filesystem subdirectories, if you put too many items in one place, it becomes very hard to find anything. Validating, automatically moving items, dynamically manipulating effective insert options, and other techniques can help you limit the number of children under any item. Consider using a relational database for data that you cannot represent as a relatively narrow hierarchy.

You can update the `MaxTreeDepth` setting in the Web.config file to change the limit depth limit allowed for nesting items in a Sitecore database. The default depth limit is 20 levels. Deep information architectures, especially containing long item names, can result in URLs that exceed path length limitations in IIS.

When appropriate, use the information architecture to drive navigation. Consider that the name of each item and its location in the information architecture relative to the home item of the managed website in which that item appears determine its default URL. Implement item naming and path conventions to support search engine optimization (SEO).

You can sometimes implement various forms of inheritance for custom purposes, such as by inheriting branding or other features from an ancestor. For example, a presentation control that renders the site logo could use the value of an Image field in the nearest ancestor of the context item that defines a value for that field. You can use the same technique to select a CSS file or other components of the page. You can even create fields to allow CMS users to select presentation components to include in the page, but you must implement a solution to render those components.

Do not allow two items with the same name at the same level of the information architecture. Validation helps enforce this requirement globally (configure global validation in the `/sitecore/system/Settings/Validation Rules/Global Rules` item). You can also use events, pipelines, and other techniques to enforce item naming requirements and conventions.

www.it-ebooks.info

Without good reason, do not delete, update, move, or rename items that appear by default in the Master database or the Core database created when you initially installed Sitecore. Inadvertent changes to these items could interfere with Sitecore functionality.

Never update a publishing target database directly, neither through the Sitecore application programming interface (API) nor through CMS user interfaces. Always update items in the Master database and publish such changes to the publishing target databases rather than editing directly in a target database.

Always use Sitecore APIs, and avoid writing queries directly against Sitecore databases. Sitecore caches data retrieved from databases, and validates data, clears caches, invokes event handlers, and activates other features both before and after updating data. Reading directly from and writing directly to Sitecore databases without using the APIs circumvents these features and could have unpredictable results, such as caches that do not contain the latest updates.

Most CMS users have read access to items to which they generally do not navigate. Hide items that are not relevant to most users. Sitecore hides a number of items by default, including the `/sitecore/layout`, `/sitecore/system`, and `/sitecore/templates` items and many of their descendants. To hide an item, select the item in the Content Editor, click the Configure tab, and then click Hide Item in the Attributes group on the Ribbon. To view hidden items, click the View tab, and then check Hidden Items in the View group.

Developers often overlook validation, a relatively simple feature that can address a wide range of requirements. For example, to enforce information architecture requirements, you can write event handlers that create item structures and move items into those structures based on the data they contain automatically, or you can implement validation to encourage or require users to create an item structure and move items into that structure manually. Implementing validation introduces less risk than other forms of customization. For more information about validation, see Chapter 7 and my blog post at `http://bit.ly/z747du`. You can also use Content Editor warnings as described in Chapter 7 and my blog post at `http://bit.ly/mYOQXT` to help users address issues with their content.

To generate consistent URLs, set the `languageEmbedding` attribute of the `/configuration/sitecore/linkManager/providers/add` element named `sitecore` in the Web.config file to `always` or `never` rather than the default (`asNeeded`), which can result in multiple URLs for a single content item in a single language. To avoid search engines indexing two URLs that activate the same content, if an item has an alias, use it whenever you generate a link to that item. For a solution that applies aliases automatically, see the LinkProvider (`http://bit.ly/AeRE0O`) Sitecore Shared Source module.

To eliminate the `.aspx` extension in item URLs, ensure that IIS uses ASP.NET to process requests that do not contain the `.aspx` extension, and then set the `addAspxExtension` attribute of the `/configuration/sitecore/linkManager/providers/add` element named `sitecore` in the Web.config file to `false`. For information about configuring IIS to use ASP.NET to process requests without the `.aspx` extension, see the guide to The Sitecore Dynamic Links Guide (`http://bit.ly/ri6Oww`).

When you move or rename an item that you have already published, its URL changes. Changing the URL of a piece of content can adversely affect search engine indexes, links from external sites to your content (including bit.ly links and other shortened URLs, such as those that appear in Twitter tweets), browser bookmarks, and other references to that content. Create items in the correct

www.it-ebooks.info

locations or move them manually or automatically to the correct locations immediately after creating, and avoid renaming items after publishing them.

Designing Data Templates

The Sitecore Client Configuration Cookbook (`http://bit.ly/vz8fnc`) contains specific instructions to implement many of the suggestions in this section that can improve the usability of data templates.

Invest significant resources into information architecture (IA) analysis, including requirements for the content tree, data templates, section and field names and grouping, standard values, workflow, security, insert options, relationships between items, languages, and even layout details. Until the organization reaches consensus as to the information architecture, you can implement the presentation layer using simple wireframes. Before you implement the IA or develop components that depend upon its structure, ensure that CMS users consider a description or visual depiction of the information architecture to be logical and relatively complete.

In designing the information architecture and data templates for your Sitecore solutions, remember that Sitecore locks, versions, translates, publishes, and puts each version in each language through workflow separately. This could encourage you to consolidate data templates, enabling you to apply these features to a greater number of field values in unison, or to separate data templates further, leading to greater flexibility in managing different groups of field values.

Organize your data templates into data template folders under the `/sitecore/templates` item. First categorize data templates by project, such as by global data templates and those specific to individually managed websites. Then categorize data templates by purpose, such as base templates and page templates. Apply additional classifications as appropriate for your solution. Use the same approach to folder or subdirectory structures for other infrastructure components, such as definition items for presentation components and the corresponding hierarchy of files, especially when managing multiple sites in a single Sitecore environment.

Use appropriate, friendly names for data templates, sections, fields, branch templates, command templates, and other types of definition items that users access. Field names and titles and data template names appear in CMS user interfaces, providing users with contextual information about data types, existing content, and the type of data to enter. Use field names that are helpful to CMS users, or provide titles for fields with unfriendly names. So that developers and CMS users use the same terms when referring to specific fields, apply titles to fields only when necessary to increase usability. In definition items for data templates and data template fields, as well as individual items that require explanation, to provide contextual information to CMS users working with those items, click the Configure tab in the Content Editor, and then click Help in the Appearance group on the Ribbon.

To optimize performance, store and use item IDs rather than paths, names, or more complex objects. Whenever possible, reference items by ID rather than path or name. For example, use the Droplink data template field type, which stores the ID of the selected item, rather than the Droplist data template field type, which stores its name. For usability, display item paths in user interfaces instead of or in addition to item IDs.

Avoid hard-coding paths and field names and rely instead on classes such as `Sitecore.ItemIDs`, `Sitecore.FieldIDs`, and `Sitecore.TemplateIDs`. To expose the IDs of key items, data template fields, and data templates in your solutions as constants, create classes similar to these.

www.it-ebooks.info

To help define consistent information architecture on which presentation components may depend, apply insert options to the standard values of your data templates. Minimize the number of users with permissions to insert items using arbitrary data templates, branch templates, and command templates. As described in Chapter 2, to restrict which users can create items of arbitrary types, configure access rights for the `/sitecore/content/Applications/Content Editor/Menues/New/Insert from Template` [sic] item in the Core database.

Do not allow two fields with the same name within a single data template. Validation for data template definition items could help you enforce this requirement.

Take advantage of data template inheritance whenever possible. When defining the information architecture of your solution, remember that data templates support multiple inheritance, meaning that a data template can inherit from multiple base templates. You can create base templates used only to define sections and fields to include in other data templates, where no actual items exist based directly on those base templates. To avoid complexity, try to keep the number of base templates to a minimum without compromising your design goals. Avoid circular template inheritance, where a data template inherits directly or indirectly from itself. When standard values should differ for a single data type in different contexts, create a new data template that inherits from and overrides standard values in the original data template, but defines no additional sections or fields.

Even when two types of pages appear to have the same structure, consider implementing multiple data templates based on a common base template, especially when logic depends on the data type. Using separate data templates initially will assist with separating layout details or other properties of each data type in the future.

Sitecore imposes no specific limits to the number of sections and fields that a data template can contain or the number of base templates that you can add to a data template. Try to separate data into as many fields as possible without adversely affecting performance or usability. Avoid data templates with very large numbers of sections and fields (including sections and fields defined in base templates), and especially avoid a large number of Rich Text fields in a single data template. The number of permissible fields in a data template varies according to factors such as solution complexity and user sophistication. Excluding the standard template itself (which generally does not appear in user interfaces due to abstraction by the Ribbon), data templates with more than a dozen sections or a hundred total fields may indicate an opportunity to improve the information architecture. When appropriate, replace groups of fields with child items containing equivalent data, or allow the user to choose items containing that data using a selection field. You can use branch templates to help users create structures meeting information architecture requirements. When planning the information architecture, remember that you will likely add fields to data templates over time to meet emerging requirements.

Group fields into sections based on commonality and to compartmentalize related data elements. Sort data template fields into a logical order within each data template section. Sort important data template fields and those that users will likely access most frequently toward the top of each data template section, and sort sections that CMS users will access most frequently toward the top of each data template.

Assign icons to data templates and data template sections as well. For example, if a data template for news articles contains a section named News, you could use the same icon for both the data template and the data template section. Remember to set the icon for the data template itself rather than in its standard values. A subsequent section of this chapter describes how to apply icons.

www.it-ebooks.info

Sitecore does not support repeating fields. As an alternative to repeating fields, in some cases you can use a selection field in a data template, such as the TreelistEx field type, to allow the user to choose some number of items containing field values to use as if the fields in those items were native to the item containing the section field. Alternatively, you can use children of an item to represent what you would otherwise manage with repeating fields in a single item. In other cases, you can use the IFrame data template field type, a custom editor, or a custom data template field type to store data as XML or another data format containing multiple elements as a single field value.

Consider providing multiple fields where you might expect a single value, especially to support internationalization, in which case values in different languages can have drastically different lengths. For example, you can use the item name or display name as the default text of links to an item, the value of the HTML `<title>` element, the value of a heading element on the page, and the text of the breadcrumb for the page. Alternatively, you can provide separate fields for each of these and any other types of title values. Instead of requiring a value for each of these fields, you can use logic to fall back to the value of another field when no value exists for one of these fields, eventually falling back to the display name and item name.

Use data template fields and renderings to define metadata for each page. For example, you can create a `Single-Line Text` field to populate the `<title>` element of the HTML page, `Multi-Line Text` fields to define values for the `<meta>` keywords and description elements, and a presentation control in the `<head>` element of the layout to render those field values. You can implement a classification taxonomy consisting of a hierarchy of items containing only metadata, and then use selection fields such as `TreelistEx` to enable users to select entries from the taxonomy to associate with the page.

In addition to content and metadata, you can use data template fields for control information, such as using a Checkbox field to control whether an item should appear in the site map, or using a TreelistEx field to select items for a presentation component to list when rendering an item.

Excluding data template standard values, do not store items with layout details except as descendants of the home item of a managed site.

To optimize Content Editor performance, use the TreelistEx data template field type in favor of other selection field types, especially when the selection tree is large or where a single data template contains a large number of selection fields. In the Content Editor, the TreelistEx field type renders the selection tree only if the user clicks the Edit command above that field. Therefore the TreelistEx field type is more efficient than the Treelist field type, which renders the selection tree each time a CMS user selects an item containing such a field in the Content Editor.

Use the Source property in definition items for data template to control what CMS users can see and pick in selection fields. For information about what you can use in the Source property for various field types, see The Sitecore Data Definition Reference (`http://bit.ly/nmGuiB`) and The Sitecore Data Definition Cookbook (`http://bit.ly/oc8F9K`).

To set the visual height of a data template field in the Content Editor, in the field definition item, click the Tree Node Style command in the Appearance group on the Configure tab in the Content Editor or the Template Manager. For example, to set the visual height of a Rich Text field to 300 pixels, enter a CSS value such as `height:300px`.

www.it-ebooks.info

Items created from branch template do not reflect subsequent structural updates to those branch templates. If you use branch templates to create structures that could vary over time, consider if and how you should update existing items created from those branch template definitions when they change.

For more information about best information architecture practices for Sitecore, see my blog post at `http://bit.ly/pSZqdf`.

Applying Standard Values

Whenever possible, use standard values to store data shared among multiple items. Standard values are especially important for defining layout details, insert options, and the initial workflow for each data template. Standard values can include default values for fields, tokens for expansion (such as `$now`), instructions to the user creating the item, lorem ipsum text, or any other kinds of values. Some solutions use lorem ipsum text because it helps people evaluating visual elements to focus on design rather than content. For more information about lorem ipsum text, see `http://bit.ly/vnGMRj`. You can find tools that generate lorem ipsum on the Internet.

Configuring Insert Options

Along with validation, use insert options to enforce information architecture requirements. Your information architecture documentation should specify insert options for each data template and item, as well as any rules for determining insert options for all items, individual items, or individual types of items. Avoid declaring insert options in individual items in favor of standard values, insert rules, insert options rules, and `uiGetMasters` pipeline processors, as well as controlling the `item:create` and `insert:show` access rights. To intentionally repeat something important that appeared earlier in this chapter, minimize the number of users who can insert items using arbitrary data templates, branch templates, and command templates.

Managing Multiple Sites

If you manage multiple sites on a single Sitecore instance, insert additional managed site definitions before the default `/configuration/sitecore/sites/site` element named `website` in the Web.config file after all other existing `/configuration/sitecore/sites/site` elements. Remember to update the configuration of existing handlers for the `publish:end` and `publish:end:remote` events in the Web.config file to clear the output caches for any sites that you add. Remember that the name of the `hostName` attribute in `/configuration/sitecore/sites/site` elements in the Web.config file is case sensitive, and that you can use the `targetHostName` to specify the hostname to use in URLs if the `hostName` attribute contains wildcards (`*`) or pipes (`|`), or if you have a preferred hostname for a web property. Unless you add or remove sites frequently, I advise that you maintain site configuration in the Web.config file, rather than investigate solutions that manage site configuration as Sitecore items. For more information about managing multiple sites, see Chapter 9.

Working with Multiple Languages

When you retrieve content from items other than the context item, and to avoid linking to items for which no version exists in the context language, check for the existence of versions in the context language before working with such items. To determine if an item contains more than zero versions in .NET code, you can check the `Sitecore.Data.Items.Item.Versions.Count` property.

www.it-ebooks.info

In XSL, you can use the `sc:HasVersions()` extension method to check whether an item contains a version in the context language. The following XSL fragment demonstrates how you can use the `sc:HasVersions()` extension method to generate an unordered list of links to the children of the context item that contain at least one version in the context language:

```
<ul>
  <xsl:for-each select="$sc_currentitem/item[sc:HasVersions(.)]">
    <li><sc:link><xsl:value-of select="@name" /></sc:link></li>
  </xsl:for-each>
</ul>
```

The following XSL fragment demonstrates how you can use the `sc:HasVersions()` extension method to determine whether a specific item (the `/sitecore/content/home/test` item) exists and contains at least one version in the context language before generating a link to that item:

```
<xsl:variable name="item" select="sc:item('/sitecore/content/home/test', .)" />
<xsl:if test="sc:HasVersions($item)">
  <sc:link select="$item"><xsl:value-of select="$item/@id" /></sc:link>
</xsl:if>
```

Avoid mixing multiple languages on a page or linking to content that does not exist in the context language. If you must link to pages in an alternate language, indicate that language — for example, by including the name of the language in parentheses after the link — and attempt to fall back to the user's preferred language if the user clicks a link from that page for which content exists in that language.

Before you designate a field as shared or not versioned, you should understand the ramifications of making that change. While sharing and not versioning fields reduces storage requirements and can improve performance in the CMS environment for features such as the links database, the values of shared fields and fields that are not versioned do not respect publishing restrictions, especially workflow. If an item exists in a publishing target database, Sitecore can publish the current values of shared fields to that database. If an item contains a version of an item in a language in a publishing target database, Sitecore can publish the current values of fields that are not versioned in that language of the item to that target database. You cannot use workflow or publishing restrictions reliably to prevent Sitecore from publishing changes to values of fields that are shared or not versioned.

Some Sitecore solutions use the same information architecture for all languages, meaning that each site has the same home item regardless of the context language. Others use a common information architecture for one or more languages, but separate information architectures for some specific languages. Such solutions involve separate home items for each language, typically with a `/configuration/sitecore/sites/site` element in the Web.config file for each such language to specify the `hostName`, `startItem`, `language`, and potentially additional attributes of each managed site.

Avoid hard-coding any text, or images containing text, in any rendering components. Instead, retrieve field values from the context item or from specific items or use the Sitecore translation dictionary. For more information about the translation dictionary, see The Sitecore Guide for Reusing and Sharing Data (`http://bit.ly/pjNlqG`).

Consider how to render dates, numbers, and other values for different regions. You can add a processor such as that shown in Listing 11-1 to the `httpRequestBegin` pipeline to set the culture for the current thread so that the layout engine formats dates and numbers appropriately for the context language, which can include region information.

www.it-ebooks.info



LISTING 11-1 Adding a processor to the httpRequestBegin Pipeline

```
namespace SitecoreBook.Pipelines.HttpRequest
{
  using System.Threading;

  public class SetCulture :
    Sitecore.Pipelines.HttpRequest.HttpRequestProcessor
  {
    public override void Process(
      Sitecore.Pipelines.HttpRequest.HttpRequestArgs args)
    {
      Thread.CurrentThread.CurrentUICulture =
        new System.Globalization.CultureInfo(Sitecore.Context.Language.Name);
      Thread.CurrentThread.CurrentCulture =
        System.Globalization.CultureInfo.CreateSpecificCulture(
          Sitecore.Context.Language.Name);
    }
  }
}
```

You can use a Web.config include file to add the processor shown in Listing 11-1 to the `httpRequestBegin` pipeline, or add a line such as the following after the existing `LanguageResolver` in the `/web.config` file itself:

```
<processor type="SitecoreBook.Pipelines.HttpRequest.SetCulture, SitecoreBook"/>
```



When you add a pipeline processor using Sitecore Rocks, it generates a stub Web.config include file automatically.

Consider using validation in the Content Editor and adding warning markup in the Page Editor for content that does not exist in the selected language or in at least one registered language. For example, if a CMS user in the Page Editor clicks a link to an item for which no version exists in the context language, instead of rendering empty field values, at least one rendering for that item could output markup indicating that the user must create a version in that language.

Storing Relational Data

You do not need to store all the information supporting your websites in a Sitecore database. Relational databases and other storage mechanisms can be appropriate for order management data, user-generated content, and other material not developed and maintained by CMS users. You may need to develop administrative interfaces to manage such data — for example, to update that data, control what appears on the published website (such as to set an approval flag for user-generated content), or to delete obsolete records. To associate records in external systems with Sitecore items, you can store the ID(s) of any Sitecore item(s) associated with that data in a column or table in the database or otherwise in the external system.

www.it-ebooks.info

You should consider using a set of relational database tables or another alternative to Sitecore items under any or all of the following conditions:

- The data and required user interfaces to maintain that data are readily available in an existing system, or such user interfaces are unnecessary.
- The content delivery environment gathers the data, such as user-generated content.
- The data is relational, transactional, and otherwise difficult to represent as a hierarchy.
- The data is associated with a user, not secured by Sitecore access rights and role membership.
- The data does not require CMS services, including translation, versioning, workflow, locking, and publishing (separation of work-in-progress data from live data).
- Other systems require access to the data.

For example, Sitecore uses ASP.NET security providers for authentication, role membership, and profile management, rather than storing that information as Sitecore items. The default providers manage data directly in an SQL database, never abstracting those records as Sitecore items.

Coding Presentation Infrastructure

To minimize administration of layout details and maximize site consistency, apply layout details in the standard values of your data templates instead of configuring layout details in individual items.

Limit the number of layouts in favor of sublayouts and other types of presentation controls. Compared to .NET presentation components, XSL renderings have significant limitations, including a lack of compile-time error detection and no support for debugging in Visual Studio. While you can use sublayouts to generate any output, particularly subdividing pages and implementing specific web forms, sublayouts can also be useful for creating reusable groups of child presentation controls.

Use sublayouts and web controls in favor of XSL renderings. Some solutions use sublayouts for everything, with no XSL or web controls. Here are some of the advantages of using sublayouts:

- Sublayouts support a superset of the features that XSL renderings, web controls, and user controls support.
- Sublayouts provide markup templates with code-behind to separate logic from presentation.
- Sublayouts can access all Sitecore APIs and any other APIs available to the ASP.NET worker process.
- Sublayouts support placeholders, allowing dynamic nesting of presentation components to any level.
- Sublayouts support debugging with the Visual Studio debugger.
- Sublayouts support compile-time error detection.
- If it helps in the migration of existing code or the integration of third-party components, you can easily implement a sublayout as a wrapper for a single web control.

www.it-ebooks.info

Other than separation of code-behind from markup and support for placeholders, web controls share all of these traits with sublayouts.

To minimize administration of layout details and maximize site consistency, prevent excessive use of placeholders. Use static binding whenever possible and appropriate; reserve placeholders for regions of the page that present different components at different times, such as on different pages that otherwise share much of a common visual layout. Placeholders also have a minor negative impact on performance.

Configure placeholder settings for each placeholder, including the Editable check box that controls whether Page Editor users can bind presentation components to the placeholder. To control which Page Editor users can bind presentation controls to placeholders, disable write access to the corresponding placeholder settings definition items.

Design presentation components such that you can cache their output by the fewest possible criteria. For example, if some output depends on whether the user is logged in and other content depends on the context item, instead of using a single rendering to generate that output, you can implement two presentation components, and vary caching for each according to the appropriate conditions.

To expand dynamic links and add CMS features around field values in user interfaces such as the Page Editor and the Sitecore browser-based debugger, use the `<sc:html>` XSL extension control or the `sc:field()` XSL extension method instead of the `sc:fld()` XSL extension method. In .NET renderings, use the `FieldRenderer` web control or the `renderField` pipeline to retrieve field values. You do not always need to use these constructs for simple field values such as checkboxes and other field types that do not contain links or support inline editing or other dynamic features.

Use rendering parameters templates to define the parameters that users can pass to presentation controls. Associate a rendering parameters data template with each rendering definition item associated with a presentation control that accepts parameters you want users to be able to define. For more information about rendering parameters templates, see Chapter 3.

To facilitate code and content reuse across devices and languages, avoid ASP.NET master pages in favor of layouts and layout details. File-based master pages have limitations that make the Sitecore layout engine a more productive foundation for most web solutions.

While you should also upload optimized graphics, use Sitecore media library features to resize images on the server in favor of resizing images with HTML attributes or other techniques that do not reduce the bandwidth required to transfer the image from the server to the client.

Consider creating classes based on the `Sitecore.Data.Items.CustomItemBase` class to abstract the implementation details of your data templates. To make it easier to adapt your code to emerging requirements, avoid referencing the context item or context database directly, and always use a variable to represent the context item. Each presentation control should respect its data source, even if you do not currently pass a data source, which defaults to the context item. Avoid accessing the static `Sitecore.Context.Item` property in .NET code; instead, use the `GetItem()` method in the `Sitecore.Web.UI.WebControl` abstract base class web controls or the `$sc_currentitem` item in XSL renderings. For example, you can use the following code to retrieve the title of the context item from .NET:

```
Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
string title = contextItem["title"];
```

www.it-ebooks.info

The layout engine does not automatically set the data source of sublayouts or pass parameters to sublayouts. You must use a solution such as that provided in the SublayoutParameterHelper (`http://bit.ly/rAcRuy`) Sitecore Shared Source project to access the data source of a sublayout. For more information about this issue, see my blog post at `http://bit.ly/qz0zYE`.

To let users create data source items for presentation components, use the Datasource Location and Datasource Template fields in the Editor Options section of the rendering definition item:

- **Datasource Location —** The user must select a data source item from this location, or may create a data source item within this location.
- **Datasource Template —** The user must select a data source item based on this data template, or may create a data source item based on this data template.

Avoid reinventing the wheel. Seemingly diverse web solutions can have a surprising degree of technical commonality. More than likely, experienced Sitecore developers have implemented something similar to what you need to do, and may have suggestions to improve the technical approach or even sample code. Before coding a complex solution, check for an existing solution in the Sitecore Shared Source repository (`http://bit.ly/vacX7d`) and post a description of your issue on the Sitecore Developer Network forums (`http://bit.ly/vbOyf8`).

When you configure layout details for an item, ensure proper nesting of controls and correct component order. For example, if a sublayout bound to placeholder A in a layout contains a placeholder B, and a web control binds to placeholder B, then layout details should list the sublayout containing that placeholder before the web control that binds to that placeholder in the sublayout. When you add two presentation components to a single placeholder, Sitecore outputs the markup for each component sequentially in the order of these components configured in layout details.

You can configure renderings to generate some output only if the user is in the Page Editor or the debugger, or only if the user is inline editing in the Page Editor, or according to your requirements. For example, below the page, you could output the values of fields that do not otherwise appear in the page body, with or without inline editing controls for those fields. For more information about this topic, see the information in Chapter 3 about the page mode exposed to .NET code by the `Sitecore.Context.PageMode` static property and to XSL renderings by the `sc:pageMode()` extension method.

Remember to disable the Show All Files feature in Solution Explorer before debugging in Visual Studio, and to debug by attaching to the existing ASP.NET worker process rather than using the web server built into Visual Studio.

If you use a disassembler to investigate an approximation of the source code used to decompile Sitecore, be sure to investigate the constructors for any classes that you review. Constructors often contain initialization logic that can illuminate system functions.

Avoid the Sitecore browser-based Developer Center in favor of Visual Studio with the Sitecore Rocks (`http://sitecorerocks.net`) extension for Sitecore developers working with that Integrated Development Environment (IDE).

www.it-ebooks.info

Automating Publishing Workflow

Determine a publishing strategy that minimizes the number of publishing operations, the number of items published per publishing operation, and the number of CMS users who can publish. For example, unless an update includes urgent changes, instead of publishing each item manually after each change or automatically after each workflow completion, publish all updated and approved items at some regular interval.

Specify an initial workflow in the standard values of each data template for content items. You can use a simple workflow that consists of only two states and allows users to publish their changes without approval. Such a workflow consists of an editorial state from which CMS users can submit their own changes to a system workflow state marked as final to allow publication of those changes. Such a minimal workflow provides automated versioning and hence an audit trail, prevents inadvertent publishing, and supports the addition of validation, review, and other features when needed.

Do not configure excessively complex workflows, especially for features such as content translation. Minimize the number of workflows, workflow states, workflow commands, and workflow actions, as well as the number of users required to act in each workflow process. One goal of CMS implementation is to facilitate changes to the website, encouraging CMS users to keep their content fresh and accurate. The more cumbersome the publishing process, the more training and support CMS users require to utilize the system, and the less likely they will be to fully utilize that that system as intended, reducing actual the actual return on investment (ROI) of that system. Remember that people (especially approvers) are often unavailable, and when overwhelmed with workflow responsibilities, can tend to approve content as suggested by contributors instead of actually reviewing it, which reduces the intended value of that workflow. Be especially careful to eliminate bottlenecks in the publishing process. It may not be feasible to automate every potential variation of a publishing procedure.

Do not configure excessive email notification from CMS workflows; if you do, users will learn to ignore such messages, reducing or eliminating the potential value of that development investment. Use client Really Simple Syndication (RSS) feeds, scheduled tasks that generate reports, and user interfaces such as the Sitecore Workbox for day-to-day content management activities. See `http://bit.ly/tic2k9` for more information about RSS, and The Sitecore Client Configuration Cookbook (`http://bit.ly/vz8fnc`) for more information about Sitecore client RSS feeds.

Reserve workflow email for exception cases, such as if a reviewer fails to act on a piece of content within a reasonable period of time, or an author considers their change urgent. A single workflow state can contain multiple workflow commands that each transition the item to the same workflow state, but with only one command containing the workflow action to send email. You can even set access rights on workflow commands to hide the action that sends email messages from users who should not have the option to generate email notification.

To encourage users to use a particular workflow command in preference to others, sort that workflow command first among its sibling command definition items under the definition item for the workflow state. For example, if the user can use one of two commands to submit content to the next state in the workflow, and you would prefer they use the command that does not include the workflow action that sends email notification, sort the definition item for the command without the action to appear first.

www.it-ebooks.info

While workflows should generally allow users to comment when transitioning from one workflow state to another, you can prevent Sitecore from prompting users when they select a workflow command. To prevent a workflow command from prompting the user for a comment, select the Suppress Comment check box in the Data section of the definition item for the workflow command.

To optimize CMS performance, avoid storing an excessive number of versions. For example, use the default model of one version per complete workflow cycle, rather than creating additional versions within a single workflow, and periodically delete obsolete versions, such as when adding a new version or using a scheduled process or custom user interface. You can use the VersionManager (`http://bit.ly/u23G7b`) Sitecore Shared Source module to manage versions through a graphical user interface beyond that provided by the Content Editor. For a solution that deletes excessive versions automatically, see my blog post at `http://bit.ly/rfxtL7`.

Securing Sitecore Solutions

This section describes two aspects of securing a Sitecore solution:

- Applying access rights to items in a Sitecore database
- Securing the infrastructure, such as the web server and the database server

There are no truly secure systems connected to any other systems, and a web server requires connections. Securing your solutions is a goal that you never reach but that requires constant effort. Your objectives in securing systems include minimizing attack surfaces, mitigating the most significant risks, and implementing and testing a disaster recovery plans.

As with virtually every aspect of any CMS implementation, keep your security infrastructure as simple as possible. A reduction in complexity generally leads to a reduction in development, maintenance, risk, documentation, training, support, and other costs.

To minimize security administration, use security inheritance whenever possible. Items automatically inherit access rights from their parents, making it easy to secure entire sections of the solution. Specify access rights in the smallest possible number of items. Avoid security presets, which duplicate access rights instead of using inheritance. Avoid defining access rights in branch templates, as using branch templates to create items duplicates rights from the branch templates to the items created.

Grant only the access rights required. Start by granting read, and if needed, write. Few users should have rights to rename items, which changes their URLs and affects search engines, bookmarks, and other links from external systems. Few users should have permission to delete or apply access rights to items. You can apply the create access rights to control which CMS users can add items at each location in the information architecture, and you can apply access rights to insert options to restrict the types of items that different CMS users can create in those locations.

To deny access rights, rather than explicitly deny access rights for a role or user, disable inheritance of access rights to that item, and grant access rights to other roles as required. This reduces the potential for confusion and inadvertent right assignment when a user is a member of one role allowed a specific access right and another denied that same right. The Anonymous user is an exception to this general rule: to deny access to all unauthenticated users easily, deny the `item:read` access right to the Anonymous user in the Extranet domain.

www.it-ebooks.info

Separate the content management environment from the content delivery environment. If possible, place the content management environment within the internal network. You can place the content delivery environment in the periphery network or even an external hosting facility. Consider multiple content delivery environments in different geographic locations. Use different databases with separate authentication credentials in each environment, and potentially on each instance in each environment. These techniques reduce the potential for attackers to compromise your entire content delivery infrastructure if they compromise an individual content delivery environment or instance. If attackers compromise one of your content delivery instances or environments, you can restore that instance or environment from the content management environment or from one of the other content delivery instances or environments.

After you create a new user with a secure password and administrative rights, remove all the default users (excluding built-in users such as anonymous), including the default admin user. If you do not delete the default admin user, at the very least change its password. Minimize the number of users with Sitecore administrative rights. Excessive access increases the potential of inadvertent error, conflicting edits, and other potential issues with the solution. Very few users should have administrative access to any system. Whenever possible, work as a user without administrative rights. This may require that you create two accounts for yourself: one with administrative rights, and one without. Do not share Sitecore credentials, even for non-administrative accounts.

Use the Sitecore client roles to configure access to Sitecore features. Membership in most client roles allows access to CMS features, but membership in those named Sitecore Limited Content Editor and Sitecore Limited Page Editor limits access to CMS features. For example, membership in the Sitecore Client Publishing role allows CMS users access to publishing operations in the desktop, the Content Editor, and the Page Editor, but membership in the Sitecore Limited Page Editor role restricts access to features available in the Page Editor. For more information about the Sitecore client roles, see The Client Configuration Cookbook (`http://bit.ly/qS8Dc1`).

Minimize role membership for each user. For CMS users, heavily restrict membership in the Sitecore client roles that control access to CMS features, excluding those few Sitecore client roles that limit CMS users' abilities within the system instead of allow access to features. Membership in a large number of roles for a single user can lead to administrative confusion, and may indicate a need for nested roles. Evaluate whether nested roles can assist in securing your solution. Minimize users' rights to the system by limiting access rights for each role. Do not grant users or roles rights beyond those necessary to complete their operational functions. The more rights a user has, the greater chance that user has of inadvertently introducing an error.

To minimize administration of access rights, apply access rights for roles rather than individual users. Roles are typically permanent, but users come and go, and may even change roles over time. Denial of an access right overrides allowance of that access right, but that allowance of an access right to a user overrides denial of that access right to any of the roles associated with that user. This potential point of confusion is another reason to apply rights for roles rather than users.

Sitecore uses a field in the standard template to define access rights for each item. If you change access rights for an item in the Master database, remember to publish that change to the publishing target databases.

In your content delivery environment, never expose error information, such as exception types, messages, and stack traces. Trap exceptions that you can handle as close to the source as possible. Set

www.it-ebooks.info

the `mode` attribute of the `/configuration/system.web/customErrors` element in the `/web.config` file to `On` or `RemoteOnly`.

Avoid replicating security information such as by importing users from another system into the CMS membership provider without decommissioning that system or copying the ASP.NET membership tables used by Sitecore. Instead, implement a membership provider, virtual users, or any other technique that allows you to authenticate against a remote system. For more information about these options, see Chapter 4.

Follow industry best practices to secure the servers and the solution, such as patching servers and validating user input in the content delivery environment on both the client and the server.

Never leave custom administrative pages unprotected. Sitecore protects all of its default administrative pages. You should use the same subdirectories or other techniques to protect any custom administrative pages that you develop.

Minimize use of the `Sitecore.Security.Accounts.UserSwitcher` and `Sitecore.SecurityModel.SecurityDisabler` classes. In addition, minimize the number of methods called and properties accessed within blocks of code contained within these constructs. While it is safe to use these classes where necessary, a better approach is to try to grant access rights to the data that users need to access.

Heavily restrict permissions on the home item of each managed site, and grant access rights to its children and descendants instead. Most users should not need to update these items, and especially should not have access to delete or rename home items. Deleting the home item of a managed website deletes all descendants of that home item, and deleting or renaming a home item results in a configuration error within the `/configuration/sitecore/sites` element in the Web.config file.

Install the correct license in each environment. Most important, do not install a license that allows content management in a content delivery environment. An improper license can increase the solution's vulnerability to attack.

For more information about securing a Sitecore solution, see The Sitecore Security Hardening Guide (`http://bit.ly/swFfRp`).

Using the Media Library

Organize the media library to be comprehensible for CMS users, and if possible, to mimic the content information architecture. If you manage multiple sites on a single instance, you can create media folders under the `/sitecore/media library` item named after each of the managed sites, and store media for each managed site within those folders. Create nested media folders within each site media folder. For example, if the `/sitecore/content/home` item contains a child item named `About` representing a section of the default managed site named `website`, you could create `/sitecore/media library/website`, `/sitecore/media library/website/about`, and `/sitecore/media library/website/about/images` folders. These folders would contain general media for the site, media specific to the About section of the site, and images specific to the About section of the site, respectively.

www.it-ebooks.info

When possible, store Sitecore media in the database rather than on the filesystem. Use database media instead of file media whenever possible. Instead of referencing a file in a subdirectory, database media include the binary component of the media item using a field of type Attachment in the media item itself. Database media work like any other items in that you can publish the binary component along with the metadata in the other fields of the item. If you store media items as files, you must synchronize filesystems between Sitecore solutions when you synchronize item metadata in the media library. If you must use file-based media, consider integration with external media management solutions and Content Delivery Networks (CDNs).

Use validation, a workflow action, the File Drop Area field type, a processor in the `publishItem` pipeline, or other techniques to ensure that users publish media before publishing any content that references that media. For an example that uses validation to assist users to publish referenced media items, see my blog post at `http://bit.ly/xVdkT0`. For an example of a workflow action that publishes related media, see my blog post at `http://bit.ly/ouU37j`.

Instead of using attributes of the `<img>` element to scale images on the client, use Sitecore features to resize images on the server before transmission to the client. Optimize each image down to the maximum size and quality required for any use of that image before uploading it to the media library. Sitecore can always scale images down without distorting the image, but you may notice quality issues if you attempt to scale images up.

Prevent media item URLs from beginning with a tilde. Web clients such as browsers interpret URLs with a leading tilde relative to the page that contains the link to the media item. The disadvantages of a leading tilde in media URLs include the following:

- When multiple pages link to a single media item, search engines see multiple URLs for that media item, which can affect search index ranking. You can resolve this issue by prefixing media URLs with a slash (/).
- When working with deep information architectures, relative URLs determined for paths that begin with a tilde can exceed path length restrictions imposed by the Microsoft Windows operating system and hence Internet Information Services (IIS). You can resolve this issue by prefixing media URLs with a slash (/).
- IIS may resolve other symbols more efficiently than the tilde. You cannot resolve this issue by prefixing media URLs with a slash (/).

For an example of a partial solution that uses the `renderField` pipeline to prepend slashes to media URLs, see the MediaUrlTransformer `http://bit.ly/sBb2m2` Sitecore Shared Source project. For a more complete solution, use an alternate prefix in media URLs. For example, to use `/-/media` as the media URL prefix:

1. Set the `Media.MediaLinkPrefix` setting in the Web.config file to `/-/media`:

```
<setting name="Media.MediaLinkPrefix" value="/-/media"/>
```

2. Insert a `<prefix>` element with `/-/media` in the `value` attribute under the `/configuration/sitecore/mediaLibrary/mediaPrefixes` element in the Web.config file:

```
<prefix value="/-/media" />
```

www.it-ebooks.info

3. Insert a `<handler>` element with `-/media/` as the value for the `trigger` attribute and `sitecore_media.ashx` as the value for the `handler` attribute under the `/configuration/sitecore/customHandlers` element in the Web.config file:

```
<handler trigger="-/media/" handler="sitecore_media.ashx"/>
```

The media upload watcher creates media folders and media items in the Sitecore media library based on subdirectories and files that you create in the subdirectory specified by the `MediaFolder` setting in the Web.config file, which defaults to the `/upload` subdirectory of the IIS document root subdirectory hosting the Sitecore solution. The media upload watcher is irrelevant in content delivery environments and in content management environments that do not use this feature. If you do not use the media upload watcher, you can disable it in the content delivery environment, the content management environment, or both. To disable the media upload watcher, comment the `/configuration/system.webServer/modules/add` and `/configuration/system.web/httpModules/add` elements named `SitecoreUploadWatcher` in the `/web.config` file.



Because these sections are not within the `/configuration/sitecore` *element, you must make this change in the actual* `/web.config` *file rather than using a Web.config include file.*

If you do not disable the media upload watcher in the content management environment, you can control the types of files that the watcher monitors, and you can prevent the watcher from monitoring subdirectories or files that contain specific character sequences. To limit the types of files that the media upload watcher monitors:

1. Comment the `/configuration/sitecore/watchers/media/filter` element with a value of `*` in the Web.config file.
2. Add `/configuration/sitecore/watchers/media/filter` elements to the Web.config file for each filename extension the watcher should monitor. For example, use the following to monitor only for `.png` and `.jpg` files:

```
<watchers>
  <media>
    <folder ref="settings/setting[@name='MediaFolder']/@value" />
<!--<filter>*</filter>-->
    <filter>*.png</filter>
    <filter>*.jpg</filter>
  </media>
...
```

To prevent the media upload watcher from monitoring files with names that contain specific patterns:

1. Add a `/configuration/sitecore/mediaLibrary/watcher/ignoreList/ignore` element to the Web.config file.
2. Set the `contains` attribute of the new element to the character pattern to ignore.

www.it-ebooks.info

To prevent the media upload watcher from monitoring subdirectories with names that contain specific patterns:

1. Add a `/configuration/sitecore/mediaLibrary/watcher/ignoreList/ignorepath` element to the Web.config file.
2. Set the `contains` attribute of the new element to the character pattern to ignore.

Maximizing Solution Performance

Do not implement presentation or other components that process a large number of items in the foreground. If necessary, such as for certain administrative functions, process a large number of items in a background process such as a scheduled agent. For more information about scheduling background tasks, see my blog post at `http://bit.ly/ncXS3p`.

When possible, use search engines, Sitecore query, or other techniques to avoid iterating large branches of items, including any constructs that involve the descendant axis, including the following:

- The `descendant` axis (abbreviated as `//`, for example `.//*` or `.//item`)
- The `descendant-or-self` axis
- The `sc:descendants()` XSL extension method
- The `GetDescendant()` and `GetDescendants()` methods of the `Sitecore.Data.Items.ItemAxes` class, such as the instance exposed by the `Axes` property of the `Sitecore.Data.Items.Item` class

If you must perform a long-running task in the foreground, such as to iterate all items, update the user interface periodically to indicate the progress of that process. Implement coding techniques to prevent background processes from consuming inordinate resources.

As described in Chapter 6, monitor and tune cache sizes over time. Enable output caching for each rendering, especially those that consume significant system resources such as CPU or memory. If you cache the output of a parent control, such as a sublayout, do not cache the output of descendant controls, such as renderings nested within that sublayout. Vary caching by the fewest possible criteria, most commonly the data source item of each presentation component. For maximum flexibility, configure output caching for individual renderings; for maximal reuse, configure output caching for the parent sublayout. Unless the number of concurrent users is relatively low and relative to the amount of available memory, avoid caching output at the user level.

As described in Chapter 6, Sitecore CMS version 6.4.1 rev. 110928, released subsequent to Sitecore CMS 6.5.0 rev. 110818 described in this book, introduces the `Caching.DisableCacheSizeLimits` setting to the Web.config file. If this setting is `true`, Sitecore disables cache size limits, meaning that caches can grow indefinitely. With modern systems, memory is often less constrained than it was in the past. You can set `Caching.DisableCacheSizeLimits` to `true` to see how large caches would grow without limits. If the solution does not approach memory capacity limitations with this configuration, you may be able to leave `Caching.DisableCacheSizeLimits` set to `true`.

www.it-ebooks.info

Enable compression of static and dynamically generated content in IIS Manager (`inetmgr`) for the IIS website hosting your Sitecore solution. Also in IIS Manager, enable content expiration headers, particularly for the `/sitecore` subdirectory and the `/temp/IconCache` subdirectory (or the directory specified by the `Icons.CacheFolder` setting in the Web.config file) under the IIS document root hosting your Sitecore solution. This can improve performance in the content management environment. Specific instructions to enable these options depend on the version of IIS that you use; to change these options, consult the documentation for your version of Windows.

SITECORE TIPS AND TRICKS

This section provides a number of tips and tricks for working with the Sitecore ASP.NET CMS. These suggestions can maximize your productivity when working with the CMS, as well as increase its usability for end users.

Optimizing Sitecore Usability

Sitecore takes usability very seriously, but you are ultimately responsible for the usability of your solutions. Usability is critical to the success of each CMS implementation. The degree to which CMS users access the system to create and maintain content is one of the factors that drives traffic to the site and hence delivers ROI from the solution.

To maximize consistency, familiarity, and usability, Sitecore follows Microsoft Windows and Office user interface (UI) conventions. Features in Sitecore user interfaces, such as the Content Editor and the Sitecore desktop, including those that you can implement by extending Sitecore, mimic aspects of the Windows and Office user interfaces. Many components of the Sitecore user interfaces are reusable, which maximizes consistency and minimizes not just development effort but also the time required to conceptualize the UI before implementation. The Sitecore UI framework also helps you to provide a uniform user experience — you generally already know how CMS UI components that you need to develop should look and function, and the framework facilitates those objectives.

Optimally, CMS users should see their ability to control content as an opportunity rather than a responsibility. Content management systems can be an enabling technology for rapid, easy, and automated deployment of web assets to the managed websites. While Sitecore provides user training and documentation for the platform and you often must develop resources specific to your solution for your CMS users, as a Sitecore developer, one of your goals should be to make the solution as intuitive and discoverable as possible. Provide as much relevant information as possible in context without overwhelming the user with details, and make additional information available to users through simple operations such as clicking or even hovering the mouse pointer over an item.

Consider usability in all Sitecore interfaces, but especially the Page Editor and the Content Editor. The Content Editor is more appropriate for certain types of operations, such as those that affect the information architecture, including sorting, copying, and moving items, as well as dragging and dropping items.

Performance can directly impact the usability of a web application. Along with striving for optimal system configuration, ensure that any extensions that you develop do not significantly affect performance.

www.it-ebooks.info

Configure all web clients, especially Microsoft Internet Explorer, according to The Sitecore Browser Configuration Reference (`http://bit.ly/t2KPxX`). For example, some CMS features may not work unless you enable pop-up windows in the browser for the root URL of the content management environment. Remember to add each CMS instance to Internet Explorer's trusted sites.

Finally, whenever they are not in use, hide the standard fields and raw field values, and after selecting any other database in the Sitecore desktop, revert to the Master database.

Logging In to a Sitecore User Interface

As explained in Chapter 1, at the Sitecore login screen, you can click Options to show the login options. When the login options appear, you can click a user interface, enter your credentials, and then click Login or press Enter to log in to that user interface. Your choices for user interfaces include the Page Editor (the default), the Content Editor, and the desktop. Alternatively, you can enter your credentials and then double-click a user interface to log in to that UI.

If you forget to click Options to show the Options panel in the login screen and select a user interface before you log in, and then enter your credentials and click Login or press Enter, Sitecore logs you in to the user interface selected on that hidden Options panel, which may not be the user interface you intended to access. If you realize you forgot to select an option before that user interface appears, and if you are fast enough, you can click Options and then double-click the interface you intended. Otherwise, log out, click Options, select a user interface, and then log in again.

To cause the Sitecore login screen to display the Options panel by default, reminding you to select a user interface before you log in, remove the `style` attribute of the `<div>` element with a value of `OptionBox` for the `id` attribute from the `/sitecore/login/default.aspx` file. If a user clicks Options in the login screen to hide the Options panel, and then fails to authenticate, the Options panel reappears. To prevent the user from clicking Options, remove or comment the `<a>` element with an `onclick` attribute that invokes the `scToggleOptions()` JavaScript function from above that `<div>` element with a value of `OptionBox` for the `id` attribute in the `/sitecore/login/default.aspx` file. It would be preferable, but more complicated, to add a base template containing fields controlling login page options to the `/sitecore/content/Applications/System/Login` item in the Core database. To customize the login process further, you can override the code-behind specified in the `/sitecore/login/default.aspx` file.

You can make the browser use all available screen space, typically by pressing F11. You can get even more space by hiding the status bar, address bar, and other aspects of the browser user interface, which can be useful on equipment with limited resolution.

If you are in a user interface such as the Page Editor or the Sitecore browser-based debugger and you wish to access the desktop, you can log out and log in, or you can change the path in the browser's address bar to `/sitecore/`, leaving the root of the URL that already appears in the address bar. For example, remove everything except `http://sitecorebook` and add`/sitecore/shell`, resulting in `http://sitecorebook/sitecore/shell`.

Limiting User Interfaces

As described previously, you can use the Sitecore client roles to restrict access to CMS features, which increases usability by simplifying the user interfaces. To avoid overwhelming CMS users by exposing unnecessary features, encourage them to access the Page Editor rather than

www.it-ebooks.info

the Content Editor, and when necessary to achieve an objective, the Content Editor rather than the Sitecore desktop. You can force users into specific user interfaces regardless of their user interface selection in the Options panel on the Sitecore login screen, and you can restrict users to specific interfaces even if they enter the URL of another interface into the browser's address bar.

Use the following steps to cause users to log in to a specific user interface regardless of their selection on the Sitecore login screen:

1. Log in to the Sitecore browser-based desktop as a user with appropriate access rights.
2. Click Sitecore ➪ Security Tools ➪ User Manager. The User Manager dialog appears.
3. Double-click the user. The Edit User dialog appears.
4. Click the Profile tab. Figure 11-1 shows an example of how the Edit User dialog appears after you click the Profile tab.

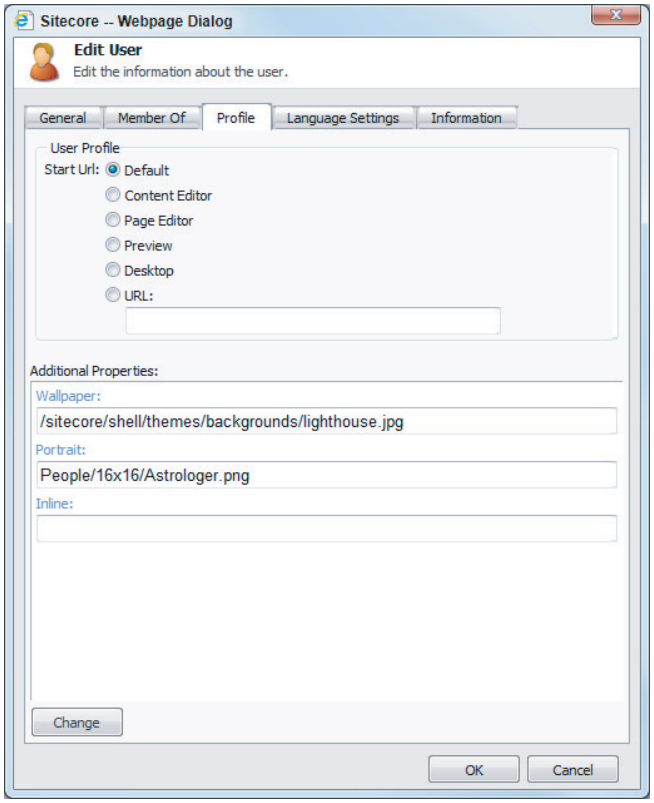


FIGURE 11-1

www.it-ebooks.info

5. In the radio group under Start Url, select the user interface that you want to apply for the user, such as Page Editor.
6. Click OK. When the User Manager appears, close it to return to the Sitecore desktop.

To control which interfaces a CMS user can access (even if they know the URL of the desktop or the Content Editor), you can limit access rights for the `/sitecore/content/Applications/Content Editor`, `/sitecore/content/Applications/Desktop`, and `/sitecore/content/Applications/WebEdit` items in the Core database.



Versions of Sitecore CMS prior to 6 included a user interface named WebEdit that provided functionality similar to that of the Page Editor introduced in Sitecore CMS 6.0. Certain aspects of Sitecore, such as the `/sitecore/content/Applications/WebEdit` *item in the Core database and any reference to content markers ("dots"), continue to reflect WebEdit terminology.*

For example, if you want to prevent a user from accessing the Sitecore desktop, you can create a role in the Sitecore domain, for example named Sitecore Client Desktop Restricted Users, deny that role `item:read` access to the `/sitecore/content/Applications/Desktop` item in the Core database, and add that user to that role.



If users cannot access the Sitecore desktop, they cannot access the Control Panel to configure their language, region, and other preferences. An administrator can accomplish these tasks through the User Manager.

Optimizing the Sitecore Desktop Experience

Each user can change the Sitecore desktop background image — for example, to help them differentiate test and production systems. To change the background image:

1. Right-click the desktop, and then click Properties. The Desktop Background dialog appears as shown in Figure 11-2.
2. Select a background in the Background field.

To make additional background images available, add them to the `/sitecore/shell/Themes/Backgrounds` subdirectory beneath the document root of the IIS website hosting the Sitecore instance. For an example that applies a random desktop background each time you log in to the Sitecore desktop, see my blog post at `http://bit.ly/cbHoUs`.

www.it-ebooks.info

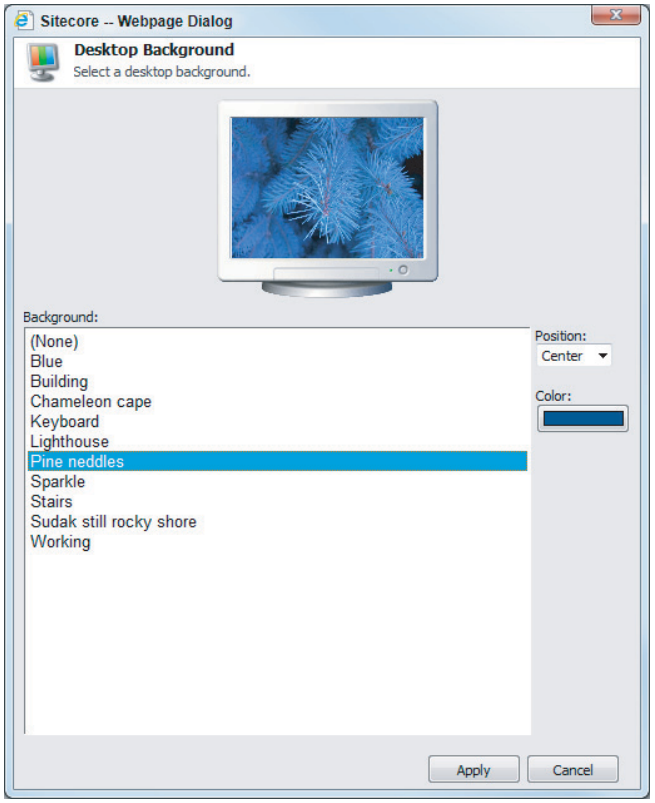


FIGURE 11-2

When you enter text in the search box in the lower right-corner of the taskbar in the Sitecore desktop and press Enter, a search results dialog appears. If you pause before pressing Enter in that process, a list of the matches for the search term that you entered appears automatically. If you enter a path or ID in the search box instead of a search term, you can open the specified item in the Content Editor by clicking the first entry in that list of matches (the one labeled Direct Hit). Figure 11-3 shows use of this feature to find the `/sitecore/media library/Images` item by path.

You can use the following keyboard shortcuts within the Sitecore desktop:

- **Ctrl+/** — Gives the search box in the lower-right corner cursor focus
- **Ctrl+F2** — Displays the window manager (then click a window to select it)



Unfortunately, an apparent defect in some versions of Sitecore CMS prevents the Ctrl+F2 keyboard shortcut from working in some browsers, including Microsoft Internet Explorer 9.

- **Ctrl+WIN** — Activates the Sitecore menu (same as clicking the Sitecore button)

www.it-ebooks.info

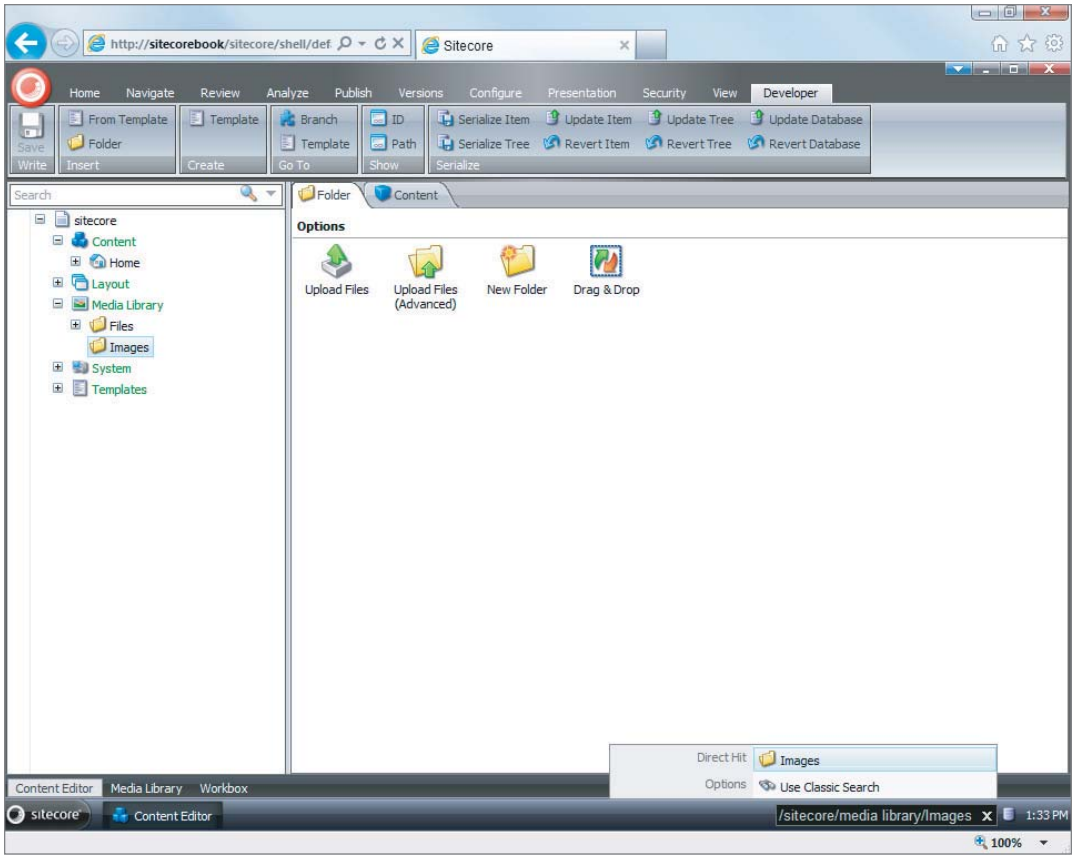


FIGURE 11-3

Ctrl+Right-Click

Sitecore prevents the browser from displaying its standard context menu when you right-click in various user interfaces. In some cases, you can hold the Ctrl key (or the Alt key, or some combination of keys) while you right-click to access the browser's context menu.



To activate features for the correct component, be sure to right-click the frame on which to operate, such as the Content Editor or a specific frame within the Content Editor within the Sitecore desktop.

For example, if you do not own a lock on an item, then Rich Text fields appear disabled (grayed out). You can scroll and select text in such fields, but if you right-click a selection, no context menu appears to let you copy that text to the operating system clipboard. If you hold the Ctrl key while you right-click, you can use the context menu to copy the text.

www.it-ebooks.info

You can also hold Ctrl and right-click to view the URL and source code of a frame, or to refresh a frame (such as to refresh a single application within the desktop without losing the context of other applications open in the Sitecore desktop).



This technique does not work for modal dialogs. To view the source of a modal dialog, press Ctrl+Shift+Alt+Scroll Lock.

Optimizing the Content Editor

This section describes the surprising number of mouse and keyboard shortcuts that the Content Editor supports.

To expand or collapse a section in the editing pane of the Content Editor, double-click the section title, or click the plus (+) or minus (-) icon in the right corner of the title.

To show or hide features in the Quick Action bar to the left of the content tree in the Content Editor, right-click the Quick Action bar, and then select the features to display.

Finding a section or field in an item based on a data template that contains a large number of sections and fields, or when showing the sections and fields defined in the standard template, can take some effort. You can use the drop-down that appears when you click the Ribbon icon to the left of the language selector (to the left of the version selector at the top of the editing pane) to navigate to a specific field more easily. Figure 11-4 shows use of this feature to navigate to the Cacheable field in the Caching section of an XSL rendering definition item.

To obtain more screen space for editing, clear the Content Tree check box in the View group of the View tab in the Content Editor.

You can search for a command by hovering over the Ribbon and using the mouse wheel to scroll through the tabs. Click the command to perform the desired action. To scroll through a subcomponent such as the content tree or the editing pane without scrolling the main Content Editor window, hover the mouse cursor over that component and use the mouse wheel.

To sort one sibling item before another, hold the Alt key and drag the item onto that sibling. Depending on your access rights, you can drag and drop items in the content tree. You can hold down the Ctrl key while dragging an item to copy that item instead of move it.

Hide user interface components in the Content Editor when you are not using features that depend on those components. Subcomponents that you can hide include the following:

- **Pages bar** — Tabs at the bottom of the Content Editor
- **Quick Action bar** — Shortcuts to the left of the content tree
- **Validator bar** — Validation features to the right of the editing pane
- **Quick Info section** — Information above the sections in the editing pane
- **Item Title bar** — Icon, name, and short description above the editing pane

www.it-ebooks.info

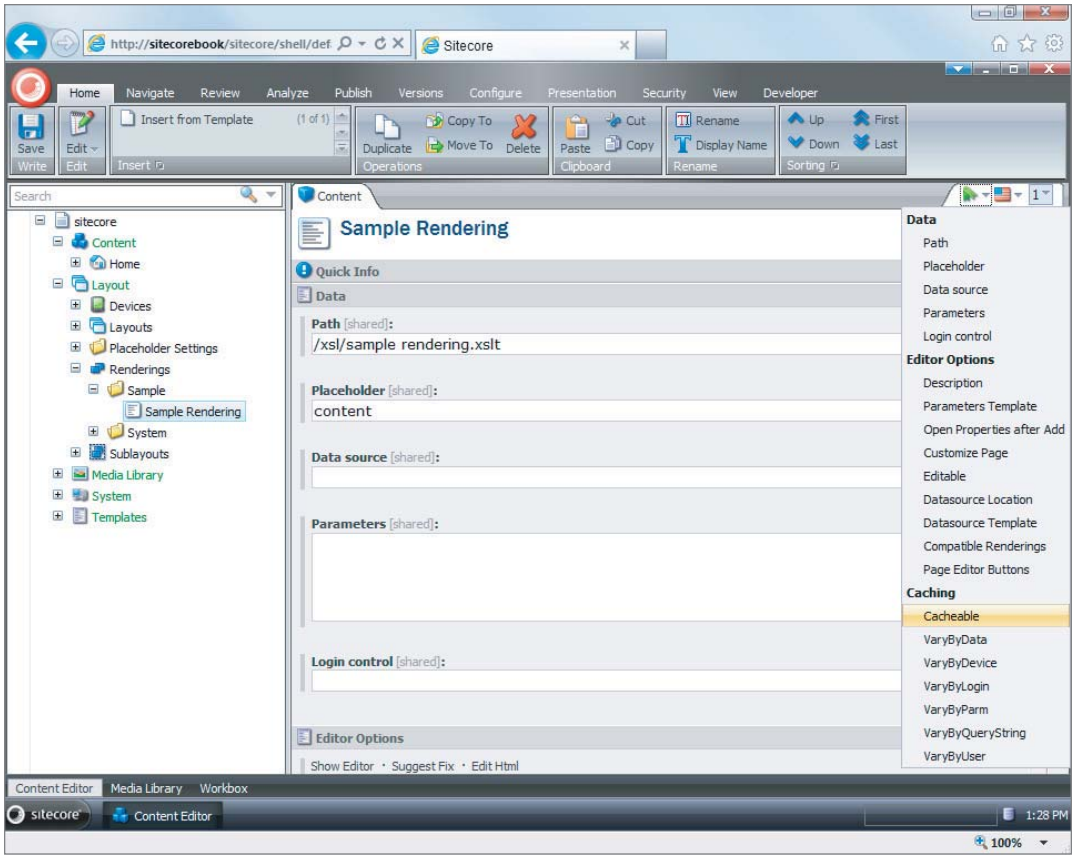


FIGURE 11-4

To hide features in the Content Editor:

1. Click the Sitecore logo at the top-left corner, and then click Application Options. The Application Options dialog appears as shown in Figure 11-5.
2. Click the Content Editor tab.
3. Clear check boxes to disable user interface features.

To further improve Content Editor performance, you can also disable prefetching of collapsed sections to load sections only when you expand them in the user interface (and remember to keep sections collapsed when not in use) using the options shown in Figure 11-5. You can hide field section titles to merge all sections into a single pane, which can be helpful with limited screen space. You should hide any other features that you do not use. For example, if you show the Quick Action bar, you can right-click it to select features to show or hide. To hide the Quick Action bar, the Validator bar, and the search box by default, set the `ContentEditor.ShowGutter`, `ContentEditor.ShowSearchPanel`, and `ContentEditor.ShowValidatorBar` settings in the Web.config file to `false`.

www.it-ebooks.info

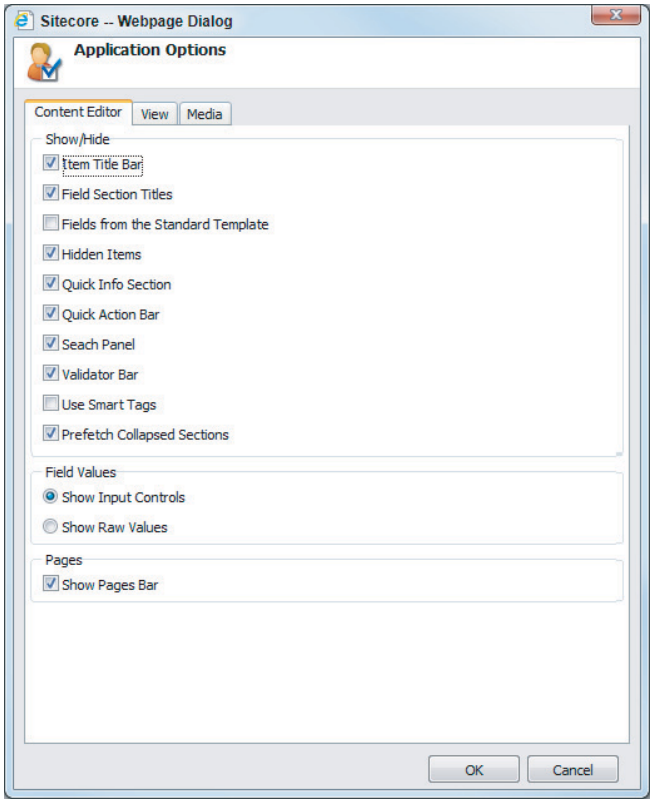


FIGURE 11-5

In addition to creating an item using insert options or an arbitrary data template, branch template, or command template, you can create an item by duplicating an existing item, and then updating field values in the new item. This technique is most useful when any of the following are true:

- You need to create an item of the same type as an existing item, but you do not know the data template associated with that item.
- The path to the data template associated with the existing item is long and therefore difficult to remember or select in the user interface.
- The new item will contain many of the same values for fields as the existing item, for example a web control definition item.

Working with the search box within the Content Editor is similar to using the search box in the Sitecore desktop. You can enter a search term, the ID of an item, or the path to an item in the search box above the content tree, and then either click the magnifying glass icon next to the search box or press Enter to obtain search results. When you click the magnifying glass or press Enter, the content tree displays search results for the value that you entered. If the list includes a Direct Hit, click it to navigate directly to the item that you specified. To view the portion of the content tree obscured by the search results panel, click the X icon in the Search Results label above the panel.

www.it-ebooks.info

Maximizing Content Editor Performance

You can use the following techniques to optimize the performance of the Content Editor, especially for users accessing Sitecore over slow network connections:

- Use the Sitecore client roles to minimize features visible to users.
- Ensure proper browser configuration according to The Sitecore Browser Configuration Reference (`http://bit.ly/qfwYUW`).
- Improve network conditions between the client and the server.
- Consider the Page Editor as an alternative to the Content Editor.
- Upgrade to the latest recommended release of Sitecore.
- Patch Microsoft Windows on your servers, particularly IIS and .NET. Patch client operating systems as well.
- Organize fields into sections, avoiding dozens of fields per section.
- Organize data into hierarchies, avoiding dozens of fields in a single data template or hundreds of items under a single parent.
- Hide the fields defined by the standard template (clear the Standard Fields check box in the View group of the View tab).
- Hide unnecessary tabs by right-clicking the tab strip and clearing the checkmarks for each tab that you do not use frequently.
- To validate content, rather than using the Quick Action bar or the Validator bar in the Content Editor, use the `Sitecore.Workflows.Simple.ValidatorsAction` workflow action and the Validate command in the Proofing group on the Review tab of the Ribbon. Right-click the Quick Action bar and use the context menu that appears to disable validation when you are not reviewing validation issues (you can use this technique to disable additional features of the Quick Action bar to improve performance).
- Set the `ContentEditor.CheckSecurityOnTreeNodes` and/or `ContentEditor.CheckHasChildrenOnTreeNodes` settings in the Web.config file to `false`.
- Avoid security options such as SSL (Secure Sockets Layer) and Windows authentication, which can adversely affect client performance.
- Use field security to show or hide fields irrelevant to specific roles.
- Double-click a tab to show or hide the Ribbon.

For more information about optimizing performance in the Content Editor, see my blog post at `http://bit.ly/q9jgkI`.

Content Editor Keyboard Shortcuts

The Content Editor supports the following keyboard shortcuts:

- **Alt+F1** — Shows or hides keyboard shortcuts for the current tab
- **Ctrl+S** — Saves the current item

www.it-ebooks.info

- **Ctrl+D** — Duplicates the current item
- **F2** — Renames the current item
- **F7** — Subscribes to an RSS feed about the current item
- **Ctrl+Shift+Alt+Up Arrow** — Moves the current item after its previous sibling in the sort order
- **Ctrl+Shift+Alt+Down Arrow** — Moves the current item after its next sibling in the sort order
- **Ctrl+Shift+Alt+L** — Protects (makes read-only) or unprotects the current item
- **Ctrl+Shift+Home** — Navigates to the home item
- **Left Arrow** — Collapses the branch
- **Right Arrow** — Expands the branch
- **Ctrl+Shift+Alt+R** — Shows or hides raw field values
- **Ctrl+Shift+Alt+T** — Shows or hides fields from the standard template
- **Alt+H** — Selects the Home tab
- **Alt+N** — Selects the Navigate tab
- **Alt+R** — Selects the Review tab
- **Alt+P** — Selects the Publish tab
- **Alt+V** — Selects the Versions tab
- **Alt+C** — Selects the Configure tab
- **Alt+E** — Selects the Presentation tab
- **Alt+S** — Selects the Security tab
- **Alt+I** — Selects the View tab



To see the shortcut for a tab or command, hover the mouse cursor over it in the Content Editor.

Investigating and Copying Raw Field Values

In some cases, viewing raw field values can help you explore and understand the system. For example, while writing this book, I did not know where Sitecore stores conditional rule definitions that apply at the rendering level. I could have tried to find the answer in the documentation, or by disassembling Sitecore, but the approach described in the following paragraph was relatively efficient for my purposes at that time.

Knowing that layout details indicate which presentation controls to use when rendering an item, I assumed that layout details would also reference the conditional rendering rules to apply to each

www.it-ebooks.info

presentation control. In the layout details for an item, I selected a presentation control, and then selected one of the default conditional rendering rules to apply to that rendering. I then selected options on the View tab in the Ribbon to show the fields from the standard template and raw values for all fields before investigating the value of the Renderings field in the Layout section, which contains layout details. I copied the ID of the conditional rendering rule definition item from that field value, pasted that ID into the search box above the content tree, and pressed Enter. I then cleared options on the View tab in order to hide the fields defined in the standard template and show usable controls instead of raw field values.

To copy a complex field value from one item to another, view the standard fields (if required), view raw field values, copy the field value to the operating system clipboard, and paste that value into the target field. Remember to hide standard fields (if showing) and raw values afterwards.

Copying and Moving Items with the Clipboard

You can use the operating system clipboard to copy and move items in the content tree. To copy or move an item and its descendants, right-click the item, then click Copy (to duplicate the item) or Cut (to move it) from the context menu that appears. If you paste the value from the clipboard to a text editor, you will see a value such as `sitecore:cut:{110D559F-DEA5-42EA-9C1C-8A5DF7E70EF9}`. To paste the item and its descendants, right-click the destination item, and then click Paste.

Sitecore Keyboard Shortcuts

In addition to the default keyboard shortcuts defined by Sitecore and applications such as the Content Editor, you can assign keyboard shortcuts to Sitecore commands. Sitecore uses a text string to identify each keystroke combination, such as `sca76` for Alt+Ctrl+Shift+L (where s stands for Shift, c stands for Ctrl, a stands for Alt, and 76 represents a capital L). To determine the text code for a keystroke combination, you can use the Keyboard Map application on the Development Tools menu in the Sitecore desktop.

Before you assign a keyboard shortcut to a command, attempt to confirm that Sitecore itself does not already use that keystroke combination. The example shown in Listing 11-2 generates an unordered list of the existing items that contain a value in the field named KeyCode used to associate a keyboard code with a command.



LISTING 11-2: Listing keyboard shortcuts

```
namespace SitecoreBook.Web.UI.WebControls
{
  using System.Web.UI;

  public class KeyboardShortcuts : Sitecore.Web.UI.WebControl
  {
    protected override void DoRender(HtmlTextWriter output)
    {
      Sitecore.Data.Database db =
        Sitecore.Configuration.Factory.GetDatabase("core");
      Sitecore.Diagnostics.Assert.IsNotNull(db, "core db");
      Sitecore.Data.Items.Item[] items = db.SelectItems(
```

www.it-ebooks.info

```
        "/sitecore/content/Applications//*[@KeyCode!='']");

      if (items == null || items.Length < 1)
      {
        return;
      }

      output.WriteLine("<ul>");

      foreach (Sitecore.Data.Items.Item item in items)
      {
        string code = item["KeyCode"];
        string message = item.Paths.FullPath
          + " : "
          + code
          + "("
          + Sitecore.Web.UI.Keyboard.GetKeyCodeText(code)
          + ")";
        output.WriteLine(
          "<li>" + message + "</li>");
      }

      output.WriteLine("</ul>");
    }
  }
}
```



I used a web control for this example because it is easy for me to implement and test by binding such a web control to a placeholder in the home item of a development environment. This approach leaves me with a library of web controls that I can easily duplicate and use at my convenience. Alternatively, you can invoke code such as this from any context that you find convenient.

After you identify a keystroke combination not already in use, enter the text code for that combination into the KeyCode field of the definition item for that command in the Core database. To specify a keyboard shortcut for a command in the Content Editor Ribbon:

1. Click the database icon in the lower-right corner of the Sitecore desktop, and then select core from the context menu so that you can edit items in the Core database.
2. Open the Content Editor.
3. Navigate to the command definition item, which is a descendant of the `/sitecore/content/Applications/Content Editor/Ribbons/Chunks` item.
4. Set the value of the KeyCode field in the Data section to the text code for the keystroke combination.
5. Click the database icon in the lower-right corner of the Sitecore desktop, and then select master from the context menu to return to the Master database.

www.it-ebooks.info

To create a keyboard shortcut for a command not in the Content Editor Ribbon:

1. Click the database icon in the lower-right corner of the Sitecore desktop, and then select core from the context menu so that you can edit items in the Core database.
2. Open the Content Editor.
3. Navigate to the `/sitecore/system/Settings/Key Map` item.
4. Insert an item using the `Sitecore Client/Key Map` data template.
5. Set the value of the KeyCode field in the Data section to the text code for the keystroke combination.
6. Set the value of the Click field in the Data section to the command to execute (from the `/App_Config/commands.config` file).
7. Click the database icon in the lower right corner of the Sitecore desktop, and then select master from the context menu to return to the Master database.

Optimizing the Rich Text Editor

The Rich Text data template field type exposes a what you see is what you get (WYSIWYG) HTML editor in a Sitecore data template. The Rich Text Editor (RTE) can give users complete control over the presentation of their content. In general, subject matter experts (CMS users) should focus on the text of their content rather than the presentation and styling of that content. Visual consistency leads to usability, usability leads to repeat visitors, and repeat visitors lead to solution value. Excessive use of and features in the RTE can reduce the visual consistency of the solution for which you strive. Additionally, by storing data as HTML, to a greater or lesser extent depending on how you use them and what you allow in them, WYSIWYG HTML editors defeat one of the primary goals of web content management systems, which is to separate content from presentation.

To maximize consistency and encourage CMS users to focus on the subject matter of their expertise rather than markup, styling, and page structure, minimize the use of the Rich Text data template field type, and configure RTE profiles to minimize available features. Use presentation components to structure and style the content. Use the RTE field type sparingly, and expose only those features required by the individual users who need them.



Click the box icon in the upper-right corner of the Rich Text Editor to expand the RTE to use the full browser window. As always, you can press F11 to cause the browser to use all available screen space.

For more information about configuring and optimizing the Rich Text Editor, see the Sitecore Client Configuration Cookbook (`http://bit.ly/qS8Dc1`).

www.it-ebooks.info

Rich Text Editor Profiles

Rich Text Editor (RTE) profiles control the commands visible in each Rich Text field. You can use the Source property in Rich Text field definition items to specify the RTE profile to use for that field.

Sitecore manages RTE profiles under the `/sitecore/system/Settings/Html Editor Profiles` item in the Core database. The default RTE profile (`/sitecore/system/Settings/Html Editor Profiles/Rich Text Default`) allows relatively few features.

You can apply access rights to items in RTE profiles to control which users can access each feature. To maximize site consistency, minimize features visible in Rich Text fields.



You can use clones to manage RTE profiles. For more information about clones, see Chapter 2.

Rich Text Editor Keyboard Shortcuts

Rich Text Editor keyboard shortcuts include the following:

- **Ctrl+A** — Selects all
- **Ctrl+B** — Applies bold formatting (`<strong>`)
- **Ctrl+C** — Copies the selection in the RTE to the operating system clipboard
- **Ctrl+F** — Activates the browser's find feature to search for text in the RTE field value
- **Ctrl+I** — Applies emphasis/italics (`<em>`)
- **Ctrl+K** — Inserts or updates an external link
- **Ctrl+Shift+K** — Removes a link
- **Ctrl+P** — Prints
- **Ctrl+U** — Underlines (`<span>`)
- **Ctrl+V** — Pastes from the operating system clipboard into the RTE field
- **Ctrl+X** — Cuts the selected text (moves it to the clipboard)
- **Ctrl+Y** — Allows unlimited redo
- **Ctrl+Z** — Un-does an action
- **Shift+Enter** — Enters a line break (`<br>`)
- **F10** — Moves focus to the first toolbar (then use Tab and Shift+Tab to move between tools on the toolbar, the arrow keys to activate drop-downs, and the Enter key to make a selection)

Common keyboard shortcuts including Home, Ctrl+Home, Ctrl+Shift+End, and others typically work as expected. You can use the `/Sitecore/shell/controls/Rich Text Editor/ToolsFile.xml` file to configure the Rich Text Editor, including keyboard shortcuts. As described in The

www.it-ebooks.info

Sitecore Client Configuration Cookbook (`http://bit.ly/vz8fnc`), you can use predefined markup snippets to allow users to insert reusable text into RTE fields easily.

Ad Hoc Reports

To easily implement ad hoc reports, you can use a temporary XSL rendering (or save the rendering for reuse later) or Sitecore Rocks. For example, viewing the raw values for the `__Base template` field in the standard template, which defines the base templates for a data templates, provided the IDs to construct the XSL fragment shown here, which I then used to identify data templates that inherit from the null template (with ID `{00000000-0000-0000-0000-000000000000}`) or from only specific sections of the standard template.




```
<xsl:for-each select="//item[contains(sc:fld('__Base template',.),
  '{646F4B34-708C-41C2-9F4B-2661849777F3}')
 or contains (sc:fld('__Base template',.), '{F5BC7AE4-F62D-47CD-9AFB-2C9F85D8313B}')
 or contains (sc:fld('__Base template',.), '{18420296-D8FC-4D93-A79E-2A2B0367A92C}')
 or contains (sc:fld('__Base template',.), '{823ECF5F-AE72-40B5-BC87-CB425FE7E5F6}')
 or contains (sc:fld('__Base template',.), '{4D30906D-0B49-4FA7-969D-BF90157357EA}')
 or contains (sc:fld('__Base template',.), '{6EF07334-7FBA-4845-BF0A-CD5B2000C75A}')
 or contains (sc:fld('__Base template',.), '{6495CF23-DE9C-48B7-9D3C-05E2418B3CAE}')
 or contains (sc:fld('__Base template',.), '{AB811AF4-393C-4A46-ACD7-81D30917E7F4}')
 or contains (sc:fld('__Base template',.), '{1597272C-C823-4AAC-86F8-CA9CC4D573B5}')
 or contains (sc:fld('__Base template',.), '{93F3A8FA-9E5A-4848-A5AD-42AAD11A4871}')
 or contains (sc:fld('__Base template',.), '{2491819E-0C61-4F5A-903C-E10FCD9D302A}')
 or contains (sc:fld('__Base template',.), '{06F366E6-A7E6-470B-9EC9-CD29A4F6C8E8}')
 or contains (sc:fld('__Base template',.), '{00000000-0000-0000-0000-000000000000}')]">
  <strong><xsl:value-of select="sc:path(.)" /></strong>
  <ul>
    <xsl:for-each select="sc:Split('__Base template', .)">
      <li>
        <xsl:variable name="item" select="sc:item(text(),.)" />
        <xsl:choose>
          <xsl:when test="$item">
            <xsl:value-of select="sc:path($item)" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="text()" /> does not exist.
          </xsl:otherwise>
        </xsl:choose>
      </li>
    </xsl:for-each>
  </ul>
</xsl:for-each>
```

Overlay Icons

Sitecore displays an icon for each item. You can set the default icon for all items based on a data template in that data template itself. You can override the icon defined in the data template for each item. To set the icon for a data template or an individual item, click the Configure tab in the Content Editor, and then click the Icon command in the Appearance group. You can either select an icon from the pallet that appears (defined by the `/App_Config/Icons.config` file), or click More Icons at the bottom of that pallet. In the latter case, the Icon dialog appears, allowing you to select an

www.it-ebooks.info

icon from all of those available on the system. You can use the drop-down at the top-left of the Icon dialog to select various categories of icons from which to select an image. Figure 11-6 shows how the Icon dialog appears when making a selection from the category drop-down.

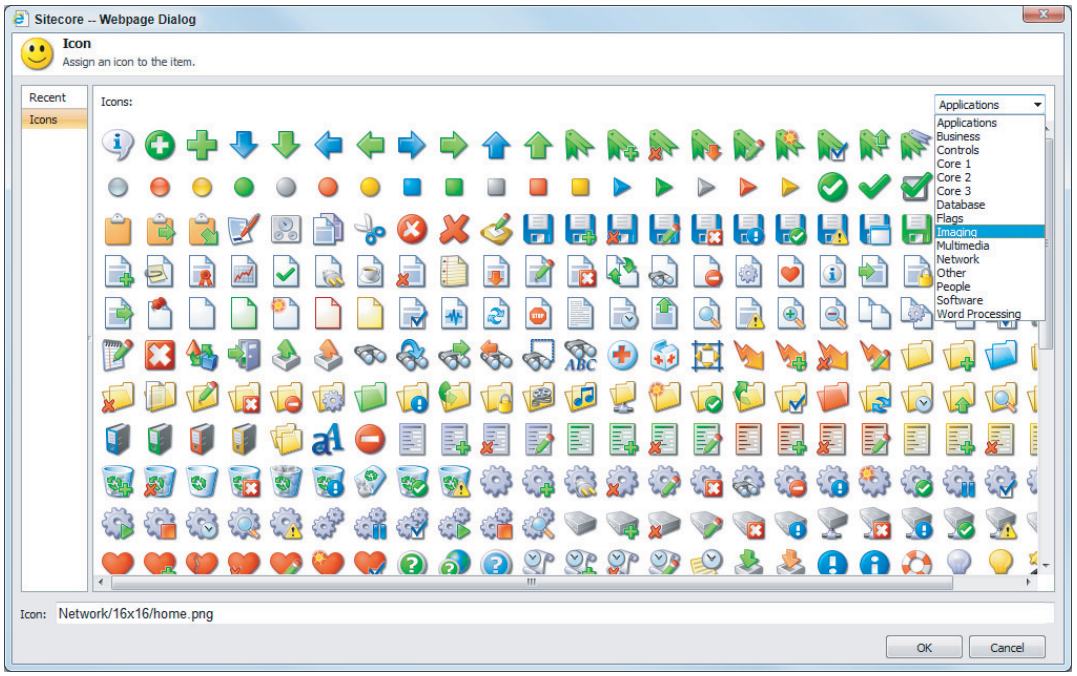


FIGURE 11-6

The Icon field at the bottom of the Icon dialog shows the path to the icon that you select.

You can overlay an icon with another icon in the lower-right corner of the original icon image. To do so, follow this procedure:

1. Determine the relative paths to the two icons that you wish to use by selecting those icons individually in the Icon dialog and noting the values in the Icon field for each selection.
2. Enter a value such as the following into the Icon field in the Icon dialog:

```
Applications/32x32/warning.png?overlay=Applications/16x16/help.png
```

3. Replace the two relative paths shown with those of the icons you wish to use, where the first (`Applications/32x32/warning.png` in this example) specifies the primary icon and the second (`Applications/16x16/help.png` in this example) specifies the overlay icon.

Note the dimensions specified in the relative path to each image — 32x32 for the first image and 16x16 for the second. Use these dimensions regardless of those identified previously using the Icon dialog (Sitecore provides all icon images in both dimensions). For more information about icons, see The Sitecore Client Configuration Cookbook (`http://bit.ly/zG3X6W`).

www.it-ebooks.info

Showing the Developer Tab

The Developer tab provides a number of commands to assist Sitecore developers. To show or hide the Developer tab in the Content Editor, right-click any tab in the Content Editor, and then select or clear the Developer option. The Path and ID commands in the Show group of the Developer tab are particularly helpful for copying the path and ID of the item selected in the content tree to the operating system clipboard, though these commands may only work in certain browsers (specifically Internet Explorer). You can also use the Developer tab to manage Sitecore item serialization as described in Chapter 9.



The My Toolbar tab enables you to group commands that you use frequently. Personally, I move between systems frequently enough that configuration of the My Toolbar tab is generally not worthwhile.

Working with the Sitecore Debugger

Remember to use the browser-based Sitecore debugger, as well as the cache and rendering statistics pages, both described in Chapter 6. These tools can help you to optimize the solution before you deploy it to production and to monitor and diagnose issues with the solution after implementation.

TAKE HOME POINTS

Sitecore is a broad, flexible, and extensible development platform providing almost infinite potential for configuration and expansion. By following the conventions outlined in this chapter, you can dramatically increase the usability, performance, security, and quality of your Sitecore solutions. Using the tips and tricks described, including extensive keyboard shortcuts in multiple applications, you can maximize your productivity as well as that of your CMS users. You should especially focus the quality of experience in the Page Editor and the Content Editor.

Restrict users' access within the system to only those features and capabilities required for their job function. Content should also pass through a publishing workflow to enforce validation, review, and any other quality standards.

www.it-ebooks.info

www.it-ebooks.info

A

Resources for Sitecore Developers

You can maximize your productivity as a Sitecore developer by reviewing a variety of resources on the Internet and by participating in online and in-person Sitecore community events. Review the information in this appendix before installing the Sitecore ASP.NET Content Management System (CMS).

ACCESSING SITECORE RESOURCES

This section provides pointers to various resources for users, administrators, project managers, and most importantly developers working with the Sitecore ASP.NET CMS. It contains information about resources including Sitecore training, Sitecore partners, online materials such as Sitecore documentation and the Sitecore Developer Network forums, local and virtual user groups, the annual Sitecore Symposium conference for developers and partners (formerly Dreamcore), and Sitecore Success Services.



In general, you can use the same log-in information (e-mail address and password) for all the websites that Sitecore manages, including the Sitecore Portal (`http://portal.sitecore.net`*), the Sitecore Developer Network (*`http://sdn.sitecore.net`*), and the Sitecore Support Portal (*`http://support.sitecore.net`*).*

www.it-ebooks.info

Sitecore Training

In addition to being one of the most efficient methods to learn a new platform, one of the best ways to evaluate a web CMS is to attend developer training. I strongly recommend that all Sitecore developers attend as many relevant Sitecore training courses as possible from those listed at `http://bit.ly/vMSXbk`. I specifically recommend that .NET developers complete the following courses in the order listed:

1. **SDF:** Sitecore CMS Developer Foundations (`http://bit.ly/zoIxdH`)
2. **SND:** Sitecore CMS Website .NET Developer (`http://bit.ly/A1Kq0W`)
3. **DOP:** Sitecore CMS .NET Developer Day of Prototyping (`http://bit.ly/yFGThH`)
4. **AND:** Sitecore CMS Advanced .NET Developer (`http://bit.ly/zqHrMY`)

If the platform architecture matches the requirements that you used to analyze the available CMS platforms, and hence to select Sitecore, then you are likely to remember much of the technical detail provided during training. Importantly, you must work with the product immediately after training in order to maintain and enhance that understanding. After training, I recommend that you spend at least one week implementing a prototype of your actual solution. Use that time to see what you can remember, research what you forgot or never learned, participate in the Sitecore community online, and attempt to build a functional solution that you may subsequently discard.

Sitecore Partners

To maximize the value of your solution quickly, Sitecore encourages all of its customers, even those with prior CMS experience, to work with seasoned Sitecore partners. Some Sitecore partners can provide a full range of services from requirements analysis, platform selection, visual design, Search Engine Optimization (SEO), technical implementation, and integrations such as with social media sites. You may also benefit from working with a Sitecore partner that has expertise implementing web solutions for organizations like yours or your specific industry vertical.

The following sections explain how an organization that provides these types of consulting services can become a Sitecore partner and the benefits of that partnership.

Becoming a Sitecore Partner

If your company uses Sitecore to implement web solutions for other organizations, you can almost certainly become a Sitecore partner. To initiate the partner engagement process, prospective Sitecore partners can complete and submit the web form found at `http://bit.ly/vaOYly`. Sitecore Certified Solution Partners must execute a General Terms of Business Agreement with Sitecore and must maintain at least one staff member certified as an ASP.NET developer on the Sitecore platform.

Benefiting from Your Sitecore Partnership

Sitecore partners can realize additional business based on the following aspects of Sitecore as an organization and as a platform:

- Sitecore is a visionary and leader within a growing web CMS space
- Sitecore is a Microsoft Gold-Certified and award-winning ISV (Independent Software Vendor) partner

www.it-ebooks.info

- Sitecore solutions scale from small to very large with great flexibility
- Sitecore is highly committed to its partner program
- Your dedicated Sitecore team supplies proven technical and business expertise

Sitecore Portal

You can use the Sitecore Portal (`http://portal.sitecore.net`) to register for access to the Sitecore Developer Network (`http://sdn.sitecore.net`), update your personal information and your public profile, and access other information about your relationship with Sitecore. As described in the following sections, you can use the portal to configure your subscriptions to Sitecore mailing lists, which keep you informed with vital information about Sitecore products. You can also use the portal to reset the password you use to access the Sitecore Developer Network, the Sitecore Support Portal (`http://support.sitecore.net`), and the Sitecore Partner Portal itself.

Registering for a Sitecore Account

Before you can access the Sitecore Portal and additional Sitecore resources online, you must register an e-mail address to create an account with Sitecore. To register for an account, do the following:

1. Access `http://portal.sitecore.net` in a browser. If Logout appears in the header, click it.
2. Click Sign-Up Here. The Validate Email page appears.
3. Type your e-mail address into the Email field, and then click Send Me a Validation Mail. The Validation Email Sent page appears. You can close the browser window.
4. Check your e-mail for that address for a message from `portal@sitecore.net`. In that message, click the link to validate your e-mail address. The Choose Password page appears in a browser window.
5. Type the required information about yourself into the various fields, including a password in the Password and Password (Repeat) fields. Then click Complete Registration. The Registration Completed page appears. You can close the browser window.

Signing Up for E-Mail Alerts

I recommend that you visit the Sitecore Support Portal to sign up for available mailing lists. You can receive newsletters intended for users, administrators, and developers regarding new Sitecore products, updates, development practices, as well as other useful information. Sitecore does not send excessive e-mail, and its newsletters can contain alerts that address technical, security, and various other issues concerning Sitecore and supporting products, including issues specific to third-party web browsers.

Resetting Your Sitecore Password

You can use the Sitecore Portal to reset the password you use to access the Sitecore Developer Network, the Sitecore Portal, the Sitecore Support Portal, and other Sitecore online resources. To reset your Sitecore password, do the following:

1. Access `http://portal.sitecore.net` in a browser. If Logout appears in the header, click it.

www.it-ebooks.info

2. Click Forgot Password. The Reset Your Password page appears.
3. Enter the e-mail address that you used to register with Sitecore in the Email field, and click Reset Password. The Done page appears. You can close the browser window.
4. Check your e-mail for that address for a message from `portal@sitecore.net`. In that message, click the link to reset your password. The Choose Password page appears in a browser window.
5. Enter a new password in the Password field, and again in the Password (Repeat) field, and click Complete Registration. The Registration Completed page appears. You can close the browser window.

Sitecore Developer Network

The Sitecore Developer Network, or SDN (`http://sdn.sitecore.net`), is the primary source of technical information for Sitecore developers. Its resources include the invaluable Sitecore documentation as well as the Sitecore Developer Network forums described in the following sections.

Sitecore Documentation

Through SDN, Sitecore provides extensive documentation for CMS users and administrators, but most importantly for developers. You can get most of the best documentation by clicking the References tab on SDN and then clicking Sitecore 6. Other documents, such as the installation and troubleshooting resources that appear on the installation materials pages, exist elsewhere on SDN. For a single, comprehensive list of current Sitecore documentation, see my blog post at `http://bit.ly/cbpSQs`. Where appropriate, this book contains direct links to these documents.

Of the Sitecore documents available online, I believe that the following are most important:

- The Installation Guides (`http://bit.ly/pv7X3B`) and Installation Troubleshooting (`http://bit.ly/qmzVpY`) materials, as well as the Browser Configuration Reference (`http://bit.ly/qfwYUW`)
- The Data Definition Reference (`http://bit.ly/nmGuiB`), the Data Definition Cookbook (`http://bit.ly/oc8F9K`), and the Data Definition API Cookbook (`http://bit.ly/ohVWlq`)
- The Content Reference (`http://bit.ly/qd6lUO`), the Content Author's Cookbook (`http://bit.ly/qaSgz5`), the Content Cookbook (`http://bit.ly/rgtLol`), and the Content API Cookbook (`http://bit.ly/oC6wHw`)



Where multiple documents exist on a subject (such as presentation), references contain descriptive information. Cookbooks contain instructions; API cookbooks describe APIs; anything with author *in the title is more for users than for administrators or developers.*

www.it-ebooks.info

- The Sitecore Client Configuration Cookbook (`http://bit.ly/qS8Dc1`)
- The Sitecore Presentation Component Reference (`http://bit.ly/o2TdvV`), the Sitecore Presentation Component Cookbook (`http://bit.ly/nDo0Ek`), and the Sitecore Presentation Component API Cookbook (`http://bit.ly/r76c6K`)
- The Sitecore CMS Performance Tuning Guide (`http://bit.ly/rbujEO`), the Sitecore Cache Configuration Reference (`http://bit.ly/qYIAia`), and the Sitecore Scaling Guide (`http://bit.ly/oKvYuN`)
- The What's New document for each new version (`http://bit.ly/qntlp9`)

You can access documentation packages to download multiple Sitecore documents as a `.zip` file, which you can then extract into a subdirectory and search as a collection. In Adobe Reader press Ctrl+Shft+F or click the Edit menu and select Advanced Search. In the Search window that appears, select All PDF Documents In, select the folder containing the extracted files, enter a search term, and click Search.

Using the Sitecore Developer Network Forums

Along with hosting an active and responsive community, the Sitecore Developer Network forums (`http://bit.ly/vbOyf8`) contain a vast archive of information about Sitecore. I suggest that you sign up for e-mail notification regarding new posts on as many forums as possible, especially the General Discussion forum. As with any online forums, especially when you are new to Sitecore, please remember to search for your issue before starting a new thread about it.

Accessing the Sitecore Shared Source Library

The Sitecore Shared Source Library (`http://bit.ly/vacX7d`) contains a variety of free modules that you can use, customize, and review for educational and other purposes. All modules include source code that you can use to further your knowledge of Sitecore. See also the Sitecore Shared Source blog (`http://sharesitecore.wordpress.com`). For information about contributing to Sitecore Shared Source projects, see my blog post at `http://bit.ly/r1SHI6`.

Sitecore Blogs

As a highly capable and extensible development framework, Sitecore is both broad and deep. One of the best ways to learn Sitecore is to read blog posts by people who have been working with the software for years, especially those who have addressed requirements similar to yours. Instead of listing individual blogs here, I suggest reviewing the Sitecore pipe on Yahoo! (`http://bit.ly/w0sPlD`, or `http://bit.ly/sjXeia` for the RSS feed).

Sitecore Demo Sites

Sitecore maintains several sample solutions for demonstration, training, testing, and other purposes. Some are available through the developer training program mentioned in a previous section. Others are available on the Sitecore Developer Network or the Sitecore Partner Network. If you cannot find an appropriate sample solution for your activity, contact Sitecore (`http://bit.ly/tOeWfu`).

www.it-ebooks.info

These are not production-quality solutions, do not necessarily follow Sitecore best practices, and are not optimized for performance or scalability. I advise against using such demonstration sites as a basis for your own solutions. You may not understand every aspect of these implementations, and you are unlikely to use every feature included. While you can learn a great deal about Sitecore by evaluating them, you will develop a better understanding of Sitecore if you build a complete implementation from scratch.

Virtual and Local Sitecore User Groups

A number of regional Sitecore groups exist for in-person exchanges with other members of the Sitecore community. You can join the Sitecore Users Virtual Group (`www.sitecoreug.org`) sponsored by Sitecore Technology Partner Hedgehog Development (`http://hhogdev.com`), organized around a mission to further educate and support the global Sitecore community. When your Sitecore skills have reached an appropriate level, you can even volunteer to present at a user group meeting.

The Sitecore Support Portal

You can file questions, defect reports, feature requests, and other types of cases at the Sitecore Support Portal (`http://support.sitecore.net`). For defect reports, include a minimal but detailed reproduction scenario.

If you cannot access the Sitecore Support Portal, contact Sitecore in your region to determine access requirements (`http://bit.ly/tOeWfu`). If you ever feel that Sitecore Customer Service does not address your issue quickly enough through the Sitecore Support Portal, provide your case identifier to your regional Sitecore contact, who can escalate it through the Sitecore Customer Service department.



To determine the location of existing documentation specific to a topic of interest, request additional documentation, report errata in Sitecore documentation, and for all other documentation requests, send an e-mail to the address listed in the footer of every SDN page (`doc@sitecore.net`).

Before you file an issue in the Sitecore Support Portal, you can use the SDN forums to collect information about your issue. Reporting issues on the SDN forums can have numerous benefits, including the following:

- Due to the number of eyes reading SDN forum posts, you can sometimes get an answer more quickly on the forums than you would through the Sitecore Support Portal.
- The more people who see an issue, the more likely it is that one of them has seen it before.
- Additional perspectives can often lead to superior solutions and workarounds.
- Forum posts inform the developer community about issues.
- Other developers can try to confirm your issue in an attempt to determine whether it is specific to a Sitecore product or version, your configuration, your solution, or something else.

www.it-ebooks.info

- People more familiar with the Sitecore documentation, release notes, and known issues will point you to them when appropriate.
- The community can search forum threads to identify topics quickly without needing to file support cases.

If you file a case in the Sitecore Support Portal after posting about it on the SDN forums, add the case identifier to the forum thread afterwards so that other developers with the same issue can reference that case.



The Sitecore Customer Service department is not responsible for the Sitecore Developer Network forums. Any information that you post on the forums should also appear in your support case. Instead of filing a support case that links to your SDN forum thread, condense that information to provide succinct details in the text of your support case.

Sitecore Symposium

In 2010, Sitecore initiated its annual conference with in-person events in Boston and Copenhagen to engage, educate, and motivate Sitecore developers and partners. Originally named Dreamcore, Sitecore subsequently renamed this conference as the Sitecore Symposium. The locations of the conference vary by year.

Each event provides multiple tracks that include customer-oriented content relevant to both technical and business users. The conference is an opportunity for members of Sitecore's active and dynamic online community to interact outside the constraints of web forums and social media, and to provide feedback directly to those within the Sitecore organization.

Sitecore Success Services

Sitecore Success Services (`http://bit.ly/xmfSxt`) can assist you through the entire project life cycle to help ensure the success of any implementation. From requirements analysis through deployment and maintenance, Sitecore Success Services helps you apply best practices and optimize your Sitecore solutions.

www.it-ebooks.info

www.it-ebooks.info

B

Installing Sitecore

WHAT'S IN THIS APPENDIX?

- Preparing to Install Sitecore
- Installing Microsoft SQL Server
- Installing Sitecore
- Creating a Microsoft Visual Studio project

This appendix contains information to help you determine how to approach Sitecore installation. It also includes instructions to install Microsoft SQL Server, install Sitecore using the setup program, install Sitecore from an existing archive distributed by Sitecore or an archive generated from an existing solution, and create a project for your solution using Microsoft Visual Studio.

PREPARING TO INSTALL SITECORE

This section provides information that you can use in conjunction with the scalability details in Chapter 6 to develop a strategy before you install Sitecore in various environments such as development, test, production content management, and production content delivery. Beginning by listing some prerequisites and highlighting information for you to collect before installing Sitecore, this section continues to provide specific instructions to install and configure Microsoft SQL Server.

You can install Sitecore using a setup program or from an archive of an existing Sitecore solution, such as a `.zip` file of a blank solution provided by Sitecore equivalent to the contents of the setup program. You can follow the `.zip` file installation process to duplicate the databases and document root of an existing Sitecore installation to create a new installation. Alternatively, you can use the Sitecore Installer (`http://bit.ly/xr9jq2`) Sitecore Shared Source module to automate installation from a `.zip` file.

www.it-ebooks.info



Before you install the Sitecore CMS, always refer to the current Sitecore Installation Guide (`http://bit.ly/pv7X3B`*), Installation Troubleshooting Guide (*`http://bit.ly/qmzVpY`*), Browser Configuration Reference (*`http://bit.ly/qfwYUW`*), release notes including known issues (*`http://bit.ly/pyPUPV`*), and other documentation on the Sitecore Developer Network (*`http://sdn.sitecore.net`*) relevant to your version of the software, including optional modules.*

Ensure that all web clients, web servers, and database servers meet Sitecore's minimum (and preferably recommended) hardware requirements. Install Sitecore in development environments first, then test, and eventually production. Installing Sitecore on the production servers on day one will have little benefit.



Install Sitecore to an IIS document root directory. Do not attempt to install Sitecore to a subdirectory or configure Sitecore as a virtual directory. Even if you can get Sitecore to function in a virtual directory, Sitecore may deny support for such solutions.

If you will host the databases on a separate machine, install the databases on the database server before installing Sitecore on the web server.

To enhance security regardless of which installation approach you choose, after you install Sitecore, create a CMS user with administrative rights in the Sitecore security domain. Then remove the default user named admin from that security domain.

Choosing an Installation Strategy

Before you install Sitecore, you should choose an installation strategy. To evaluate the product without consideration for performance or scalability, you can use a single machine for both the database server and the Sitecore application, and you can use the setup program for installation. For convenience, you can implement this simplest-possible configuration on development workstations, especially portable computers such as laptops that may not have persistent connections to dedicated database servers. You can also use this setup in test environments.

The further you move into the CMS project life cycle, the more likely you are to install Sitecore manually rather than use the setup program, and to use separate database servers. As the solution develops, assets accumulate, consisting of at least files and items in Sitecore databases. To create a new instance of the solution, you need to install Sitecore and then replicate these assets. Rather than perform a two-step process, depending on how you manage releases of your solution, you may find it easier to follow the instructions for installing the `.zip` file distributive of Sitecore to copy your entire solution, including Sitecore itself and your custom implementation, as well as the supporting databases. You are most likely to use the setup program to install Sitecore instances supported by new, dedicated databases in development environments than in production or even

www.it-ebooks.info

test environments. For example, to add a production content delivery instance, you might clone an existing instance, rather than starting from a blank solution created by the Sitecore automated setup program, reconfiguring that solution, and then migrating your solution assets.

Sitecore depends on a database server; this book assumes that you use Microsoft SQL Server. You must install the database server before you install Sitecore. The Sitecore setup program configures Sitecore to use SQL Server authentication for the database connection rather than Windows authentication.

When you use the setup program to install Sitecore, typically in a single-instance environment, you have a small number of significant choices to make, such as whether to use a local or remote database server. Configurations that include a number of development, test, production content management, and production content delivery instances, or that require load-balancing within environments, require more decisions. The same is true for solutions that involve optional modules such as the Digital Marketing System (DMS), especially those that rely on additional databases. These decisions include when to share databases between instances, how to replicate data between instances and databases, how to deploy updates to Windows components, Sitecore, and the custom solution, when to add instances to an environment, and when to add additional content delivery and other databases. For more information about such complex configurations, see Chapter 6 and the Sitecore Scaling Guide (`http://bit.ly/oKvYuN`).

Before you begin any type of Sitecore installation process, you must determine values for the system components shown in Table B-1.

TABLE B-1: Required Sitecore Installation Information

COMPONENT	DESCRIPTION	EXAMPLE
Installation Path	Subdirectory containing Sitecore subdirectories and files	`C:\inetpub\sitecore\SitecoreBook`
Website Name	Name of IIS website to host the Sitecore solution	`sitecorebook`
Hostname or DNS Entry	DNS entry or an entry in the hosts file on all clients mapping a name to the IIS website hosting Sitecore, allowing URL routing to the instance	`sitecorebook.net`
Database Prefix	Prefix for naming Sitecore databases connected to the instance	`SitecoreBook`
Namespace	Code namespace for .NET customizations and extensions	`SitecoreBook`
Assembly	Name of assembly (`.dll` file) containing .NET customizations and extensions	`SitecoreBook.dll`
License	Sitecore content management, content delivery, or other license	`license.xml`

www.it-ebooks.info



Install Sitecore to a subdirectory outside of the `C:\inetpub\wwwroot` *subdirectory so that your Sitecore solution does not appear to be a subdirectory of the default IIS web site.*

As shown by these examples, you can often derive many of these variables from the project name (SitecoreBook in this example). Specific database names can also include a suffix and an additional prefix to indicate the environment type, such as test or production and content management or content delivery.



In addition to using a hostname or a DNS entry to access the Sitecore instance, you can configure IIS to bind `localhost`*, the machine name, and the IP address of the server to the website hosting the Sitecore solution.*

Installing Microsoft SQL Server

As mentioned previously, you can install Microsoft SQL Server on the web server hosting the Sitecore solution. This configuration is appropriate for developers and in demonstration environments, but it is not scalable and is less secure than separating the database server from the web server. You can even use the free Microsoft SQL Server Express Edition to support Sitecore. This section contains instructions to install SQL Server Express.

You can download Microsoft SQL Server Express 2008 R2 from `http://bit.ly/uc9kJo`. I recommend that you select a SQL Server distributive that includes tools, specifically the Management Studio Express application.



Management Studio Express distributed for SQL Server Express is visually and functionally similar to Management Studio distributed for SQL Server. This book refers to both applications as Management Studio.

When you install SQL Server, install Management Studio, enable SQL Server authentication, and create a password for the sa user.



While the general principles described in this section apply to any release of SQL Server, the specific instructions and screen shots may not match versions released after the publication of this book.

www.it-ebooks.info

The installation process has many screens, but you can accept most of the defaults described in the following instructions to install SQL Server:

1. Run the SQL Server setup program. After extracting files, the SQL Server Installation Center appears.
2. Click New Installation or Add Features To An Existing Installation. The SQL Server 2008 R2 Setup window appears at the License Terms dialog.
3. Accept the license terms and click Next. The Feature Selection dialog appears as shown in Figure B-1.

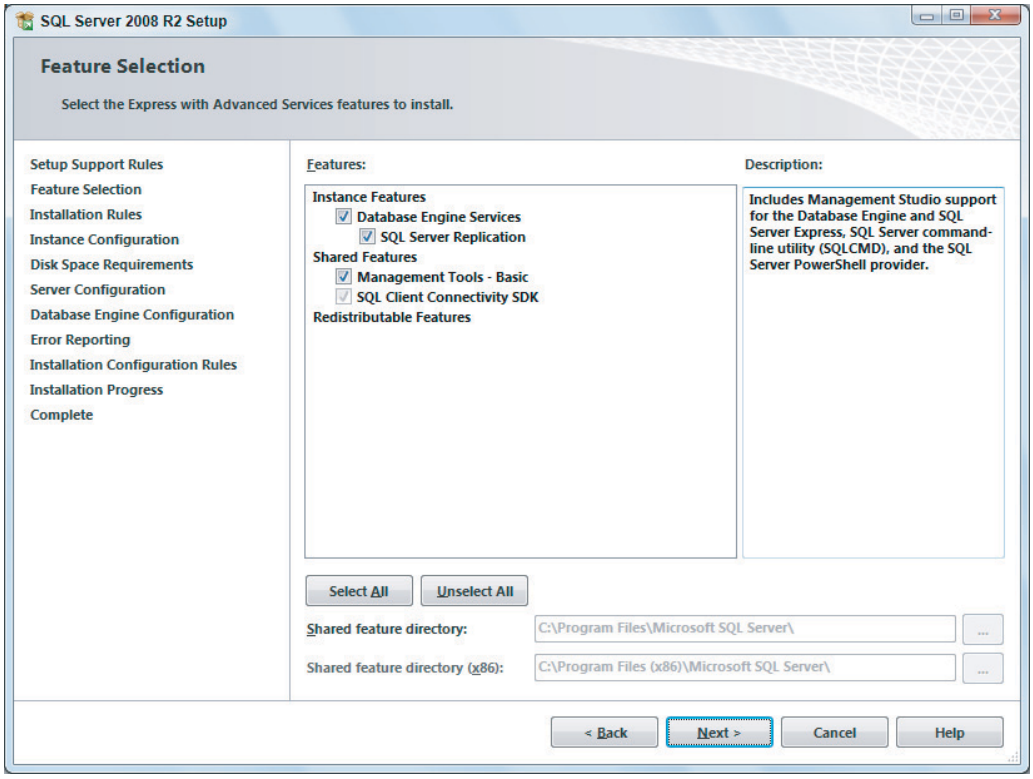


FIGURE B-1

4. Under Shared Features, select Management Tools – Basic, and click Next. The Instance Configuration dialog appears as shown in Figure B-2.
5. Accept the defaults and click Next. The Server Configuration dialog appears as shown in Figure B-3.

www.it-ebooks.info

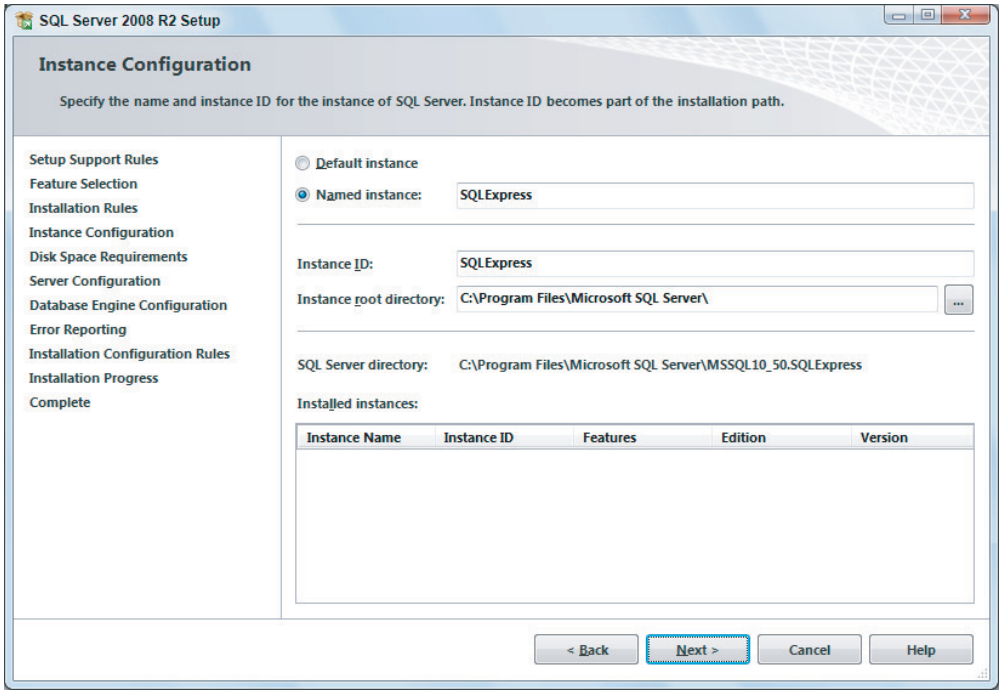


FIGURE B-2

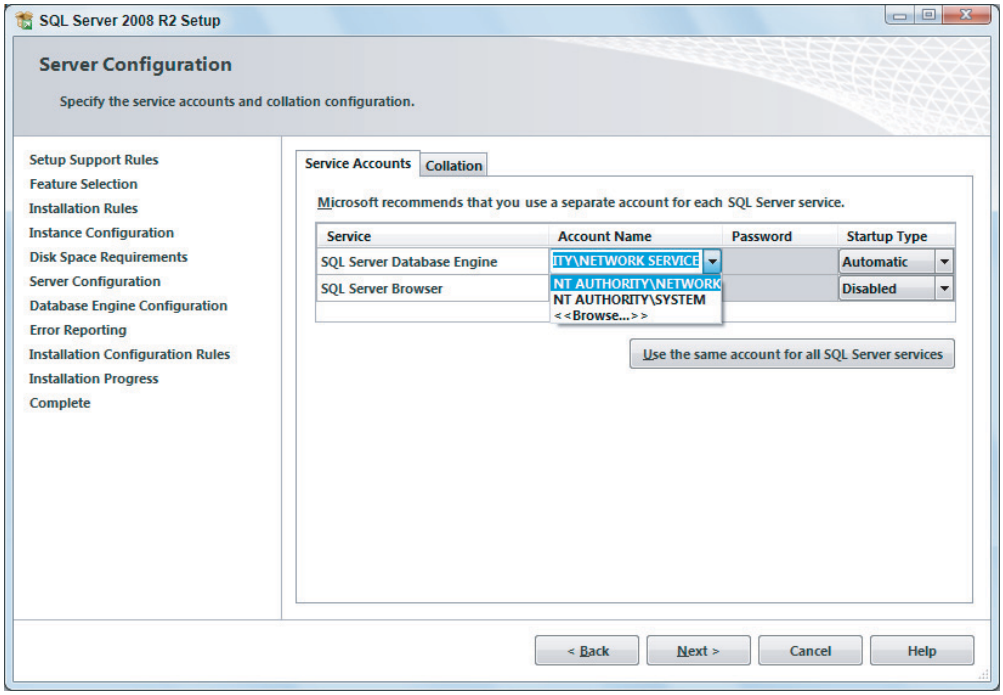


FIGURE B-3

www.it-ebooks.info



The Windows security account that you select will own the SQL Server Database Engine service. In some cases, you may want to change from the default to an alternate user. You may need to grant filesystem access rights to the database files for that user.

6. Click Next. The Database Engine Configuration dialog appears as shown in Figure B-4.

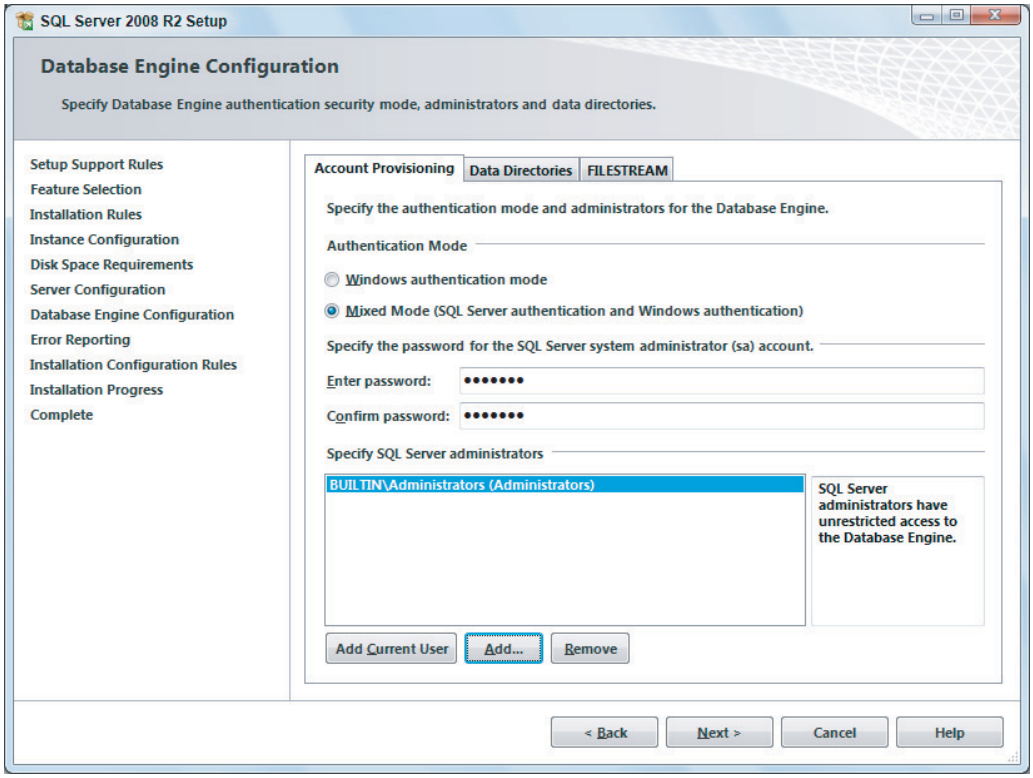


FIGURE B-4

7. Select Mixed Mode (SQL Server authentication and Windows authentication), enter a password for the sa user twice, click Add to add the local administrators group to the list of SQL Server administrators, and then click Next. The Error Reporting dialog appears.
8. Click Next. The Complete dialog appears.
9. Click Close. When you next see the SQL Server Installation Center, click Close again.

After installing SQL Server, use Microsoft Windows Update to install important updates to SQL Server and its components.

www.it-ebooks.info

Configuring an Existing SQL Server Instance

If you have already installed SQL Server, to enable SQL authentication for an existing instance of SQL Server configured to allow only Windows Authentication in Management Studio:

1. Right-click the server and then click Properties. The Server Properties dialog appears as shown in Figure B-5.

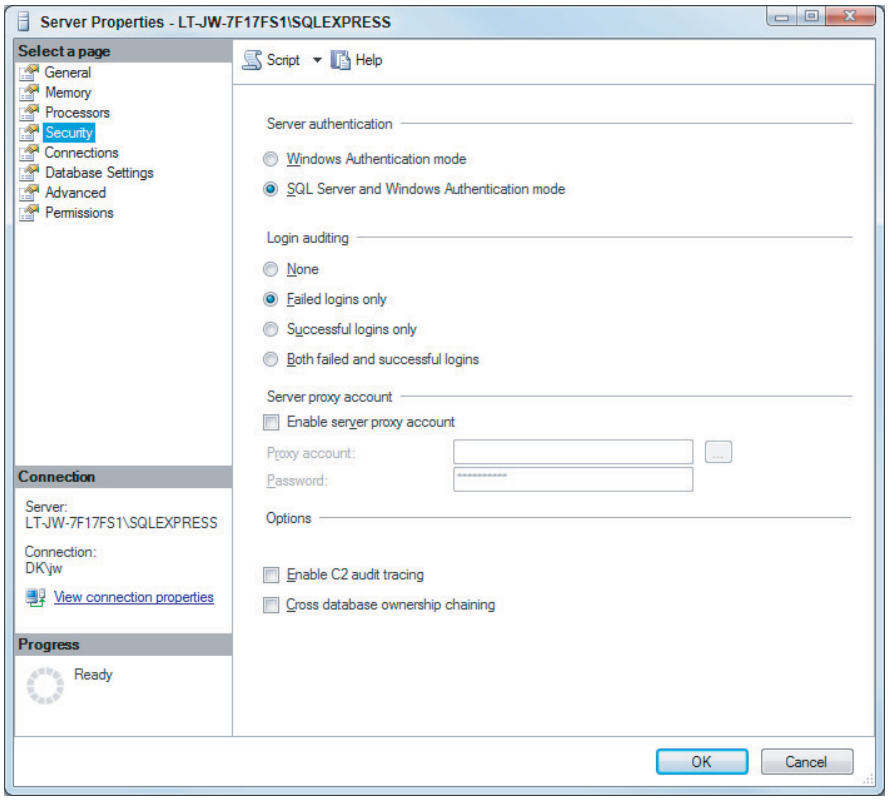


FIGURE B-5

2. Select the Security tab. Under Server authentication, select SQL Server and Windows Authentication mode, and then click OK. Management Studio appears.
3. Right-click the server, and then click Restart. A confirmation dialog appears.
4. Click Yes. The Service Control dialog appears after SQL Server restarts, returning you to Management Studio.
5. Under the server, expand Security, and then expand Logins. Right-click the sa login, and then select Properties. The Login Properties dialog appears as shown in Figure B-6.

www.it-ebooks.info

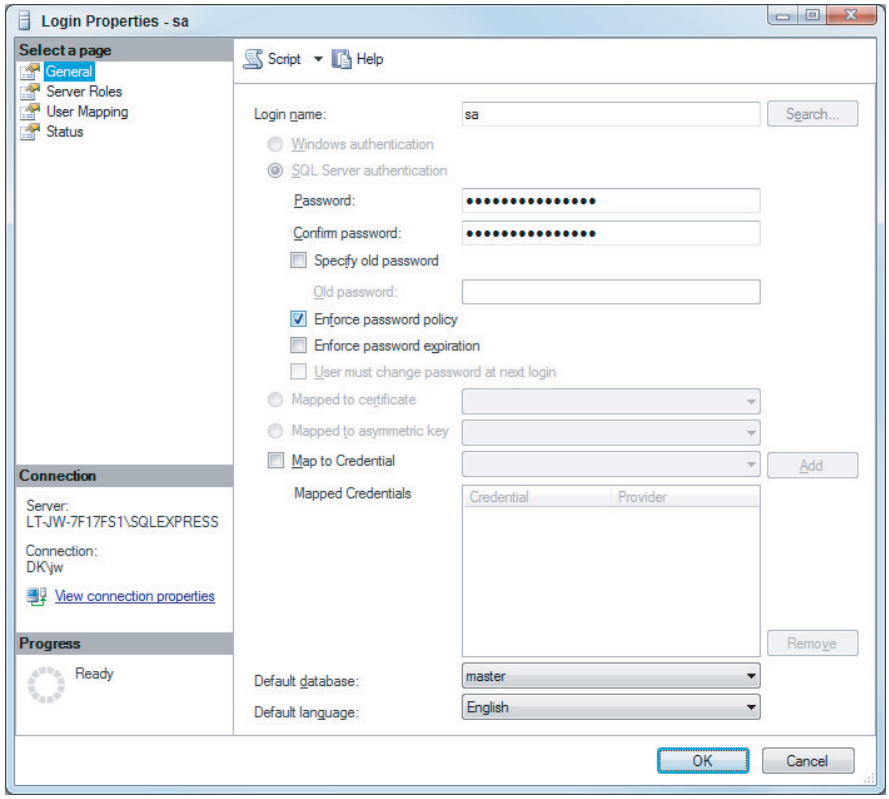


FIGURE B-6

6. Enter a password for the sa user in the Password field. Type the password a second time in the Confirm field.

7. Select the Status tab, choose Enabled under Login, and then click OK to return to Management Studio.

INSTALLING SITECORE WITH THE SETUP PROGRAM

If you plan to use a single machine as both the database server and the web server, you can install Sitecore by running the Installation Wizard once on that machine. Otherwise, you should first run the setup program on the database server to create the databases, and then run the setup program on the web server to install Sitecore.



If you cannot run the Sitecore setup program on the database server, you can create the databases from a `.zip` *file of the Sitecore CMS (the* `.zip` *file distributive) as described the Sitecore Installation Guide (*`http://bit.ly/pv7X3B`*).*

www.it-ebooks.info

When you run the Sitecore setup program, review each page of instructions. To see additional configuration options, such as to select a .NET Framework version, when you reach the screens on which the Advanced button appears, click that button to apply additional configuration options.



These instructions and screen shots describe the installation program for Sitecore CMS 6.5.0 rev. 110818, which may not exactly match other product versions.

If you plan to host databases on the web server, run the Sitecore setup program on the web server and select Full mode. If you use a separate database server, instead run the same setup program on the database server to create the databases (select Database Only) that you use to install Sitecore on the web server (select Client Only), or you can create the databases manually from a `.zip` file provided by Sitecore.

The Sitecore setup program overwrites the `SitecoreInstaller.log` file in the subdirectory specified by the `%TEMP%` environment variable or under `C:\`. If installation fails without providing a useful error message, review the contents of this file and provide it to Sitecore support. The following steps describe how to start the installation:

1. Start the installation by double-clicking the setup program (`.exe` file) in Windows Explorer. After the setup program extracts the required files, you see the Welcome to the Installation Wizard for Sitecore dialog.
2. Click Next. The License Agreement dialog appears.
3. Put a check beside I Accept the Terms of the License Agreement and then click Next. The Installation Type dialog shown in Figure B-7 appears.

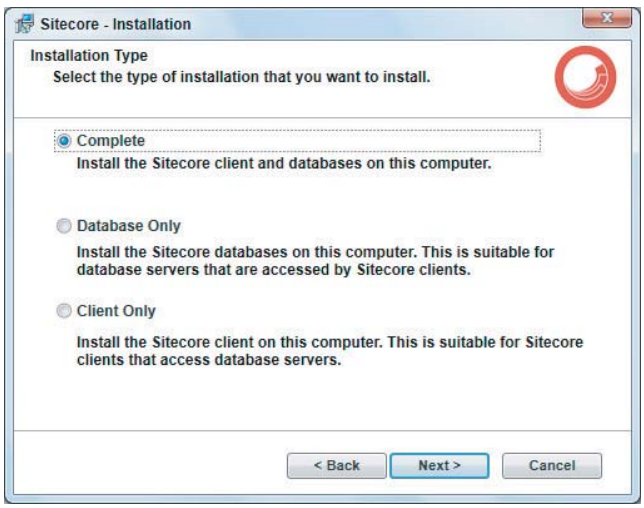


FIGURE B-7

4. To install both the databases and the Sitecore application, select Complete. To install only the databases, select Database Only. To install only the Sitecore application, select Client

www.it-ebooks.info

Only. Then click Next. The procedure will continue with the kind of installation you selected in one of the sections that follow.

Performing a Complete Installation

If you selected Complete in the Installation Type dialog shown in Figure B-7, the Instance Name dialog shown in Figure B-8 appears.

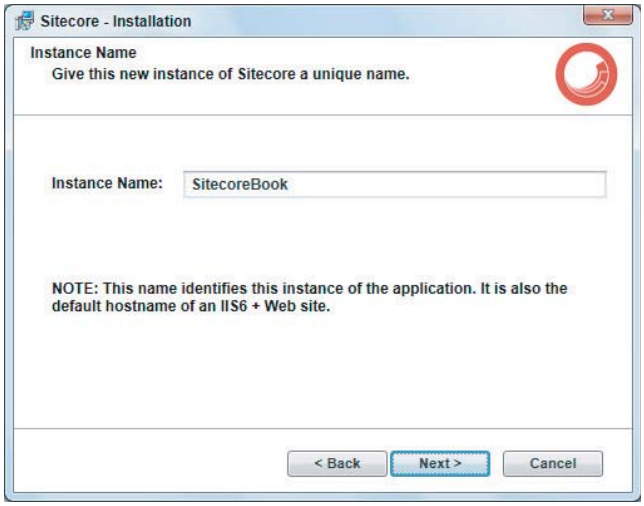


FIGURE B-8

In that case, follow these instructions to install both Sitecore and the databases on a single web server:

1. Type a unique name in the Instance Name field, which will set the name of the installation subdirectory, the database prefix, the website, and the application pool, and then click Next. The License File dialog shown in Figure B-9 appears.

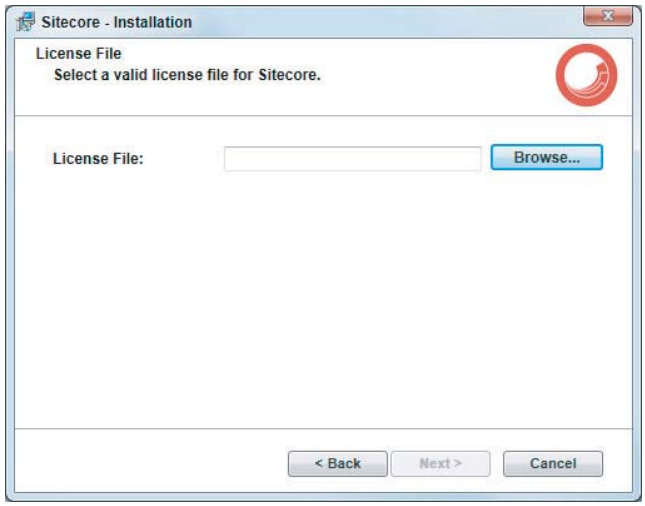


FIGURE B-9

www.it-ebooks.info

2. Select a valid license file for the instance, and then click Next. The Database Server dialog shown in Figure B-10 appears.

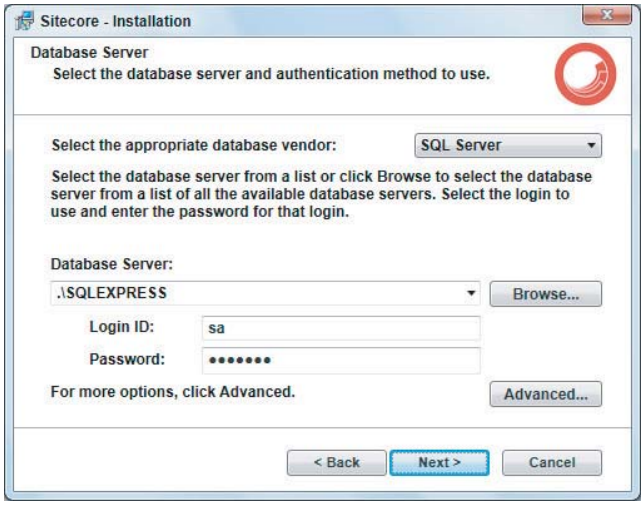


FIGURE B-10

3. Enter criteria to connect to the database server. To connect to the default instance of SQL Express on the local system, type **.\SQLEXPRESS** in the Database Server field. Type **sa** in the Login ID field, and enter the password for the SQL Server sa user in the Password field. You can click Advanced to configure options not shown by this dialog, such as whether to include the database prefix in the names of the files used to implement the database. When you are done, click Next. The Destination Folder dialog shown in Figure B-11 appears.

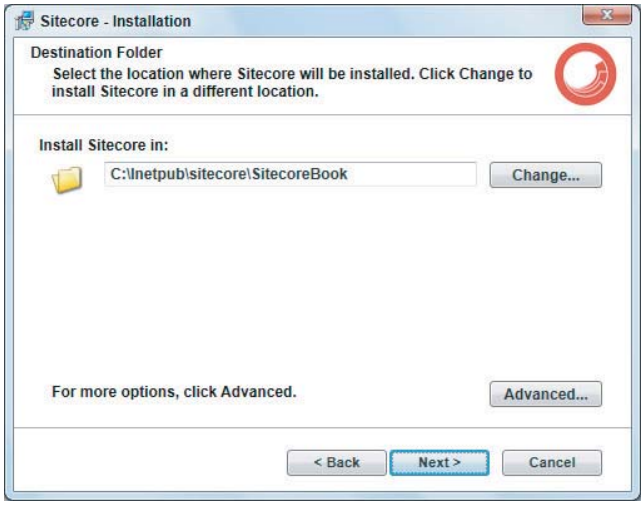


FIGURE B-11

www.it-ebooks.info

4. Type the path to the installation subdirectory in the Install Sitecore in field. You can click Advanced to access options not shown by this dialog, such as to configure the locations of the files used to implement the database. When you are done, click Next. The IIS Web Site dialog shown in Figure B-12 appears.

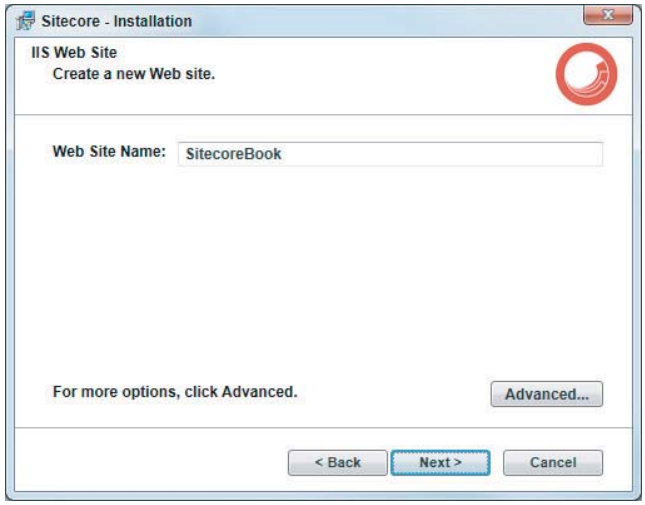


FIGURE B-12

5. Type the name of the new website in the Web Site Name field. You can click Advanced to access configuration options not shown by this dialog, such as to specify an HTTP port, to configure the application pool, or to select a .NET Framework version. Click Next. The Ready to Install Sitecore dialog appears.

6. Click Install. When installation is complete, the Sitecore Setup dialog appears.



You can select the Launch Sitecore checkbox to open the solution in a new browser window.

8. Click Finish to close the Sitecore Setup dialog and return to Windows Explorer.

www.it-ebooks.info

Performing a Database Only Installation

If you selected Database Only in the Installation Type dialog as shown in Figure B-13, the Instance Name dialog shown in Figure B-8 appears.

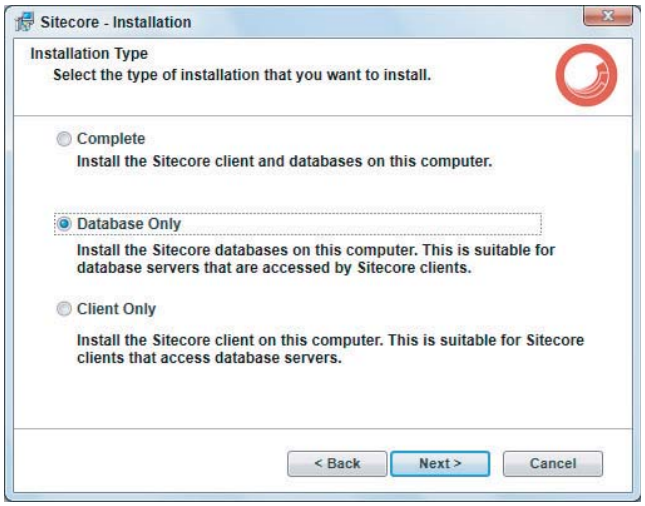


FIGURE B-13

In that case, follow these instructions to configure Sitecore databases on a database server:

1. In the Instance Name dialog shown in Figure B-8, type a unique name for the instance in the Instance Name field, which will set the database prefix. You can click Advanced to configure options not shown by this dialog, such as whether to include the database prefix in the names of the files used to implement the database. Then click Next. The Database Server dialog shown in Figure B-10 appears.
2. Enter criteria to connect to the database server as described in the section "Performing a Complete Installation." You can click Advanced to configure options not shown by this dialog, such as the prefix for database names and whether to include the database prefix in the names of the files that implement those databases. Click Next. The Destination Folder dialog shown in Figure B-14 appears.

www.it-ebooks.info

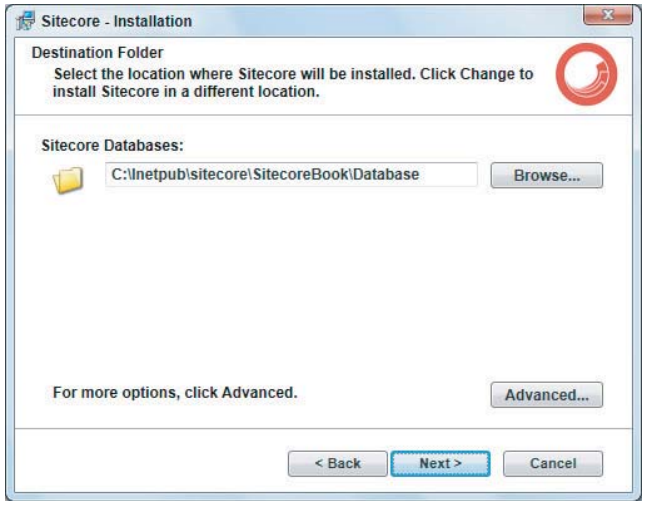


FIGURE B-14

3. Type the path where you want to store the SQL server database files in the Sitecore Databases field, and then click Next. The Ready to Install Sitecore dialog appears.
4. Click Install and wait for the setup process to complete.
5. Click Finish to close the Sitecore Setup dialog and return to Windows Explorer.

Performing a Client Only Installation

If you selected Client Only on the Installation Type dialog as shown in Figure B-15, the Instance Name dialog shown in Figure B-8 appears.

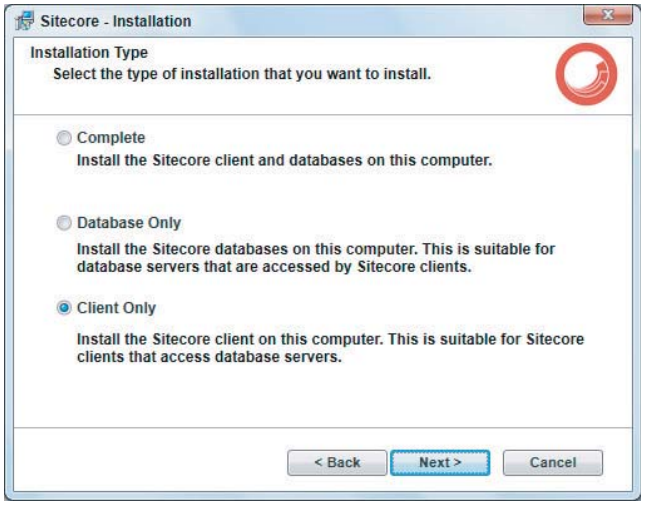


FIGURE B-15

www.it-ebooks.info

In that case, follow these instructions to install a Sitecore instance connected to existing databases:

1. When the Instance Name dialog appears as shown in Figure B-8, type a unique name for the instance in the Instance Name field, which will set the name of the installation subdirectory, the database prefix, the website, and the application pool, and then click Next. The License File dialog appears as shown in Figure B-9.
2. Select a valid license file, and then click Next. The Database Server dialog shown in Figure B-16 appears.

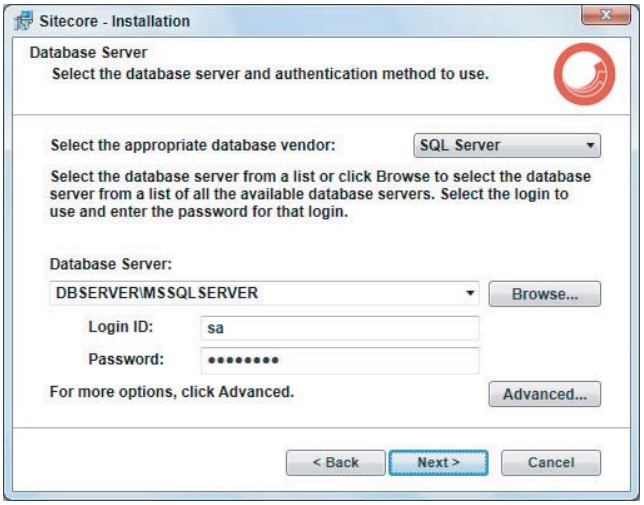


FIGURE B-16

3. Enter criteria to connect to the database server as described in the section "Performing a Complete Installation," and then click Next. The Destination Folder dialog appears as shown in Figure B-11.
4. Type the path to the installation subdirectory in the Install Sitecore field, and then click Next. The IIS Web Site dialog appears as shown in Figure B-12.
5. Type a name for the new website in the Web Site Name field, and then click Next. The Ready to Install Sitecore dialog appears.
6. Click Install. When installation completes, the Sitecore Setup dialog appears.
7. Click Finish to close the Sitecore setup program and return to Windows Explorer.

INSTALLING SITECORE FROM A ZIP OR OTHER ARCHIVE

If you cannot install Sitecore using the setup program, or if you need to replicate an existing Sitecore solution from one instance to another, you can follow the instructions to install Sitecore from a `.zip` file. To ensure a secure solution, follow the installation instructions meticulously. The Sitecore Installation Guide (`http://bit.ly/pv7X3B`) provides comprehensive manual installation instructions. Therefore, this section provides a process that you can follow to clone an existing Sitecore installation.

www.it-ebooks.info

Archiving a Sitecore Solution

First create an archive of the existing solution. Begin by stopping IIS. To stop IIS, I generally start a command prompt as a Windows administrator and issue the following command.

```
net stop w3svc
```

To start a command prompt as an administrator, right-click the Command Prompt icon in Windows and choose Run As Administrator from the context menu. After you stop IIS, use Microsoft SQL Server Management Studio to create a backup of each of the existing Sitecore databases. To back up Microsoft SQL Server databases easily, you can create a backup device, which can be a simple backup file that contains archives of multiple databases. To create such a backup device in Management Studio, follow these instructions:

1. Right-click Server Objects, then click New ➪ Backup Device. The Backup Device dialog appears as shown in Figure B-17.

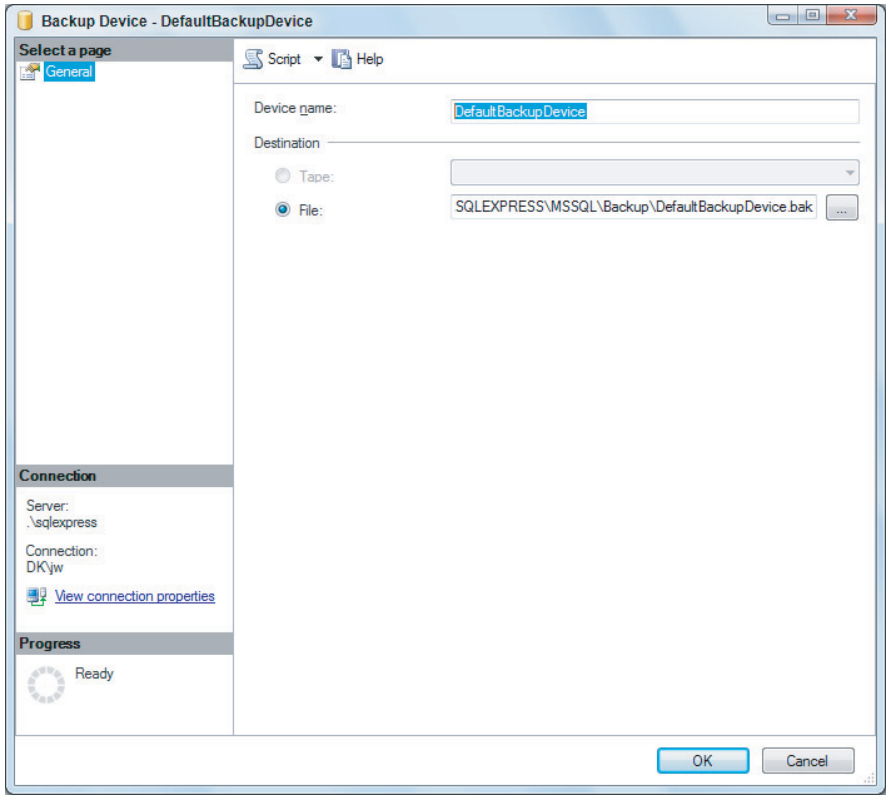


FIGURE B-17

2. Type a name for the new backup device in the Device name field, such as DefaultBackupDevice.
3. Select File under Destination, enter the full path to the file to use for database backups, and then click OK to return to Management Studio.

www.it-ebooks.info

To back up a database in Management Studio:

1. Right-click the database, then click Tasks ➪ Back Up. The Back Up Database dialog appears as shown in Figure B-18.

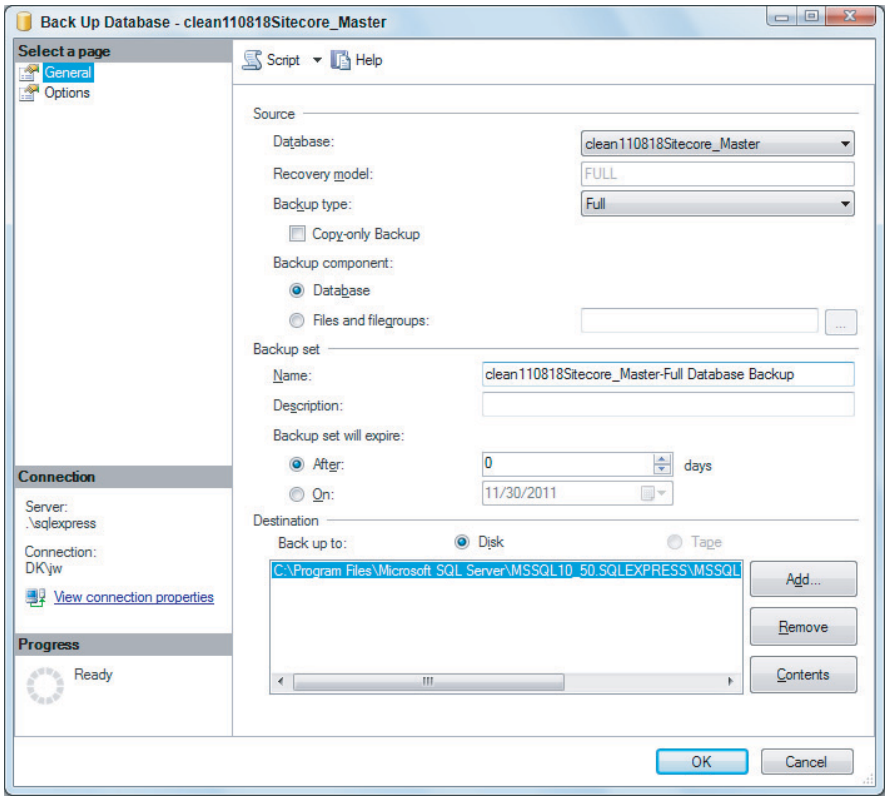


FIGURE B-18

2. Select Disk under Destination on the General tab, and then click Add. The Select Backup Destination dialog appears as shown in Figure B-19.

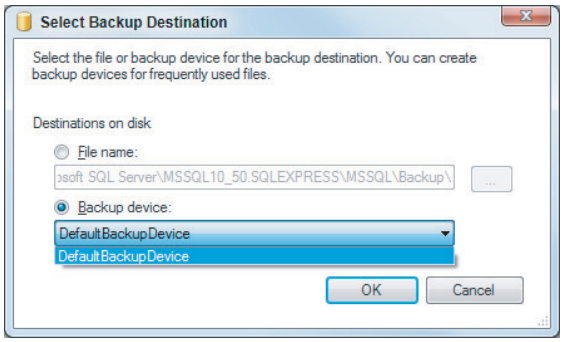


FIGURE B-19

www.it-ebooks.info

3. Click the Backup device radio button and select the backup device created previously. Click OK, and then click OK again. SQL Server adds an archive of the database to the backup device file, and then you return to Management Studio.

After backing up the Sitecore databases, create a compressed archive of the document root subdirectory for the existing installation (typically named `/Website` within an existing Sitecore installation subdirectory). Other than licenses, Sitecore creates all files in the `/Data` subdirectory of the installation subdirectory dynamically, so you do not need to copy the contents of that subdirectory. The process described in this section uses database backups, so you do not need a copy of the `/Databases` subdirectory, even if such a subdirectory exists in the existing installation subdirectory. Optionally, you can remove the contents of the `/temp` subdirectory within the `/Website` subdirectory, either before creating the archive or afterwards.

After you archive the existing solution, you can restart IIS. To do so, I execute the following command in a command prompt running as a Windows administrator as described previously.

```
net start w3svc
```

Creating and Configuring a New Installation

To implement the infrastructure required for the new installation, create the subdirectory that will contain the new installation, which does not have to use the same name as the existing installation but should follow the same structure (a subdirectory named after the project to contain the `/Website` and `/Data` subdirectories). Extract the archive created previously to populate the `/Website` subdirectory that will become the document root of the new instance.

To configure and secure the new instance, apply the manual installation instructions provided in the Sitecore Installation Guide (`http://bit.ly/pv7X3B`). At a minimum, manual configuration includes the following tasks described in this section:

1. Update references in copied configuration files, including subdirectory paths and database connection strings, to specify the new installation subdirectory.
2. Create a DNS entry for the new instance or update the hosts file.
3. Apply the appropriate license file to the new instance.
4. Configure the IIS website and application pool for the new instance.
5. Apply security to the subdirectories supporting the new instance.

In the Web.config file of the new instance, update any references to the installation path of the existing solution to the installation path of the new solution. For instance, set the `value` attribute of the `/configuration/sitecore/sc.variable` element named `dataFolder` in the Web.config file to the absolute operating system path to the `/Data` subdirectory in the new installation, such as `C:\inetpub\sitecore\SitecoreBook\Data`.

Create a DNS entry or update the `C:\windows\system32\drivers\etc\hosts` file on all clients to create a new entry for the host header that you will bind to the new instance.

www.it-ebooks.info

To create new databases from the database backups, in Management Studio (once for each database):

1. Right-click Databases, and then click Restore Database. The Restore Database dialog appears as shown in Figure B-20.

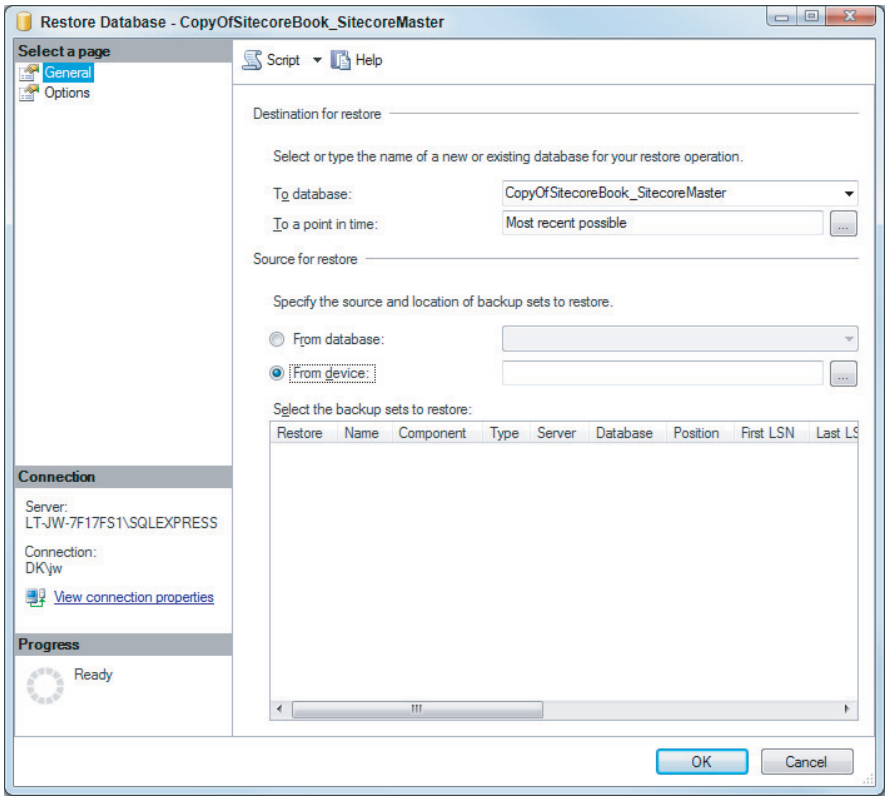


FIGURE B-20

2. Type a name for the new database in the Destination For Restore field, including a prefix, such as SitecoreBook, to identify the project and a suffix, such as _Master, to identify the purpose of the database. Optionally, include additional prefixes and suffixes to identify the environment such as test or production and content management or content delivery.

www.it-ebooks.info

3. Select the From device radio button in the Source For Restore field, and then click the ellipses (...). The Specify Backup dialog appears as shown in Figure B-21.

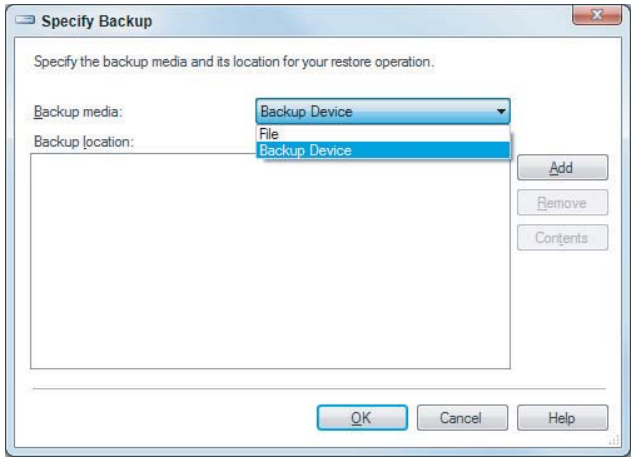


FIGURE B-21

4. Select Backup Device in the Backup media field and then click Add. The Select Backup Device dialog appears as shown in Figure B-22.

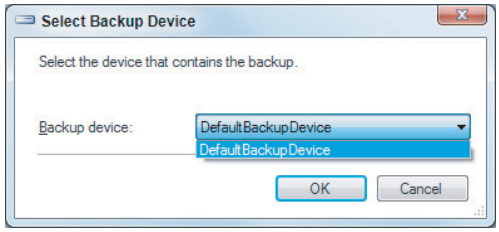


FIGURE B-22

5. Select the backup device you created previously in the Backup Device field, and then click OK. The Specify Backup dialog appears as shown in Figure B-21.

6. Click OK. When the Restore Database dialog appears, select the database to restore in the Select The Backup Sets To Restore field, and then click OK. SQL Server creates the new database from the backup.



If you enable remote event management as described in Chapter 6 and the Sitecore Scaling Guide (`http://bit.ly/oKvYuN`*), content delivery instances can share databases with other content delivery instances and even the content management environment, and therefore may not require new databases. Multiple content management instances can share databases as well. You do not necessarily need to create new databases and update connection strings for the new instance as described in this section.*

www.it-ebooks.info

Update the `connectionString` attributes of the `/connectionStrings/add` elements in the `/Website/App_Config/ConnectionStrings.config` file, to specify connection information for the new databases.

Overwrite the file specified by the `LicenseFile` setting in the Web.config file with the appropriate license for the new environment.

Perform the following steps to create an IIS website to host the new instance:

1. Open the IIS Manager shown in Figure B-23, found in the Administrative Tools folder of the Windows Control Panel or by issuing the `inetmgr` command.

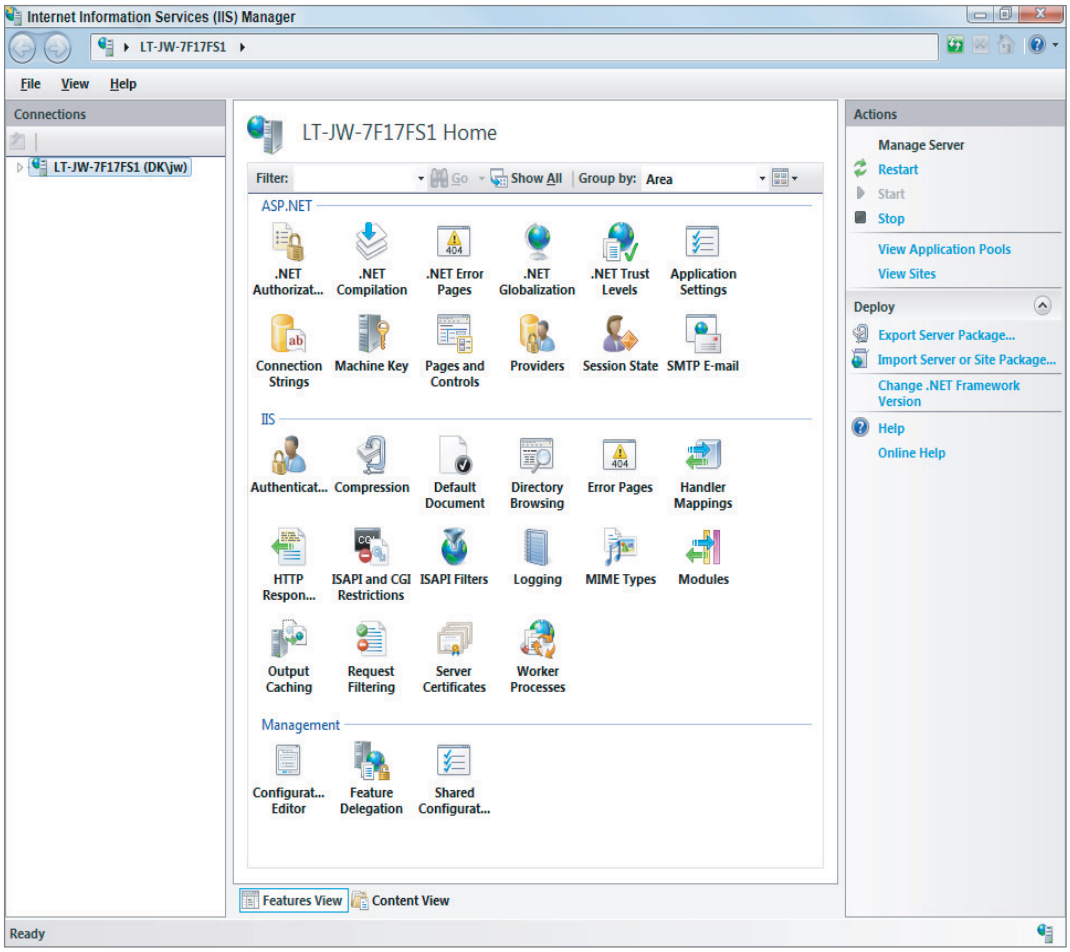


FIGURE B-23

www.it-ebooks.info

2. Expand the tree at the left, then right-click Sites, and then click Add Web Site. The Add Web Site dialog shown in Figure B-24 appears.

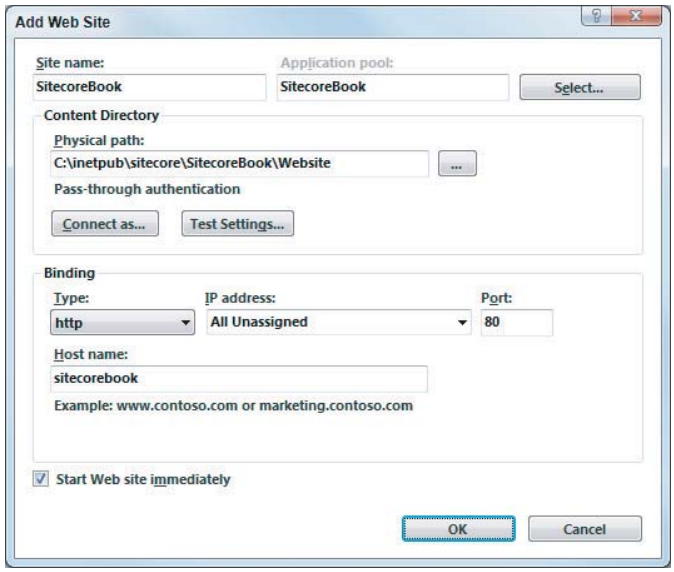


FIGURE B-24

3. Type the name of the website in the Site name field.
4. Select the `/Website` subdirectory from within the subdirectory for the new instance in the Physical path field.
5. Type the DNS entry or hostname created previously in the Host name field, and then click OK. You return to IIS Manager.
6. Click Application Pools in the tree at the left. You see a list of application pools as shown in Figure B-25, including the application pool IIS created automatically when you created the website (SitecoreBook in this example).

www.it-ebooks.info

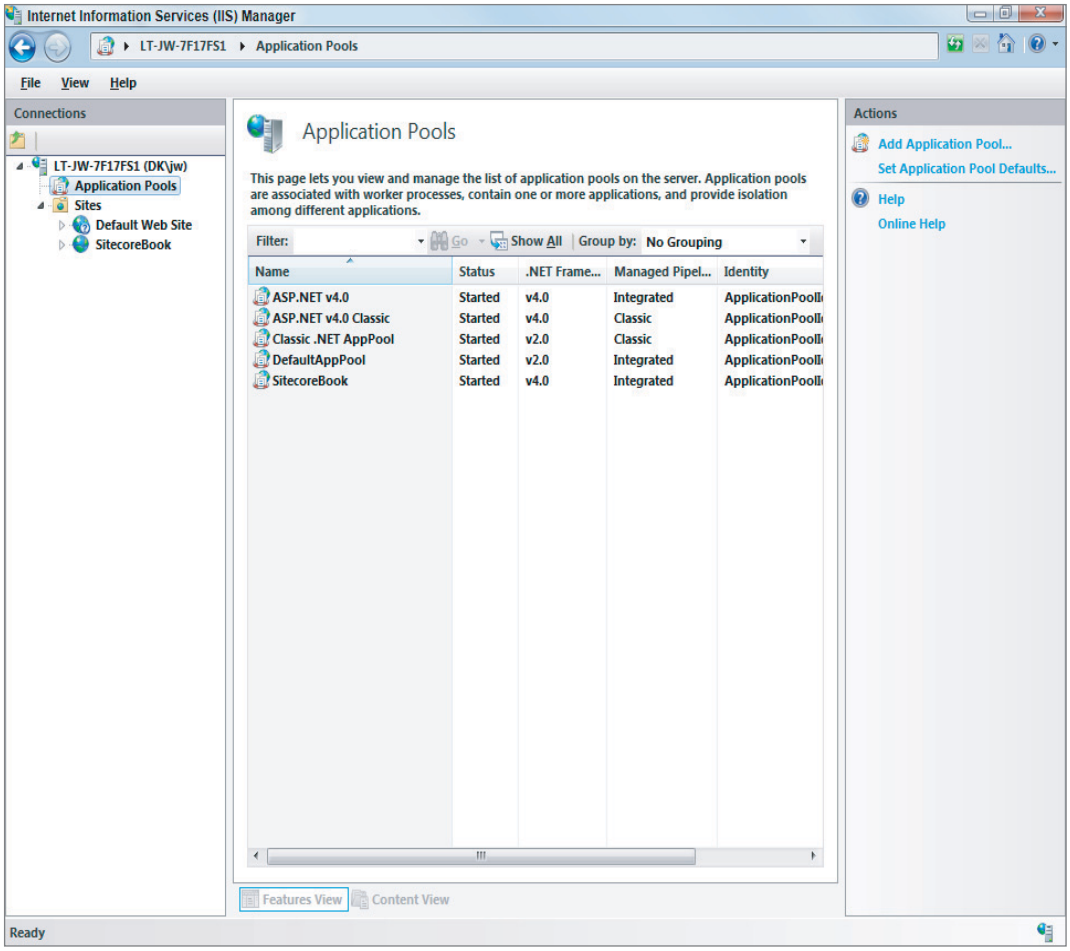


FIGURE B-25

7. Right-click the application pool named after the new website, and then click Basic Settings. The Edit Application Pool dialog shown in Figure B-26 appears.

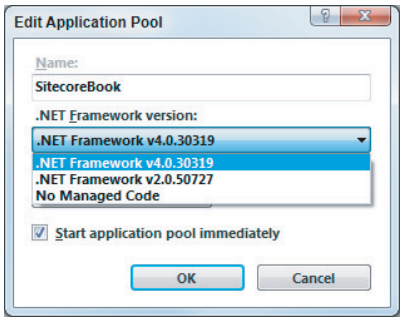


FIGURE B-26

www.it-ebooks.info

8. Select the version of the .NET Framework associated with the website that you copied or the version appropriate for the project in the .NET Framework version field. Figure B-26 shows selection of a version of the .NET 4.0 Framework.

9. Select the mode associated with the application pool supporting the website that you copied or the mode appropriate for the project in the Managed pipeline mode field. Figure B-27 shows selection of the Integrated pipeline mode.

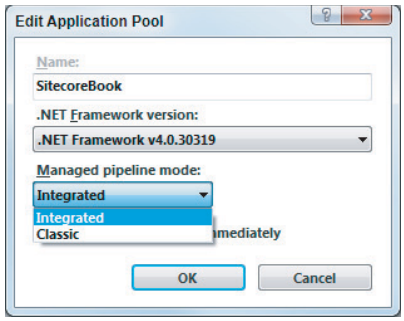


FIGURE B-27

10. Click OK to return to IIS Manager.

Confirm that you can access the solution, which at this point is not secure. Perform the following steps to deny anonymous access to the `/App_Config`, `/sitecore/admin`, `/sitecore/debug`, and `/sitecore/shell/WebService` subdirectories:

1. Open the IIS Manager shown in Figure B-23, found in the Administrative Tools folder of the Control Panel or by issuing the `inetmgr` command.

2. Expand the tree at the left, expand the new website, select the subdirectory that you want to modify, and then double-click Authentication under IIS. The Authentication dialog appears as shown in Figure B-28.

www.it-ebooks.info

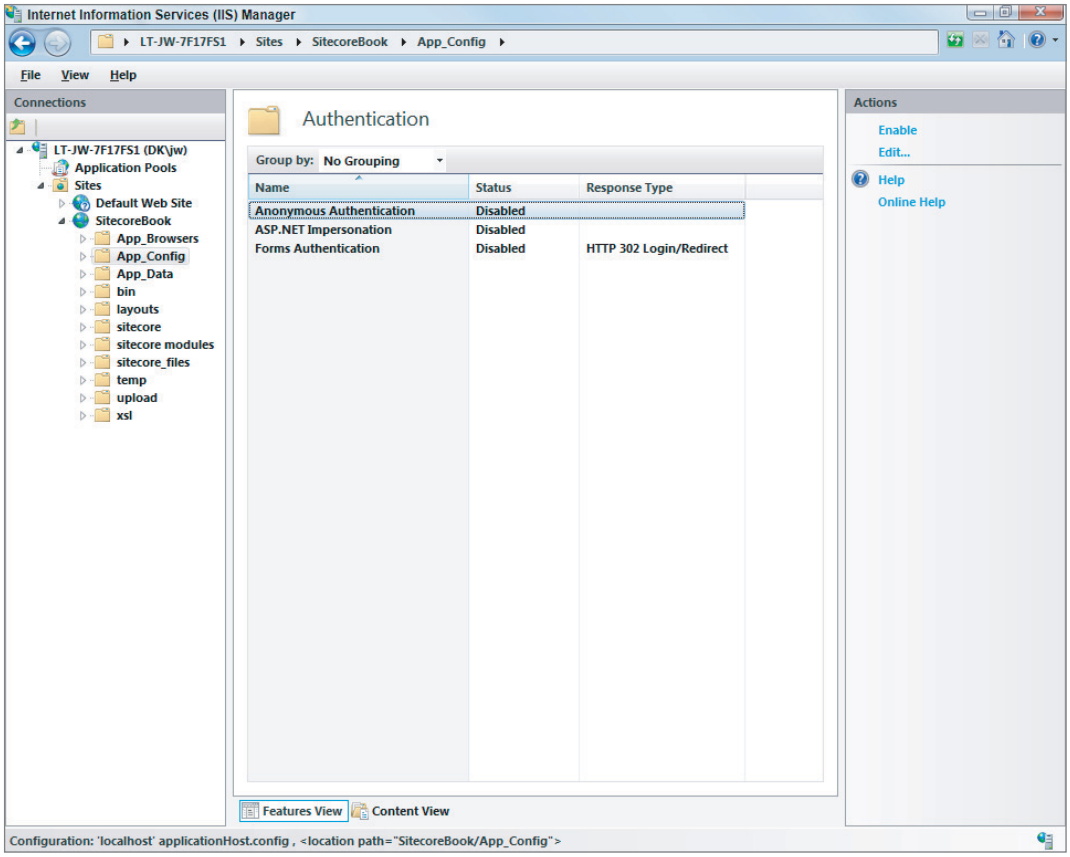


FIGURE B-28

3. Click Anonymous Authentication, and then click Disable to ensure that the Status column indicates Disabled, not Enabled as shown in Figure B-28.

4. Repeat steps 2 and 3 for each of the specified subdirectories.

Apply filesystem permissions as specified in the Sitecore Installation Guide (`http://bit.ly/pv7X3B`). The Sitecore setup program sets the application pool owner to the local Network Service account. Creating a new website in IIS Manager creates an application pool associated with the Application Pool Identity, which uses a virtual Windows account. Using Application Pool Identities has advantages over assigning ownership of application pools to the Network Service account. For one thing, you can more easily identify the ASP.NET worker process associated with each website when attaching the Visual Studio debugger. For more information about application pool identities, see `http://bit.ly/s3yDRi` and `http://bit.ly/sGruB1`.

You cannot see virtual accounts in user interfaces in which you apply access rights. Instead, you can type the name of the account into the user interface as when assigning filesystem rights as described in this section. The Application Pool Identity, associated with the application pool named DefaultAppPool and associated with the default IIS website is IIS AppPool\DefaultAppPool. This is what you would enter to set rights for that Application Pool Identity. If you create a website named

www.it-ebooks.info

SitecoreBook, by default, IIS creates an application pool with the same name, named after the website (technically, it uses the value that you enter in the Application pool field of the Add Web Site dialog shown in Figure B-24, which defaults to the name of the website). The Application Pool Identity account for that website is the name of the application pool (IIS AppPool\SitecoreBook), which is what you would enter to set rights for that Application Pool Identity.



Do not confuse the IIS AppPool prefix, which identifies a domain built into the operating system for dynamic virtual accounts, with a machine name or a domain name, which provide domains that contain traditional built-in accounts and accounts such as Network Service, and accounts that you create such as your personal account.

As an alternative to typing in the virtual account name, you can configure the application pool to run under a different user, such as Network Service. Perform the following steps to configure the application pool to run under a different user:

1. Open the IIS Manager, found in the Administrative Tools folder of the Windows Control Panel or by issuing the `inetmgr` command, and then click Application Pools. Figure B-25 shows the IIS Manager as it appears after you click Application Pools.
2. Select the application pool created automatically (named after the website), and then click Advanced Settings. The Advanced Settings dialog shown in Figure B-29 appears.

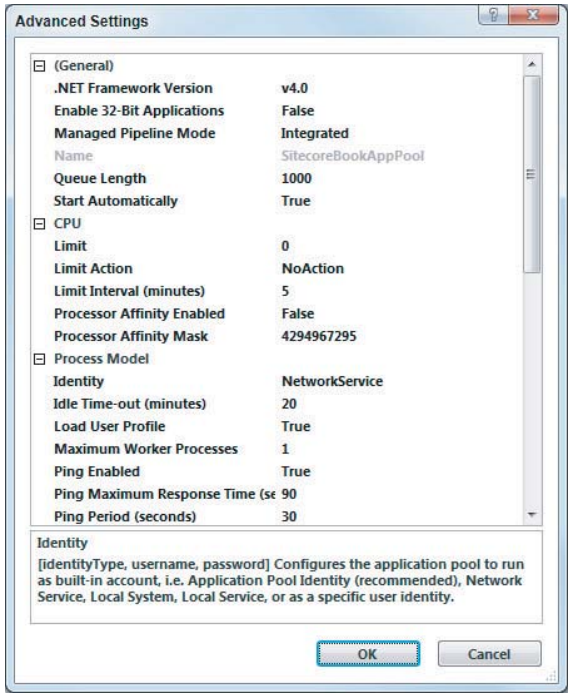


FIGURE B-29

www.it-ebooks.info

3. Click the ellipses (...) in the Identity field under Process Model. The Application Pool Identity dialog shown in Figure B-30 appears.

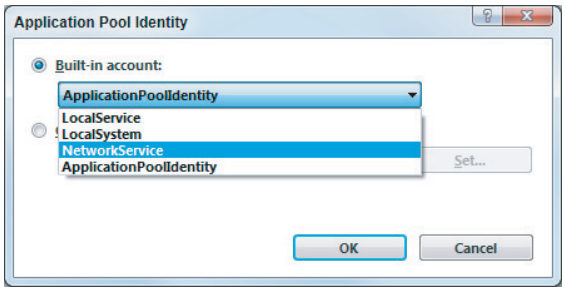


FIGURE B-30

4. Click the Built-in account radio button, select the account (such as NetworkService), and then click OK. You see the Advanced Settings dialog shown in Figure B-29.
5. Ensure that the Load User Profile field under Process model is set to True and the Maximum Worker Processes field under Process model is set to 1 as shown in Figure B-29.
6. Click OK.

The owner of the application pool must have the listed permissions to the following subdirectories, including read access to ancestors and the listed access to descendant subdirectories:

- `%WINDIR%\Globalization` — Modify (typically `C:\Windows`)
- `%WINDIR%\temp` — Modify (typically `C:\Windows\Temp`)
- `%PROGRAMDATA%\Microsoft\Crypto` — Modify (typically `C:\ProgramData`)

The following steps describe how to apply filesystem permissions using Windows Explorer:

1. Right-click the subdirectory to invoke the context menu.
2. Select Properties. The Properties dialog shown in Figure B-31 appears.

www.it-ebooks.info

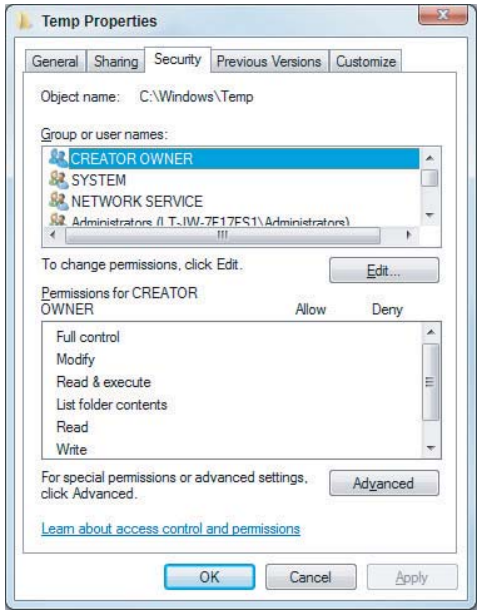


FIGURE B-31

3. Select the Security tab, and then click Edit. The Permissions dialog shown in Figure B-32 appears.

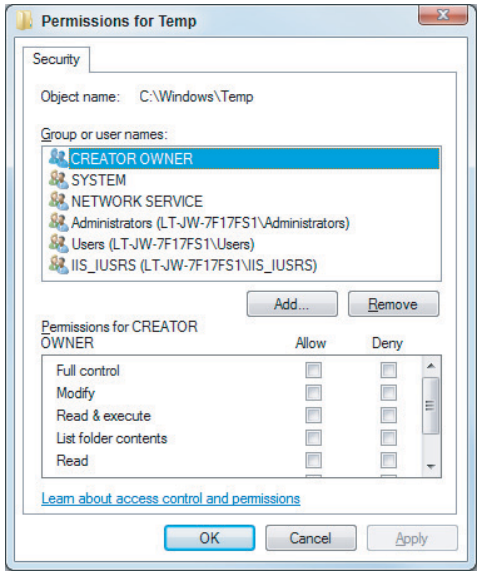


FIGURE B-32

www.it-ebooks.info

To update filesystem permissions for an account that already has some explicit access rights to the subdirectory, perform the following steps in the Permissions dialog:

1. Choose the account in the Group or User Names field.
2. Check the Allow or Deny checkboxes as appropriate, and then click OK. The Windows Security dialog appears while Windows applies the permissions, and then you see the Properties window.
3. Click OK to return to Windows Explorer. If you see a Windows Security dialog asking you to confirm changes to filesystem permissions, click Yes.

To grant filesystem permissions to an account that does not already have some explicit access rights to the subdirectory, perform the following steps in the Permissions dialog:

1. Click Add. The Select Users or Groups dialog shown in Figure B-33 appears.

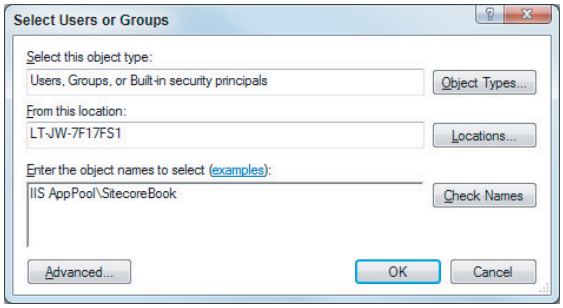


FIGURE B-33



When checking or setting ownership of an application pool, applying filesystem permissions, or otherwise working with Windows accounts, be sure to check or select the local machine or domain before checking or setting the account, for example in the From this location field in the Select Users or Groups dialog shown in Figure B-33. Otherwise, you may not find the user or group on which you wish to operate. In some cases, you may wish to select a Windows security domain and select a user or group from that domain rather than from the local machine.

2. Type the name of the account in the Enter The Object Names To Select field, such as Network Service or the name of the Application Pool Identity. You can click Check Names to validate what you enter. Alternatively, you can click Advanced to access the Select Users and Groups dialog, in which you can enter search criteria, click Find Now, select a user, and then click OK to select an account.

www.it-ebooks.info



If Windows presents the Name Not Found dialog, click Cancel to dismiss the dialog, and then in the Select Users or Groups dialog, click Locations and ensure that Location is the local machine or a Windows security domain according to your requirements.

3. Put checks in the Allow or Deny columns to allow or deny the access right, and then click OK. The Windows Security dialog appears while Windows applies the permissions, and then you see the Properties window.
4. Click OK to return to Windows Explorer.

To propagate permissions from a subdirectory to all of its subdirectories, recursively, perform the following steps in Windows Explorer:

1. Right-click the subdirectory to invoke the context menu.
2. Click Properties to invoke the Properties dialog shown in Figure B-31.
3. Select the Security tab, and then click Advanced. The Advanced Security Settings dialog shown in Figure B-34 appears.

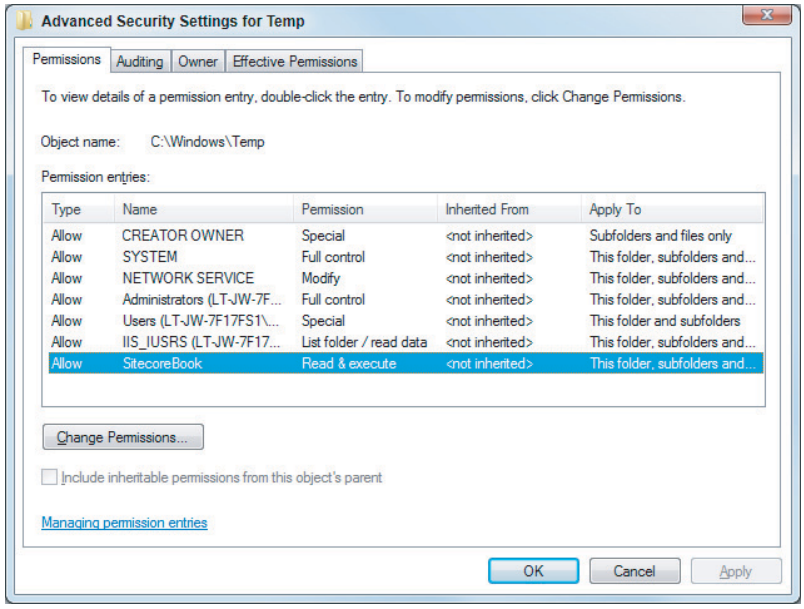


FIGURE B-34

www.it-ebooks.info

4. Select the user or role, and then click Change Permissions. A new Advanced Security Settings dialog like the one shown in Figure B-35 appears.

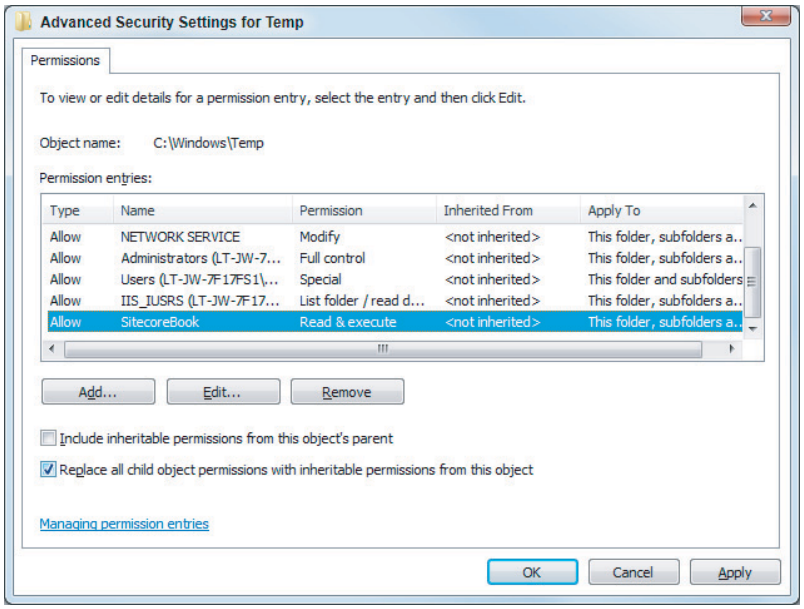


FIGURE B-35

5. Select the application pool owner, check Replace All Child Permissions With Inheritable Permissions From This Object, and then click OK. The Windows Security dialog appears.
6. Click Yes. After Windows has applied the permissions, you return to the Advanced Security Settings dialog.
7. Click OK, and then click OK again to return to Windows Explorer.

Perform the following steps to make the owner of the application pool a member of the Performance Monitor Users group:

1. Right-click Computer on the Windows desktop to invoke the context menu, and then click Manage. The Computer Management application appears.
2. Expand Local Users and Groups, select Groups, and then double-click the Performance Monitor Users group. The Properties dialog appears as shown in Figure B-36.

www.it-ebooks.info

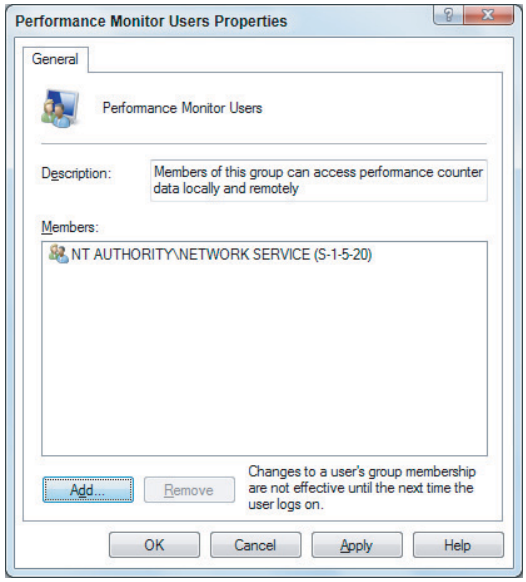


FIGURE B-36

3. Click Add. The Select Users dialog appears as shown in Figure B-37.

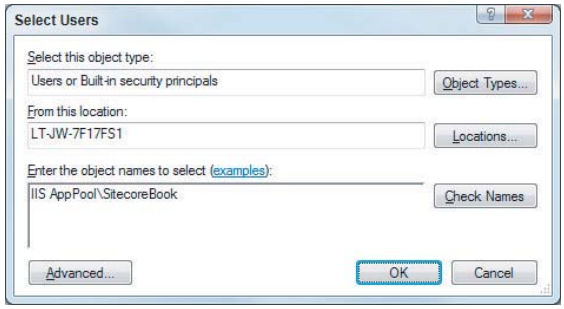


FIGURE B-37

4. Enter the name or select the owner of the application pool as described previously, and then click OK.
5. In the Properties dialog that appears, click OK to return to the Computer Management application.
6. Close the Computer Management application to return to the Windows desktop.

At this point, configuration of the new instance is complete; test the solution to ensure that it functions. In some cases, you may wish to rebuild search indexes using the Databases option in the Control Panel of the Sitecore browser-based desktop.

www.it-ebooks.info

CREATING A VISUAL STUDIO PROJECT

The ASP.NET Web Application project model is typically easiest for developers learning Sitecore. With this project model and the instructions provided in this section, Visual Studio stores all source code and project configuration files within the document root of the solution. Some developers prefer to store their project outside of the document root subdirectory, and deploy only the necessary components from that location to the document root during or after a build process. You can use that approach with this project model, or with alternate project models. For more about this topic, see the comments on my blog post at `http://bit.ly/oGYa80`.

Perform the following steps to create a Web Application project for a new Sitecore installation in Visual Studio 2010:

1. Click File ➪ New, ➪ Project. The New Project dialog appears as shown in Figure B-38.

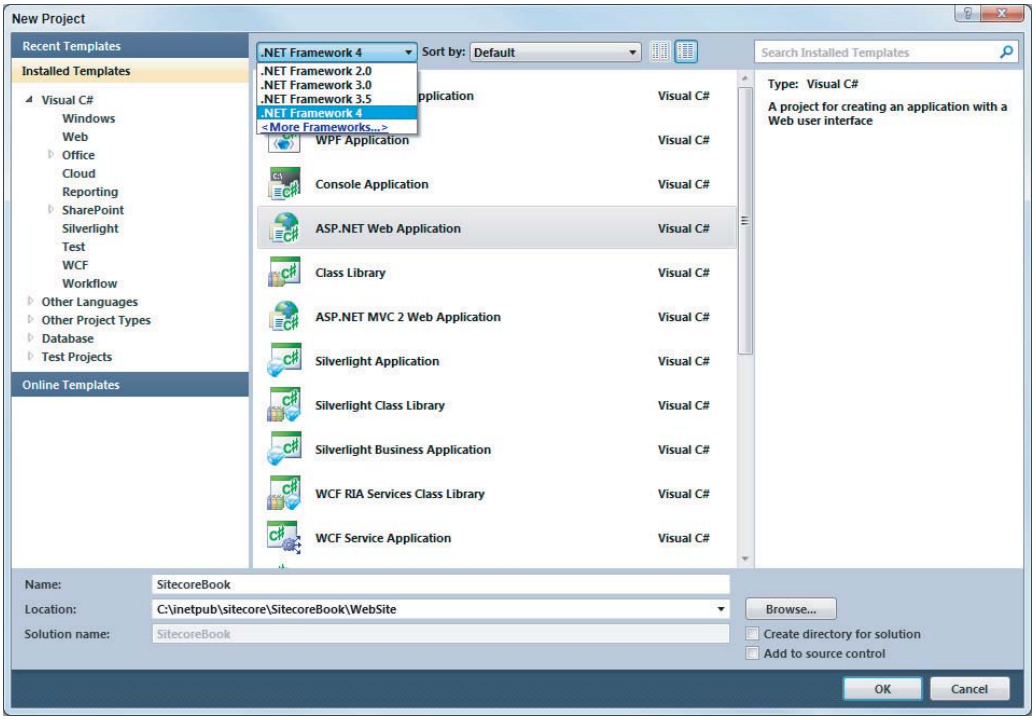


FIGURE B-38

2. Select .NET Framework 4 as shown in Figure B-38.
3. Select the ASP.NET Web Application project model as shown in Figure B-38.
4. Enter the name of the project in the Name field.
5. Enter the path to the Website subdirectory hosting the installation in the Location field.

www.it-ebooks.info

6. Clear the Create Directory For Solution checkbox, and then click OK. Visual Studio creates a subdirectory named after the name of the project within the `/Website` subdirectory.
7. Click File, and then click Close Solution.
8. Move the `/Properties` subdirectory, the `.csproj` file, and the `.csproj.user` file from the subdirectory named after the project created by Visual Studio within the `/Website` subdirectory up one level to the `/Website` subdirectory itself — for example, from the `C:\inetpub\sitecore\SitecoreBook\Website\SitecoreBook` subdirectory to the `C:\inetpub\sitecore\SitecoreBook\Website` subdirectory.
9. Delete the subdirectory named after the project created by Visual Studio — for example, the `C:\inetpub\sitecore\SitecoreBook\Website\SitecoreBook` subdirectory.

Perform the following steps to configure the Web Application project:

1. Click the File menu, and then click Open ➪ File/Solution. The Open Project dialog appears.
2. Select the `.csproj` file that Visual Studio produced when you created the project and you subsequently moved to the `/Website` subdirectory of the Sitecore installation, such as `C:\inetpub\sitecore\SitecoreBook\Website\SitecoreBook.csproj`.
3. Click the View menu, and then click Solution Explorer.
4. Expand `/Web.config`, right-click `/Web.config/Web.Debug.config`, and then click Delete. When Visual Studio prompts for confirmation, click OK. Repeat these steps for the `/Web.config/Web.Release.config` file, the `/About.aspx` file, the `/Site.Master` file, the `/Account` subdirectory, the `/App_Data` subdirectory, the `/Scripts` subdirectory, the `/Styles` subdirectory, and any other subdirectories Visual Studio included in the project by default. You can add these subdirectories to the project later if you need them.

You should not compile the `/Default.aspx` file at the document root into your project, but do not delete this file. To remove `/Default.aspx` from the project without deleting the file, right-click the `/Default.aspx` file in Solution Explorer, and then click Exclude From Project.

If you do not configure the web application using the `/Global.asax` file, you can exclude it from your project, but do not delete this file. If you do not exclude `/Global.asax` from your project, expand and delete the `/Global.asax/Global.asax.cs` file.

To add references to assemblies (`.dll` files) containing Sitecore and third-party APIs, perform the following steps in Visual Studio:

1. Right-click References in Solution Explorer to display the context menu.
2. Click Add Reference. The Add Reference dialog shown in Figure B-39 appears.

www.it-ebooks.info

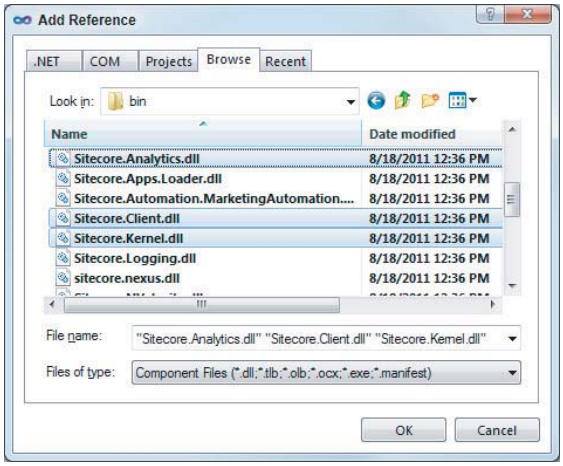


FIGURE B-39

3. Select the Browse tab, and then browse to the `/bin` subdirectory within the `/Website` subdirectory of the installation subdirectory, such as `C:\inetpub\sitecore\SitecoreBook\Website\bin`.

4. Select the assembly or assemblies, and then click OK. You see Solution Explorer.

5. Right-click the new assembly reference, and then click Properties. The Properties pane shown in Figure B-40 appears.

www.it-ebooks.info

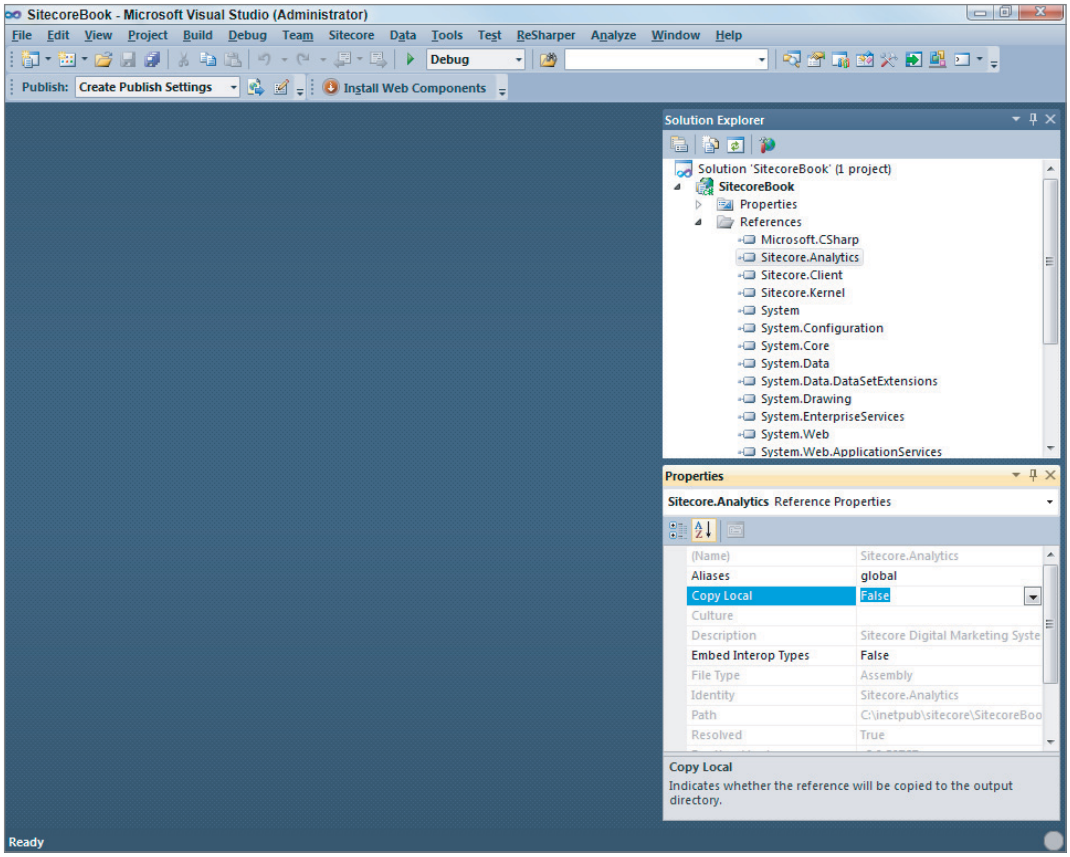


FIGURE B-40

6. Set the Copy Local property to False.



If you do not set the Copy Local property of a reference to an assembly in the `/bin` *subdirectory to False, Visual Studio may delete assemblies from that subdirectory when you compile your project, which will cause the instance to error.*

www.it-ebooks.info

Whenever you work with assemblies in Visual Studio, look for any broken references identified by triangular, yellow warning icons containing exclamation points as shown in Figure B-41 while you have References expanded in Solution Explorer.

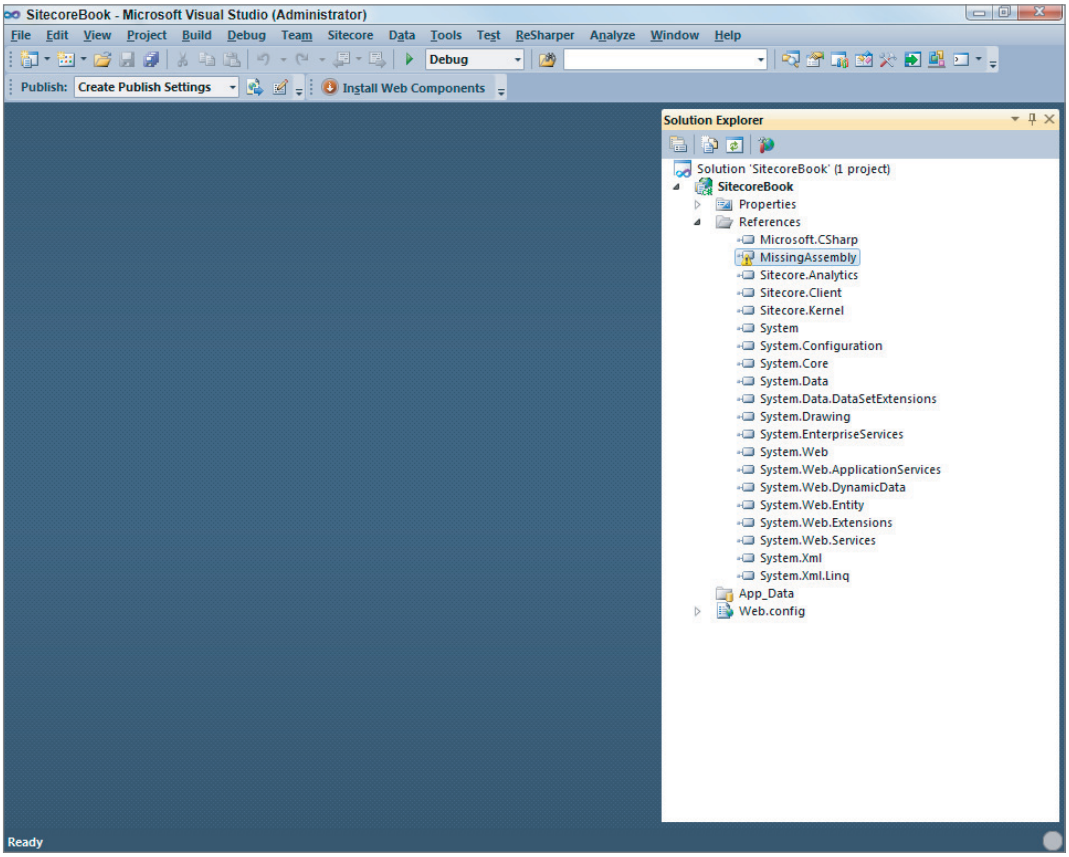


FIGURE B-41

To remove a broken reference, right-click that reference in Solution Explorer, and then click Delete.

Sitecore projects usually reference the `Sitecore.Kernel.dll`, `Sitecore.Client.dll`, `Sitecore.Analytics.dll`, and potentially additional Sitecore and third-party assemblies such as `HtmlAgilityPack.dll` and `Lucene.Net.dll`. All of these assemblies exist by default in the `/bin` subdirectory under the document root of the IIS website.

Perform the following steps to configure the assembly name and default namespace for the ASP.NET Web Application project:

1. Double-click Properties in Solution Explorer. The project properties dialog shown in Figure B-42 appears.

www.it-ebooks.info

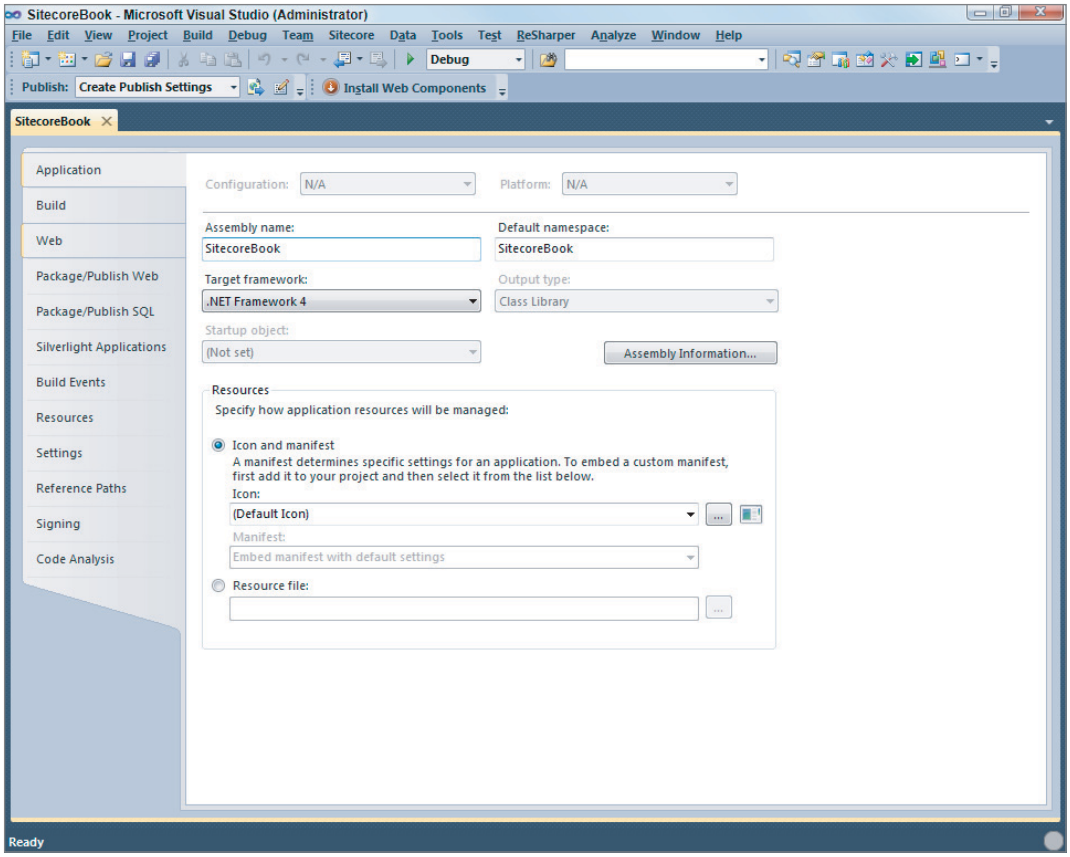


FIGURE B-42

2. Click the Application tab. For Assembly name, enter the name of the assembly, without the subdirectory or the `.dll` extension. When you compile your project, Visual Studio automatically copies the generated assembly to the `/bin` subdirectory within the `/Website` subdirectory.
3. Type the default namespace for the project in the Default namespace field.
4. Close the project properties dialog.

To add files and subdirectories that you use to the solution, such as the `/layouts` subdirectory and the `/xsl` subdirectory, or specific files and subdirectories from within such subdirectories, perform the following steps in Solution Explorer:

1. Select the project, and then click the Show All Files button shown in Figure B-43.

www.it-ebooks.info

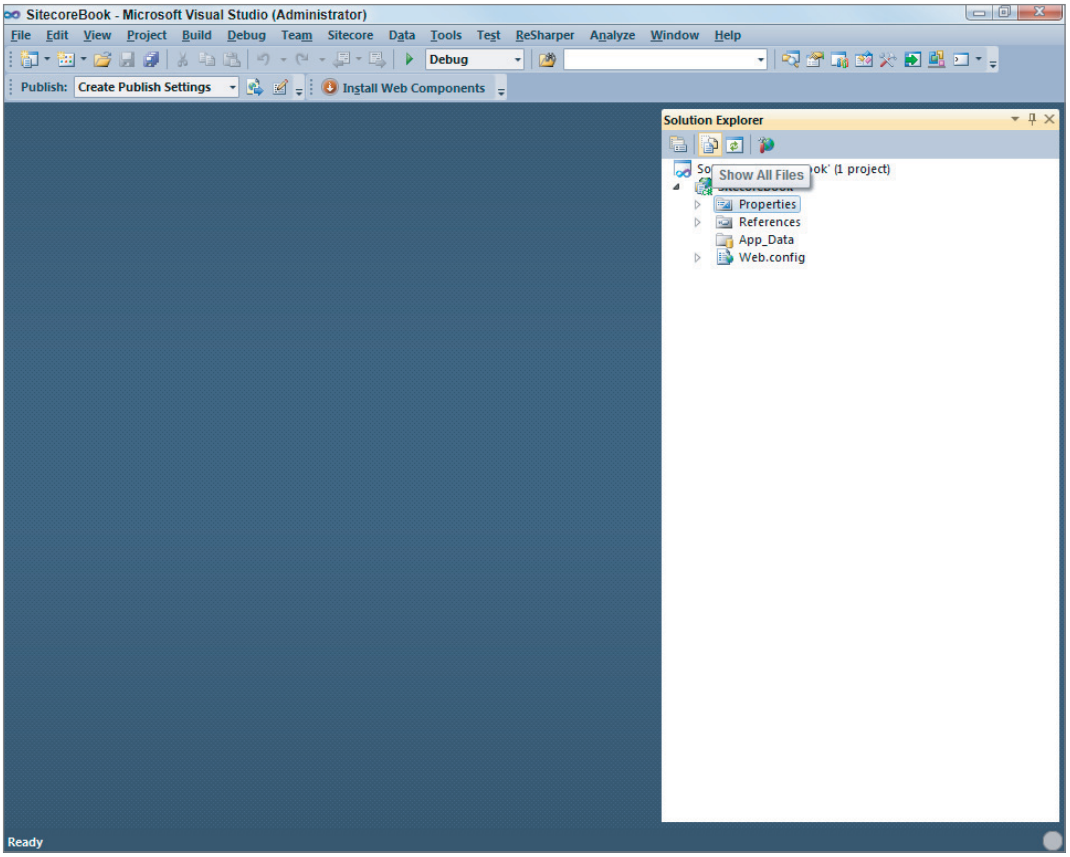


FIGURE B-43

2. Right-click each relevant subdirectory or file to display the context menu.
3. Click Include In Project. To exclude nested subdirectories and files, right-click them, and then click Exclude From Project.

When you close the project or exit the application, Visual Studio prompts you to create a solution. Allow Visual Studio to create the solution.



After including files and subdirectories in the project, click Show All Files as shown in Figure B-43. If you leave all files showing, debugging in Visual Studio may fail.

After you create the initial ASP.NET Web Application project, you can optionally create any number of class library projects within the solution to encapsulate specific functions, such as to separate

www.it-ebooks.info

libraries of XSL extension methods and web controls from the layouts, sublayouts, and code-behind of the solution.



I highly advise anyone using Visual Studio with Sitecore to install Sitecore Rocks, the free extension to Visual Studio for Sitecore developers. To download Sitecore Rocks, see `http://sitecorerocks.net`. *For more information about Sitecore Rocks, see my blog post at* `http://bit.ly/oZTaZI`.

TAKE HOME POINTS

Sitecore offers numerous installation and configuration options. Whether you install Sitecore using the setup program, a manual process, or the Sitecore Installer (`http://bit.ly/xr9jq2`) Shared Source module, and whether you use a remote database server or local databases, choose an installation strategy before installing Sitecore in any environment. Always read the current installation documentation and release notes before installing Sitecore.

Although it is impossible to provide comprehensive coverage of Sitecore development in a single resource, I hope this book has given you the fundamental information, and pointers to additional resources, that you need to work with and enjoy using this full-featured, extensible product for your own unique solutions.

www.it-ebooks.info

www.it-ebooks.info

INDEX

A

A/B testing, 392
A/B/N split testing, 388
`AbortPipeline()`, 287
access rights, 28, 124, 237
  applying, with APIs, 137–140
  commands, 253
  default, 125–126
  evaluating, 127–130
  inheriting, 127
  interpreting, with Access Viewer, 131–133
  roles, 419
  securing data template fields, 133–137
  securing Sitecore solutions, 418
  setting, 130–131
access rule definitions, 124
Access Viewer, 11, 131–133
`AccessControl` table, 124
action definition items, 266
Active Directory (AD), 124, 156, 157, 382, 395
ad hoc reports, 439
Adaptive Print Studio (APS), 381, 395–396
Add Language Wizard, 22
Add Service Reference Dialog, 344–345
Add Web Site dialog, 473
`addAspxExtension`, 407
`AddDatabase()`, 283
administration, of Sitecore solutions, 405–406
administrative rights, 419
administrator, running code as, 112
Adobe Flash, 221
Adobe InCopy, 396
Adobe InDesign, 395–396
Adobe Photoshop, 359
Adobe Reader, 447
Advanced Security Settings dialog, 481–482
`<agent>` elements, 277–281
agents, 147, 149–150, 196, 205, 277, 278–280
agile project management, 354, 358
AJAX. *See* Asynchronous JavaScript and XML
alias items, 38–39
`AliasResolver`, 38–39
Analytics fields, 43
AND. *See* Sitecore CMS Advanced .NET Developer
anonymous users, 145–146, 418
Apache log4net, 225–227
Apache Lucene, 16, 17, 109–113, 237, 488
Apache Solr, 110
API cookbooks, 446
APIs. *See* Application Programming Interfaces
Apple Safari, 4
application exception handlers, 186–187
application initialization
  cache memory allocation, 212
  data caches, 202
  hooks, 246, 282, 299
  item caches, 203
  media caches, 205
  prefetch cache loading, 200, 201
application pool, 16, 196, 473–475
  cycling, 200
  ownership, 480
  restarting, 279
  threshold warnings, 229
  Visual Studio debugging, 233

www.it-ebooks.info

Application Pool Identities, 476–478
Application Programming Interfaces (APIs), 4, 20, 28, 73
  applying access rights with, 137–140
  best practices, 407
  content, 70, 99, 446
  data definition, 20, 70, 99, 446
  fundamental, 99–107
  invoking directly, 336–338
  presentation component, 74, 77, 99, 161, 447
  security, 124, 155
  test data creation, 347–348
  testing without HTTP context, 338–342
Application Server Pages (ASP), 1, 13
APS. *See* Adaptive Print Studio
archiving, 467–469
The Art of Unit Testing (Osherove), 309
`.ashx` extension, 62
ASP. *See* Application Server Pages
ASP.NET, 1–4, 13, 15, 39, 62, 70, 71, 359
  controls, 81
  embedded test runners, 326
  error management, 161
  error management pages, 187–189
  exception levels, 163
  exception trapping, 175
  404 handling, 189, 196
  HTTP requests, 74
  infrastructure, 217
  lifecycle methods, 195, 207
  log-in controls, 155
  master pages, 120, 415
  membership, profile, and role providers, 156
  MVC, 74
  page lifecycle, 74–76, 209, 288
  postbacks, 212
  `renderLayout` manipulating control tree, 290
  scheduled processes, 278
  scheduling engine, 279
  security providers, 123–124, 246, 395
  session management, 219
  testing, 309–310
  testing time, 321
  trace, 230
  Web Application project, 484–485, 488, 490
  web user controls, 81
  worker process, 65
"ASP.NET Page Life Cycle Overview," 76
ASP.NET Web Forms, 76, 79–80, 262, 265
Assembly fields, 43
assertion methods, 314
Asynchronous JavaScript and XML (AJAX), 3
Attachment fields, 43
audience segmenting rules, 391
`Audit()`, 226
audit messages, 225
audit trail monitoring, 225
authentication, 124
  credentials, 8
  requiring, for managed website, 154–155
  switching provider, 157
  Windows, 458
Automated Publisher Sitecore Shared Source project, 281, 366
automated testing
  HTTP protocol, 317–322
  techniques, 307–308
availability, 216, 218, 223–224

B

back end integration, 282–299
Barracuda, 218
base templates, 21, 42
`BeginEdit()`, 104–105
BIG-IP, 218
branch templates, 30, 31
Browser Configuration Reference, 446, 451

C

C#, 112, 239, 323, 324
cache administration page, 212–213
cache ID, 208
cache key, 208

www.it-ebooks.info

cache statistics page, 202
`Cacheable` attribute, 207
`cacheMedia`, 205
caches
  application initialization, 200–203, 205, 212
  CDNs, 223
  clearing, 200, 212
  Client Data Store, 199
  data, 202, 203, 214, 215
  database, 199–204
  database path, 203
  disabling size limits, 214
  edge, 121, 197, 219, 222–223
  initialization, 212
  item, 202–203, 214, 215
  item path, 203–204
  layout engine, 214
  load-balanced environments, 212
  media, 69, 204–205
  output, 83, 205–211, 215–216
  path, 203–204
  prefetch, 200–202, 214, 215
  publishing, 209, 220
  purging, 203
  registry, 199
  security, 199
  sublayouts, 205
  tuning, 202, 423
  tuning sizes, 211–213
  utility, 199
  viewstate, 199
  web controls, 209–210
  XSL renderings, 199, 232–233
CacheTuner Sitecore Shared Project, 212
caching, 198–215
Caching Manager, 213
`Caching.AverageItemSize`, 203
`Caching.DefaultDataCacheSize`, 203
`Caching.DisableCacheSizeLimits` setting, 214, 423
`Caching.LoadFactor`, 203
Calendar, 381, 395, 396
campaign dashboards, 386
campaign management, 383, 390–391
`CanReadLanguage`, 260
`CanWrite`, 260
`CanWriteLanguage`, 260
`CanWriteWorkflow`, 260
Carousel, 11
Cascading Style Sheets (CSS), 3, 223, 359, 373
CDNs. *See* content delivery networks
CEP. *See* Customer Engagement Platform
CGI. *See* Common Gateway Interface
Checkbox field, 43
Checklist field, 44, 47
child sorting rules, 31–32
Cisco, 218
Citrix, 218
Citrix NetScaler, 222–223
Class fields, 43
`CleanupAgent`, 279
`CleanupEventQueue`, 279
`CleanupHistory`, 279
`CleanupPublishingQueue`, 279
Client Data Store cache, 199
client only installation, 465–466
client roles, 253
client RSS feeds, 120
`ClientFeeds.Enabled`, 120
`ClientFeeds.ItemExpiration`, 120
`ClientFeeds.MaximumItemsInFeed`, 120
clipboard, 435
clone items, 36–37, 49, 252–253
`CloneTo()`, 37
clustering, 222
CMS. *See* content management system
codebehind, 80, 81, 414–415
code-beside, 264–265
CodeGen Sitecore Shared Source project, 22
command templates, 30, 246
commands, 250–251. *See also specific commands*
  disabling for clones, 252–253
  Go To group, 258
  Page Editor, 246, 247, 300
  in ribbon, 254
  user access control, 253
  workflow, 370, 417

www.it-ebooks.info

Common Gateway Interface (CGI), 2
`CompactClientDataAgent`, 279
compile-time error detection, 186
complete installation, 461–463
component caching, 88
condition definition items, 266
configuration factory, 236, 246
  administration, 405–406
  type determination, 247–248
  Web.config, 247–248
`/configuration/sitecore` section, 236–238
content
  differentiation by language, 57
  importing, 70–71
  sharing, 69–70
  syndication, 118–120
content APIs, 70, 99, 446
content database, 100, 250
content delivery environment, 217–220, 223, 224, 419
content delivery load balancing, 220
content delivery networks (CDNs), 121, 197, 219, 222–223, 421
Content Editor, 4, 6, 8, 10–13, 43
  access rights and, 127, 130, 132, 134, 136
  caching settings, 206–207
  clipboard, 435
  clones, 36
  context site and database, 250
  debugger, 233, 255
  Developer tab, 258, 259
  extending, 246
  field validators, 273
  item validators, 277
  keyboard shortcuts, 433–434
  limiting UI, 426–427
  optimizing, 430–435
  performance optimization, 410, 433
  publishing, 221, 360–362
  raw field values, 434–435
  rendering parameters, 85–86
  ribbon, 53, 233, 251–253, 436–437
  role restrictions, 419
  save indicators, 286
  search box, 432
  Template Builder, 40
  usability, 424
  versions, 25
  warnings, 56, 246, 259–262, 267–269
content management environment, 419, 422
content management system (CMS), 1–3, 13, 19
  load balancing, 218–220
  preview functionality, 221
  project methodologies and practices, 354
content tree, 61, 342, 343
`ContentEditor.ShowGutter`, 431
`ContentEditor.ShowSearchPanel`, 431
`ContentEditor.ShowValidatorBar`, 431
context database, 100, 214, 250
context language, 58–59, 208, 411–412
context menus, 246, 259–260
context site, 250
context user, 112, 146, 192, 208, 348
Contextual Tab dialog, 252
control data, 22
Control Panel, 11, 23
cookbooks, 446
Copy Local property, 273
Core database, 43, 113, 224, 242, 258
  event handler processing, 283
  UI definitions, 250–251
`CounterDumpAgent`, 279
`CreateChildControls()`, 175, 184
CRM. *See* Customer Relationship Management
CRM Security Provider, 156
CSS. *See* Cascading Style Sheets
CSS selectors, 324
ctrl+right-click, 429–430
culture differentiation, 57, 412–413
custom applications, 246
custom business logic/objects, 359
custom editors, 246
custom field types, 46, 247
Customer Engagement Platform (CEP), 14, 43, 382
Customer Relationship Management (CRM), 124, 246, 386, 387
`CustomHandlers` processor, 237
CustomItemGenerator Sitecore Shared Source project, 22

www.it-ebooks.info

D

data access techniques, 107–118
data caches, 202, 203, 214, 215
data infrastructure, architecting, 406–414
data providers, 246
data template fields, 21, 42–49
  definition items, 40
  link management, 238
  metadata, 410
  properties, 46–49
  securing, 133–137
  types and categories, 42–46
data templates, 19–21, 36, 39–40
  best practices, 408–411
  custom tabs, 252
  designing, 408–411
  fields, 42–49
  inheritance, 53, 409
  output caching, 206
  prefetch cache settings, 201
  RAD, 356
  sections, 41–42
  standard values, 49–53
data validation, 55–56
database browser, 242–243
database caches, 199–204
Database Engine Configuration dialog, 457
database only installation, 464–465
database path caches, 203
`DatabaseAgent`, 278, 280
`DatabaseResolver` processor, 255
databases, 14–15
  backing up, 467
  clustering, 222
  content, 100, 250
  context, 100, 214, 250
  Core, 43, 113, 224, 242, 250–251, 258, 283
  DMS, 225
  internal links, 113–116, 237
  Master, 37, 100, 200, 202, 209, 220, 250, 283
  publishing target, 224
  relational, 20, 21, 196
  Restore Database dialog, 470–471
  scaling servers, 216, 221–222
  server scalability, 216, 221–222
Date field, 43
Datetime field, 43
`Debug()`, 226
Debug button, 253
Debug command, 253–255, 257, 258
debugger, 11, 74, 209, 415–416, 441
debugging
  in browser, 229–233
  in Content Editor, 233, 255
  IIS, 233
  in Microsoft Visual Studio, 233–234, 476
`defaultDevice`, 377
defaults, 20
definition items, 22, 29–30, 43, 56, 88–89, 118–119
  action, 266
  condition, 266
  configuration factory, 247
  data template fields, 40
  layout, 263
  rule, 266
  sublayout, 365
  UI, 250–251
dependency injection, 247, 248
deployments
  changes, 374
  creating, 373
  managing, 372–375
  production, 374
  QA, 373–374
Deprecated fields, 45
desktop, 6–9, 229, 250, 264, 427–429
DetailedConfigReport Sitecore Shared Source project, 234
Developer Center, 11, 74, 416
Developer tab, 258, 259, 441
developer training, 403, 444
Developer Types fields, 45
development environment, 3
Development Tools, 43, 373
`device`, 377
`DeviceResolver` processor, 79
devices, 78–79

www.it-ebooks.info

DFS. *See* Microsoft Distributed File System
dialog launchers, 251
Digital Marketing System (DMS), 4, 11, 13, 15, 43, 239, 289, 381
  campaign management, 390–391
  database server, 225
  edge caching and CDNs, 223
  email campaign manager, 392
  engagement analytics, 385–386
  engagement automation, 387
  features, 382–383
  installation strategies, 453
  pipelines, 298
  real-time personalization, 389–390
  terminology, 384
  testing, 388–389
  virtualization resource usage, 222
  visitor profiling, 388–389
disassembly, 238–240
`__DisplayName` field, 27
DMS. *See* Digital Marketing System
DNS. *See* Domain Name System
Document Object Model (DOM), 322
document root subdirectory, 16, 208, 239, 451
DOM. *See* Document Object Model
Domain Manager, 11
Domain Name System (DNS), 223
DOP. *See* Sitecore CMS .NET Developer Day of Prototyping
`DoRender()`, 83, 170, 171, 173, 180–181, 184, 195
Droplink field, 44, 45, 408
Droplist field, 44, 47, 408
Droptree field, 44, 45
dtSearch, 381, 395, 396–397
dynamic image manipulation, 121
dynamic links, 34, 415
Dynamic Segmentation, 383, 391

E

ECM. *See* Email Campaign Manager
eCommerce, 225
E-Commerce. *See* Sitecore E-Commerce Services
edge caching, 121, 197, 219, 222–223
Edit Frames, 13
Email Campaign Manager (ECM), 382, 391, 392
embedded applications, 10–12
embedded test runners, 325–334, 347
`EnableEventQueues` setting, 217
`EndEdit()`, 104–105
engagement analytics, 11, 382, 385–386
engagement automation, 382–383, 387
engagement intelligence, 385–386
Engagement Intelligence Analyzer, 386
engagement management, 382–383
engagement reporting, 385
engagement value, 385
Enterprise Resource Planning (ERP), 397
`Error()`, 168–169, 226
error controls, 170
error handling, 76, 169–185
error helper class, 163–166
error management, 161, 162
error management pages, 187–196
error pages, 163
`ErrorPage` setting, 187, 190
estimating hardware requirements, 404–405
ETag, 291–292
`Evaluate()`, 270
event handlers, 246, 299
  arguments, 282–283
  defining, 283–285
  load-balanced environments, 283
  performance impact, 287
  typical events, 286
event queues, 217
events, 286
everyone roles, 150–151
exceptions
  application level handling, 186–187
  error management pages, 187
  hiding information, 184–185
  logging, 226
  page level handling, 185–186
  security and information about, 188
  in sublayouts, 174–180

www.it-ebooks.info

trapping, 168–169, 175
in web controls, 180–184
exception classes, 167–168
exception levels, 163
exception management, 163
`Execute()`, 251, 252, 254, 258
`ExecuteRequest` processor, 189–190
`Exists()`, 140–141
expanding tokens, 50
`expandInitialFieldValue` pipeline, 288
eXtensible Stylesheet Language (XSL), 28, 33, 73, 412
extension classes, 246
extension methods, 249–250
external scheduling engines, 278

F

Facebook, 157
failover, 217, 219
fast query, 108
`Fatal()`, 226
fault tolerance, 218
federated authentication systems, 157
`FeedIsEmpty`, 261
Field Editor, 13
field validators, 271–274
`field:read` access right, 133, 136
`FieldRenderer` web control, 98
FieldValueComparer Sitecore Shared Source project, 32
`field:write` access right, 133, 136
File Drop Area field, 14, 45, 421
File field, 43, 46–47
file media, 219
filesystem permissions, 476, 478–480
`filterItem` pipeline, 289
final workflow states, 370
firewalls, 216
Form Data Viewer, 392
`FromName()`, 141
front-end markup integration, 359
Full Page XHtml validator, 56

G

General Link field, 45
GeoIP, 384, 390
`GetAccessRules()`, 138
`GetAllUsers()`, 141–142
`GetCacheKey()`, 210–211
`GetCachingID()`, 83, 195, 208, 211
`GetCachingKey()`, 84
`getChromeData` pipeline, 288, 300
`getContentEditorFields` pipeline, 288
`getContentEditorWarnings` pipeline, 260, 261–262, 288
`GetControl()`, 171, 174, 180
`GetDescendants()`, 107
`GetETag()`, 291
`GetItem()`, 90, 101, 107, 129, 192
`GetItems()`, 108
`getLookupSourceItems` pipeline, 288
`GetMaxValidatorResult()`, 271
`getMediaStream` pipeline, 288
`GetNonInheritedFieldIDs()`, 37
`GetNotFoundItem()`, 192
`GetOuterXml()`, 28, 349
`getPlaceholderRenderings` pipeline, 288
`getRenderingDatasource` pipeline, 288
`GetSelectedChildren()`, 336–338
`GetSourceItems()`, 119
`GetUser()`, 141
`GetUsers()`, 142
global conditional rendering rules, 91, 267
`global.asax`, 186
Globalization Control Panel, 23
globally unique identifier (GUID/ID), 22
Go To group, 258
Good Old Web Service, 246
Google+, 157
Google Analytics, 249
Google Chrome, 4
graphical user interface (GUI), 242, 356. *See also* user interfaces
Grouped Droplink field, 44, 47
Grouped Droplist field, 44, 47
GUI. *See* graphical user interface
GUID. *See* globally unique identifier

www.it-ebooks.info

H

HA. *See* high availability
HAP. *See* HTML Agility Pack
Hard Rock Web Service, 246
hardware capacity, 224–225, 404–405
hardware failure, 216, 218
`HasBeenCached()`, 210–211
`HasNoFields`, 260
health monitoring hooks, 282
health monitoring thresholds, 231
`healthMonitor` pipeline, 288
Hedgehog Development, 354, 375
hidden items, 407
hierarchical information architectures, 21
high availability (HA), 218, 223–224
home item, 36
Home tab, 251
hooks, 246, 282, 299
horizontal scaling, 216
HTML, 359
HTML Agility Pack (HAP), 239, 273, 318–320, 335
HTML editors, 437
HTML5, 3
`HtmlAgilityPack.dll`, 239, 319, 488
`HtmlCacheClearAgent`, 279
`HtmlDocument`, 320
HTTP 404 (not found) condition, 188, 189–196
HTTP 500 (server error) condition, 189
HTTP modules, 75–76
HTTP requests, 74–75, 77, 81, 222–223, 232
HTTP response, 75, 81
"HttpModules," 76
`httpRequestBegin` pipeline, 33, 35, 39, 58, 77, 79, 288
  configuration factory, 237
  context site settings, 379
  culture settings, 412–413
  `CustomHandlers` processor, 237
  `DatabaseResolver` processor, 255
  error management pages, 187
  `ExecuteRequest` processor, 189–190
  handling, 289–293
  `ItemResolver`, 192, 255
  nested processor elements, 247
  `NotFoundResolver`, 193–194
  processors, 289–290
  `SiteResolver`, 254
  `UserResolver` processor, 192
  uses, 290–291
`httpRequestEnd` pipeline, 229, 288

I

Icon field, 45
`Icons.CacheFolder`, 424
ID. *See* globally unique identifier
IDE. *See* integrated development environment
IDTable, 237
IFrame field, 45, 247
IIS. *See* Internet Information Server
IIS Manager, 424, 472, 475, 476
IL. *See* intermediate language
Image field, 43, 46–47
Improving .Net Application Performance and Scalability (Microsoft Press), 198
include files, 235–236
incremental publishing, 362
`inetmgr`, 189, 424, 475
`Info()`, 226, 230
information architecture, 20–22, 406–414
infrastructure
  data, 406–414
  network, 216
  presentation, 414–416
  scaling, 217
initial workflow state, 369
initialization hooks, 246, 282, 299
`Initialize()`, 282
`initialize` pipeline, 282, 288
InProc, 219, 220
insert options, 30–31, 411
`insertRenderings` pipeline, 288
Insite for Sitecore, 397
installation
  client only, 465–466
  complete, 461–463

www.it-ebooks.info

creating and configuring from archive, 469–483
database only, 464–465
preparation, 451–459
required information, 453
setup program, 405, 459–466
SQL Server, 454–457
strategies, 452–454
zip or other archive, 466–483
Installation Guides, 446
Installation Troubleshooting, 446
Installation Troubleshooting Guide, 451
Installation Wizard, 43, 459. *See also* Package Installation Wizard
Integer field, 43
integrated development environment (IDE), 11, 74, 313, 317, 416
intermediate language (IL), 239
internal links database, 113–116, 237
internal links management, 238
internationalization, 121
Internet Information Server (IIS), 1, 14–16, 62, 74–75, 161, 163
`.aspx` extension, 407
debugging, 233
document root directory, 451
error management pages, 189
404 handler, 189, 196
infrastructure, 217
media caches, 205
prefetch cache configuration, 200–201
stopping, 467
Web.config include files, 235
`IsClone()`, 37
`IsLocked`, 260
ISO format, 44
`IsReadOnly`, 260
items, 21. *See also* definition items
alias, 38–39
clone, 36–37, 49, 252–253
direct access, 107
hidden, 407
home, 36
IDs, 408
iterating, 423
languages, 22–26
managing URLs, 33–36
names, 22, 26
in prefetch caches, 200, 201
properties, 26–27
query access, 108
recursion for accessing, 116–117
required XML nodes, 349
revisions, 22–26
serialization, 372–373
sorting, 31–33
structure, 28
validating, 274–277
versions, 22–26
wildcard, 39
item caches, 202–203, 214, 215
item context menus, 259–260
item path caches, 203–204
item validator, 274–277
`item:added`, 286
`item:copied`, 286
`item:copying`, 286
`item:created`, 286
`item:creating`, 286
`item:deleted`, 286
`item:deleting`, 286
`item:moved`, 286
`item:moving`, 286
`ItemNotFound`, 260
`ItemNotFoundURL`, 189–190
`ItemPublishingRestricted`, 260
`item:Removed`, 286
`item:Removing`, 286
`item:renamed`, 286
`item:renaming`, 286
`ItemResolver` processor, 192, 255
`item:saved`, 247, 282, 286, 299
`item:saved:remote`, 283
`item:saving`, 282, 283, 285–286, 299
`item:sortorderchanged`, 286
`item:templateUpdated`, 286
`item:versionAdded`, 286
`item:versionAdding`, 286
`item:write` access right, 133

www.it-ebooks.info

J

Java, 13, 323
JavaScript, 223, 323–325, 359, 373
JetBrains, 317
`job` pipeline, 288
jQuery, 359

K

Kapow Software, 71
keyboard shortcuts
    Content Editor, 434–435
    RTE, 438–439
    Sitecore, 435–437
KeyCode, 435–437
keywords, in prefetch caches, 201

L

landing workflow, 371
`languageEmbedding`, 407
LanguageFallback Sitecore Shared Source project, 59
LanguageFallbackItemProvider Sitecore Shared Source project, 59
Language-Integrated Query (LINQ), 33, 338
`LanguageResolver` processor, 58
languages
    alternative, 59
    best practices, 411–413
    content differentiation, 57
    context, 58–59, 208, 411–412
    determining context, 58–59
    falling back, 59
    information architecture, 412
    managing, 56–59
    registering, 23
    versions, 22–26
layout definition items, 263
layout deltas, 78
layout details, 78, 206
Layout Details dialog box, 206–207
layout engine, 73
    ASP.NET Web Forms, 76
    best practices, 120–121
    caches, 214
    conditional rendering, 91
    content delivery environment, 220
    sublayout data sources, 416
    sublayout exception handling, 175
    XSL rendering error handling, 170
`LayoutNotFoundUrl` setting, 190
layouts, 79–80. *See also* sublayouts
    deploying, 373
    placeholders, 356
    XSL files, 170
LDAP, 157
Library Sitecore Shared Source project, 250
Licenses command, 262–265
Licenses dialog, 264, 265
link provider, 34–35
Link Types fields, 45
LinkedIn, 157
LinkProvider Sitecore Shared Source project, 35, 38, 407
LINQ. *See* Language-Integrated Query
List Types fields, 44
load balancing, 216, 218–220
    content delivery, 220
    content management, 218–220
    event handling, 283
    hardware, 218
    high availability, 218, 223
    media files, 219
    publishing, 218, 221
    software, 218
    test environments, 219
load-balanced environments, 212
localization, 121
locks, 368
log files, 225–227, 242, 286, 460
`LogFileAppender`, 225–226
`LogFolder` setting, 225
login options, 6, 7, 425
`loginPage` attribute, 154–155
Lucene, 16, 17, 109–113, 237, 488
`Lucene.Net.dll`, 488

www.it-ebooks.info

M

managed website, 236
  attributes, 375–377
  default, 378
  multiple, 375–379, 411
  output caches, 207–209
  requiring authentication for, 154–155
  `Sitecore.Sites.SiteContext` representing, 249
  Twitter account association, 249
Management Studio, 454, 458, 467–469, 470
manual configuration, 469
Marketing Center, 10, 387, 389, 390, 391
Marketing Performance Quickview, 386, 391
Master database, 37, 100, 200, 202
  content delivery environment references, 220
  as context database, 250
  event handler processing, 283
  publishing changes from, 209
masters, 13, 31
`__Masters` field, 48, 51
MaxMind, 384
`MaxTreeDepth`, 406
media
  load balancing files, 219
  managing, 60–68
  path provider, 68
  publishing, 366–367
  settings, 64
  types, 66–68
  URLS, 61–64, 421
media caches, 69, 204–205
media library, 10, 60, 415, 420–423
  CDNs, 223
  configuring, 64–65
  structure, 61
media path provider, 237
media upload watcher, 65, 422–423
`Media.CacheFolder`, 204–205
`Media.CachingEnabled`, 205
`Media.DisableFileMedia`, 219
`MediaFolder` setting, 65
`Media.MediaLinkPrefix`, 61–62
MediaUrlTransformer Sitecore Shared Source project, 62, 296, 421
`Media.useItemsPaths`, 64
membership providers, 155, 156
memory monitoring hooks, 282
merge replication, 224
Message Transfer Agent (MTA), 392
metadata, 22
  data template fields, 410
  in prefetch caches, 201
method rendering, 82, 83, 90
Microsoft Distributed File System (DFS), 219
Microsoft Dynamics, 124
Microsoft Internet Explorer, 4, 425
Microsoft Network Load Balancing (MS NLB), 218
Microsoft Office, 12, 424
Microsoft Outlook, 390
Microsoft SQL Server, 14, 108, 217, 224, 236, 398, 451, 453
  backing up databases, 467
  configuring, 458–459
  installing, 454–457
Microsoft SQL Server Express 2008, 454, 462
Microsoft Unity Application Block, 248
Microsoft Visual Studio, 4, 11, 14, 17, 74, 242, 246, 261, 273, 451
  broken references, 488
  creating project, 484–491
  debugger, 233–234, 476
  deployment creation, 373
  test projects, 312–313
  testing extensions, 317
Microsoft WebDeploy, 219
Microsoft Windows, 1, 12, 14, 196, 217, 251, 424
Microsoft Windows Azure, 14, 217, 382, 395, 398
Microsoft Windows Server 2008, 405
Microsoft Windows Update, 457
MIME. *See* Multipurpose Internet Mail Extensions
MIME types, 68
minification, 223
`mode` attribute, 188
Mozilla Firefox, 4, 323–325

www.it-ebooks.info

MS NLB. *See* Microsoft Network Load Balancing
MSBuild, 340
MTA. *See* Message Transfer Agent
Multi-Line Text field, 43, 410
multilingual solutions
    best practices, 411–413
    managing, 56–59
Multilist field, 44, 47, 273
multiple servers, 216
multiple sites, 375–379
Multipurpose Internet Mail Extensions (MIME), 68
multivariate tests (MV tests), 388

N

Name Lookup Value List field, 44, 47
name validation, 27
Name Value List field, 44
namespaces, 313, 319
nested roles, 151–152
.NET Framework, 1, 13, 33, 168. *See also* ASP.NET
    extension classes, 246
    extension methods, 249–250
    selecting version, 460, 463, 475, 484
    types, 247
    web drivers, 323, 324
.NET Reflector, 239
network infrastructure capacity, 216
Network Service, 16, 477, 480
`NeverPublish`, 260
NewsMover Sitecore Shared Source project, 287
`NoAccessUrl` setting, 190
`NoLicenseUrl` setting, 190
`NotFoundAttribute` constant, 192
`NotFoundResolver`, 193–194
notification engine, 261
`Notifications`, 261
NUnit, 311–315, 317
    embedded test runner, 326, 331–334
    tests over HTTP, 321
`NUnit.Core.EvenListener`, 331
`NUnit.Framework`, 343
`NUnit.Framework.Assert.AreEqual()`, 314
`NUnit.Framework.Test`, 314
`NUnit.Framework.TestFixture`, 321
`NUnit.Framework.TextFixture`, 314

O

object-oriented coding techniques, 358–359
Ohloh, 317
OLAP. *See* online analytical processing
OMS. *See* Online Marketing Suite
`OnInit()`, 211
`OnItemSaved()`, 283
online analytical processing (OLAP), 386
Online Marketing Suite (OMS), 13, 382, 384. *See also* Digital Marketing System
Operations group, 251
optimization, 387–388
Oracle, 14, 108, 217, 236, 398
order management systems, 202
Osherove, Roy, 309
Out of Office Wizard, 390
output caching, 83, 205–211, 215–216
overlay icons, 439–440

P

Package Designer, 12, 373, 374
Package Installation Wizard, 373–374
Page Editor, 4–5, 13–14, 74, 221, 258, 377, 425, 427
    commands, 246, 247, 300
    context site and database, 250
    dynamic link expansion, 415
    email creation, 392
    extending, 299–305
    page mode, 98–99
    personalization, 390
    resetting fields, 300–305
    role restrictions, 419
    usability, 424

www.it-ebooks.info

page hit counters, 209
page lifecycle
    ASP.NET, 74–76, 209, 288
    initialization phase, 209
    Sitecore API without HTTP context, 342
page mode, 98–99
page reload, 212
page-level exception handlers, 185–186
PageNotFound Sitecore Shared Source project, 196, 250, 293
`Paste()`, 350–351
`patch:` prefix, 235, 236
`patch:after`, 236
`patch:attribute`, 236
`patch:before`, 236
`patch:delete`, 236
`patch:instead`, 236
path cache, 203–204
Path command, 259
performance
    cache size tuning, 211–213
    Content Editor, 410, 433
    event handlers, 287
    item IDs, 408
    maximizing, 423–424
    optimizing Sitecore, 198–215
    publishing, 221
    threshold warnings, 229, 231
    versions, 418
Performance Monitor Users group, 482
periphery networks, 217
personalization, 91–97, 383, 389–390
PIM. *See* product information management
pipeline processors, 287, 299
pipelines, 245. *See also specific pipelines*
    DMS, 298
    invoking, 247
    order of processors, 261
    processing, 287–298
    types, 287–289
    UI, 289
placeholders, 80, 356, 414–415
PNUnit Runner, 311
polling frequency, 279–280
prefetch caches, 200–202, 214, 215
`preprocessRequest` pipeline, 75, 288, 290
presentation components, 78–99
    APIs, 74, 77, 99, 161, 447
    caching, 415
    data sources, 90–91
    devices, 78–79
    exception management, 163
    layouts, 79–80
    nesting, 416
    output caching configuration, 206
    personalization, 390
    placeholders, 80, 356
    XML, 262–263, 265
presentation controls, 81–90
    cache IDs, 208
    cache keys, 208
    caching options, 207
    error handling, 169–185
    404 response, 194–195
    hiding errors, 184–185
    instantiating, 308
presentation infrastructure, 414–416
presentation technology, 90
Preview, 11, 221, 250
Print Studio Editor, 396
`Process()`, 287
processing pipelines, 287–298
product information management (PIM), 202
production content delivery, 3
production content management, 3, 219
production environment, 3, 374
Profile group, 229
profile providers, 155, 156
profiling, 229, 383, 388–389
project approaches, 354–359
    custom business logic/object integration, 359
    front-end markup integration, 359
    object-oriented coding techniques, 358–359
    prototyping, 354–358
prototyping
    Sitecore and RAD, 354–355
    Sitecore RAD, 355–358
    traditional RAD, 355

www.it-ebooks.info

publish pipeline, 237, 288
Publish Wizard, 360–361
PublishAgent, 279, 281
publish:begin, 286
publish:end, 236, 286
publish:end:remote event, 215, 236
publish:fail, 286
publishing, 37
  automating workflow, 417–418
  caches, 209, 220
  Content Editor, 221, 360–362
  incremental, 362
  instance settings, 221
  load balancing, 218, 221
  media, 366–367
  modes, 362–363
  output caches, 209, 220
  performance, 221
  preproduction environments, 365–366
  preview functionality, 221
  republishing, 362
  restrictions, 260, 363–364
  scaling, 220–221
  scheduling, 221, 366
  security information, 220
  Sitecore, 360–367
  smart, 362–363
  target databases, 224
  targets, 364–365
  triggering operations, 220
PublishingInstance, 221
Publishing.PublishingInstance, 219
publishItem pipeline, 367, 421
publishItems pipeline, 288

Q

QA environment, 373–374
queries, 108. *See also specific query strings*
Query Analyzer, 373
Query.MaxItems, 108
QueryState(), 252, 258

R

rapid application development (RAD), 354–358
raw field values, 434–435
Really Simple Syndication (RSS), 29, 118–120, 261, 417
real-time feedback loops (RTFLs), 392
real-time personalization, 383, 389–390
Rebuild(), 115
Rebuild Search Index, 110
recursion, 116–117
Recycle Bin, 11, 13
RedGate, 239
Reflector, 239–242
region differentiation, 57
registry cache, 199
relational data, 413–414
relational databases, 20, 21, 196
remote contend delivery instances, 209
RemoveInvalidAccessRulesTask.cs, 147–149
Render(), 76, 83–84, 170, 180, 195
RenderAsText(), 335
RenderChromeData processor, 300
renderField pipeline, 34, 74, 98, 121, 288
  adding commands, 303
  chrome, 300
  processing, 293–296
rendering
  conditional, 91–97
  method, 82, 83, 90
  parameters, 84–87, 415
  statistics, 227–228
rendering parameters templates, 84–87
renderings, 81–90. *See also* XSL renderings
  caching, 205–209, 214, 215
  GetCachingID(), 208
  optimizing, 209
  URL, 82–83, 90
__Renderings field, 51
Rendering.SiteResolving, 34, 36
renderLayout pipeline, 288
  ASP.NET control tree, 290
  intercepting, 296–298
republishing, 362

www.it-ebooks.info

RequestErrors.UserServerSideRedirect, 189–190
requirements analysis, 403–404
Reset Fields, 51–52
ReSharper, 317
Restore Database dialog, 470–471
return on investment (ROI), 12, 359, 417
__Revision field, 26
ribbon, 246
  commands, 254
  Content Editor, 53, 233, 251–253, 436–437
  extending, 251–259
Rich Text Editor (RTE), 46, 121, 246
  field validators, 271
  item validators, 274
  keyboard shortcuts, 438–439
  optimizing, 437–439
  profiles, 438
Rich Text fields, 34, 43, 46, 56, 113
ROI. *See* return on investment
role management switching provider, 158
Role Manager, 11
role providers, 155, 156
roles, 146–152, 253, 419
RSS. *See* Really Simple Syndication
RTE. *See* Rich Text Editor
RTFLs. *See* real-time feedback loops
rule definition items, 266
Rule Set Editor, 92–95, 267–269
rules
  access, 124, 138
  audience segmenting, 391
  child sorting, 31–32
  definition items, 266
  global conditional rendering, 91, 267
Rules Editor, 390
rules engine, 13, 27, 91, 96, 246, 266, 299
  Content Editor warning generation, 261, 267–270
  workflows, 371
Rules field, 45, 266
RunRules, 261
runtime errors, in XSL renderings, 169–174

S

SAC. *See* Sitecore App Center
saveUI pipeline, 286
scaling, 216–225
  content delivery networks, 222–223
  database servers, 216, 221–222
  edge caching, 222–223
  hardware capacity planning, 224–225
  horizontal, 216
  hosting infrastructure, 216
  infrastructure, 217
  load balancing, 216, 218–220, 221, 283
  publishing, 220–221
  vertical, 216
  virtualization, 222
sc_database query string, 255, 258
sc_debug query string, 254
sc_device query string, 79, 377
sc:field(), 239
sc:HasVersions(), 412
scheduled agents, 147, 150, 205, 278–280
scheduled processes, 246, 277–281
scheduled tasks, 278
scheduling engine, 278, 279
$sc_item parameter, 90
sc_itemid query string, 255
sc_lang query string, 58–59, 255
SCMS. *See* source code management system
sc_prof query string, 254
sc_ri query string, 209, 254
Scrum methodology, 354, 358
sc:trace(), 230
sc_trace query string, 254
SDF. *See* Sitecore CMS Developer Foundations
SDN. *See* Sitecore Developer Network
search engine optimization (SEO), 11, 38, 394, 406
Search Engine Optimization Toolkit (SEO), 382, 395, 397–398
search indexes, 110–113
search pipeline, 289
security
  API, 124, 155
  best practices, 418–420

www.it-ebooks.info

content delivery environment, 220
disabling, 154
exception information, 188
integration and extension, 155–157
preventing application, 153–154
providers, 123–124, 156, 246, 395
__Security, 124
security accounts, 140–153
security caches, 199
security context, 192
security domains, 140, 152–153
Security Editor, 11, 127, 130–131, 132, 135
Security Tools, 130
SEFE. *See* Sitecore E-Commerce Fundamentals Edition
segmentation, 383, 391
selectedChildren, 336
selection field types, 47
SelectItems(), 108
SelectSingleItem(), 108
Selenium, 323–324
SEO. *See* search engine optimization; Search Engine Optimization Toolkit
serialization, 372–373, 375
server affinity, 219, 220
service proxy class, 345
SES. *See* Sitecore E-Commerce Services
sessionEnd pipeline, 288
SetAccessRight(), 139–140
SetAccessRules(), 140
setup program, 405, 459–466
shared fields, 48
SharePoint Integration Framework (SPIF), 382, 395, 397
sharing content, 69–70
shell, 250
Show group, 259
ShowingInputBoxes, 261
Simple Types fields, 43–44
single sign-on (SSO), 291, 395
Single-Line Text field, 43
SIP. *See* Sitecore Intranet Portal
site media caches, 204–205
site output caches, 83, 205–211
Sitecore, 1
administering, 405–406
archiving solution, 467–469
ASP.NET CMS version history, 13–14
ASP.NET page lifecycle extensions, 76
caching, 198–215
company history, 13
components, 14–17
configuration factory, 236
content syndication with, 118–120
content tree, 61, 342, 343
context, 77
control flow, 309
data access techniques, 107–118
data subdirectory, 16
database browsing, 242–243
databases, 14–15
default managed sites, 378
deployment management, 372–375
desktop, 6–9, 229, 250, 264, 427–429
developer training, 403
devices, 78–79
disabling security, 154
document root subdirectory, 16
documentation, 446–447
error handling, 76
event queues, 217
exception levels, 163
extending, 245–247
extension methods, 249–250
facilities, 3–4
fast query, 108
fundamental APIs, 99–107
ID constants, 106–107
IIS hosting, 15–16
impersonating user, 153
implementation management, 403–404
keyboard shortcuts, 435–437
log files, 225–227, 242
logging in, 4–10, 425
login options, 6, 7, 425
managing multiple sites, 375–379
media folders, 61
optional modules, 395–399

www.it-ebooks.info

performance optimization, 198–215
placeholders, 80
preventing security application, 153–154
project approaches, 354–359
query access, 108
references, 488
removing full path from URLs, 36
scheduling processes, 277–281
securing solutions, 418–420
security integration and extension, 155–157
service proxy class, 345
setup program, 405, 459–466
testing complexities, 309–310
testing techniques, 310–311
training, 403, 444
UI framework, 250–266
upgrading, 374
URL generation, 34–35
URL resolution, 35–36
usability optimization, 424–425
users, 140–146
virtualizing, 197, 214, 222, 224
Visual Studio project, 17
web services, 118
WebDAV implementation, 66
working with, 12–13
Sitecore App Center (SAC), 11, 381, 394
Sitecore Azure, 217, 382, 394, 395, 398–399
Sitecore Best Practices Blogs, 402
Sitecore blogs, 447
Sitecore Browser Configuration Reference, 425, 433
Sitecore Cache Configuration Reference, 211, 213, 447
Sitecore Client Configuration Cookbook, 45, 120, 152, 300, 408, 417, 447
Sitecore Client Publishing role, 419
Sitecore Client roles, 151, 152
Sitecore Client RSS feeds, 14
Sitecore CMS Advanced .NET Developer (AND), 403, 444
Sitecore CMS Developer Foundations (SDF), 403, 444
Sitecore CMS .NET Developer Day of Prototyping (DOP), 403, 444
Sitecore CMS Performance Tuning Guide, 447
Sitecore CMS Website .NET Developer (SND), 403, 444
Sitecore Content API Cookbook, 70, 99, 446
Sitecore Content Author's Cookbook, 446
Sitecore Content Cookbook, 446
Sitecore Content Reference manual, 365, 446
Sitecore Customer Service, 448
Sitecore Data Definition API Cookbook, 20, 70, 99, 446
Sitecore Data Definition Cookbook, 20, 84, 410, 446
Sitecore Data Definition Reference, 20, 108, 410, 446
Sitecore demo sites, 447–448
Sitecore Developer Network (SDN), 13, 46, 108, 116, 124, 242, 287, 338, 362, 416, 443, 445–447, 451
Sitecore Dynamic Links Guide, 407
Sitecore E-Commerce, 248
Sitecore E-Commerce Fundamentals Edition (SEFE), 397
Sitecore E-Commerce Services (SES), 381, 395, 397
Sitecore Explorer, 349, 373
Sitecore Foundry, 381, 394
Sitecore Guide to Dynamic Links, 75
Sitecore Guide to Reusing and Sharing Data, 22, 70, 412
Sitecore Guide to Troubleshooting Presentation Components, 74
Sitecore Installation Guide, 222, 405, 446, 451, 459, 466, 476
Sitecore Installer Shared Source module, 451
Sitecore Intranet Portal (SIP), 381, 393–394
Sitecore Limited Content Editor role, 419
Sitecore Limited Page Editor role, 419
Sitecore Package Designer Administrator's Guide, 43, 373
Sitecore Partner Network, 447
Sitecore Partners, 444–445
Sitecore Portal, 445–446

www.it-ebooks.info

Sitecore Presentation Component API Cookbook, 74, 77, 99, 161, 447
Sitecore Presentation Component Cookbook, 74, 447
Sitecore Presentation Component Reference, 74, 78, 88, 91, 211, 447
Sitecore Presentation Component XSL Reference, 82
Sitecore RAD, 355–358
Sitecore Recommended Practices, 403
Sitecore Rocks, 4, 11, 14, 17, 74, 242, 246, 349, 373, 374, 413, 416, 439, 491
Sitecore Rules Engine Cookbook, 96, 269
Sitecore Scaling Guide, 15, 209, 215, 217, 219, 220, 222, 447, 453, 471
Sitecore Security Administrator's Cookbook, 124
Sitecore Security API Cookbook, 124, 155
Sitecore Security Hardening Guide, 220, 420
Sitecore Security Reference, 124
Sitecore Serialization Guide, 373
Sitecore Shared Source, 447
    Automated Publisher, 281, 366
    CacheTuner, 212
    CodeGen, 22
    CustomItemGenerator, 22
    DetailedConfigReport, 234
    FieldValueComparer, 32
    LanguageFallback, 59
    LanguageFallbackItemProvider, 59
    Library, 250
    LinkProvider, 35, 38, 407
    MediaUrlTransformer, 62, 296, 421
    NewsMover, 287
    PageNotFound, 196, 250, 293
    Sitecore Installer, 451
    Sitecore Stuff, 271
    SublayoutParameterHelper, 90, 416
    SublayoutRenderingType, 179–183
    VersionManager, 418
    XmlImporter, 71
    XslHelper, 82
Sitecore Social Connected, 157
Sitecore Stuff Shared Source project, 271
Sitecore Success Services, 443, 449
Sitecore Support Portal, 445, 448–449
Sitecore Symposium, 443, 449
Sitecore Workflow Cookbook, 367
Sitecore Workflow Reference, 367
Sitecore XSL Presentation Component Reference, 239
`SitecoreBook.Tasks.PreventSaveEventHandler`, 283
`Sitecorebook.Tests`, 312
`SitecoreBook.Web.UI.WebControl`, 211
`SitecoreBook.Web.UI.WebControls.PerformsExpensiveInitialization`, 209–211
`Sitecore.Configuration.Factory`, 99–100
`Sitecore.Configuration.Factory.CreateItemNavigator()`, 28
`Sitecore.Context`, 240–241
`Sitecore.Context.ContentDatabase`, 250
`Sitecore.Context.Database`, 250, 334–335
`Sitecore.Context.Item`, 26, 33, 335
`Sitecore.Context.Items`, 39
`Sitecore.Context.PageMode`, 299
`Sitecore.Context.RawUrl`, 39
`Sitecore.Context.Site`, 191, 291
`Sitecore.Data.Database`, 100, 107, 108, 129, 192
`Sitecore.Data.Fields.Field`, 53, 103–104
`Sitecore.Data.ID`, 104
`Sitecore.Data.Items`, 22
`Sitecore.Data.Items.CloneItem`, 37
`Sitecore.Data.Items.CustomItem`, 105–106
`Sitecore.Data.Items.CustomItemBase`, 22, 415
`Sitecore.Data.Items.EditContext`, 104–105
`Sitecore.Data.Items.Item`, 28, 33, 37, 101–102, 107, 108, 348
    caching, 202
    extending, 249
    `Paste()` method, 350–351
`Sitecore.Data.Validators.BaseValidator`, 270–271
`Sitecore.Data.Validators.StandardValidator`, 270–271
`Sitecore.Data.Validators.ValidatorResult.Valid`, 270
`Sitecore.Diagnostics.Log`, 168–169, 226

www.it-ebooks.info

Sitecore.Diagnostics.Tracer, 230
Sitecore.Events.Event.ExtractParameters(), 283
Sitecore.Events.SitecoreEventArgs, 282, 283
Sitecore.FieldIDs, 106, 408
Sitecore.FieldIDs.DisplayName, 27
Sitecore.FieldIDs.Revision, 26
Sitecore.FieldIDs.Security, 124
Sitecore.FieldIDs.Sortorder, 31
Sitecore.FieldIDs.Source, 37
Sitecore.Globals.LinkDatabase.GetReferrers(), 114
Sitecore.Globals.Load(), 342
Sitecore.Globals.TaskDatabase, 281
SitecoreInstaller.log, 460
Sitecore.ItemIDs, 26, 106, 408
Sitecore.Kernel.dll, 239, 254
Sitecore.Links.LinkManager.ExpandDynamicLinks(), 34
Sitecore.Links.LinkManager.GetItemUrl(), 35
Sitecore.Pipelines.Loader.LoadHooks, 282
Sitecore.Pipelines.PipelineArgs, 287
Sitecore.Pipelines.RenderField.RenderFieldArgs, 293
Sitecore.Resources.Media.MediaUrlOptions, 62, 63
Sitecore.Security.AccessControl.AccessRight, 126
Sitecore.Security.AccessControl.AccessRuleCollection, 137–138
Sitecore.Security.AccessControl.AuthorizationManager.IsAllowed(), 192
Sitecore.Security.AccessControl.ItemSecurity, 138
Sitecore.Security.Accounts.Role, 146
Sitecore.Security.Accounts.User, 140–141
Sitecore.Security.Accounts.UserSwitcher, 112, 420
Sitecore.SecurityModel.SecurityDisabler, 112, 348, 420
Sitecore.Shell.Framework.Commands.Command, 251, 252, 254, 257–258
Sitecore.Shell.Framework.Commands.CommandContext, 252
Sitecore.Shell.Framework.Commands.CommandState, 252, 258
Sitecore.Shell.Framework.Commands.Delete, 251
Sitecore:Shell.Framework.Commands.System.Debug, 254
Sitecore.Sites.SiteContext, 191, 192, 249, 291
Sitecore.Syndication.PublicFeed, 120
Sitecore.Tasks.CounterDumpAgent, 226
Sitecore.TemplateIDs, 26, 107, 408
Sitecore.Web.Application, 75
Sitecore.Web.HttpApplication, 75
Sitecore.Web.UI.HtmlTextWriter, 170
Sitecore.Web.UI.SublayoutRenderingType, 180, 248
Sitecore.Web.UI.WebControl, 83, 184, 211
Sitecore.Web.UI.WebControls.FieldRenderer, 34
Sitecore.Web.UI.WebControls.Sublayout, 83
Sitecore.Web.UI.WebControls.XslControlRenderingType, 171
Sitecore.Web.UI.WebControls.XslFile, 83, 170, 171, 173
Sitecore.Xml.XPath.ItemNavigator, 28
Sitecore.Xml.Xsl.XslHelper, 239
site:enter access right, 137, 138–139
SiteResolver processor, 254, 378–379
smart publishing, 362–363
SND. *See* Sitecore CMS Website .NET Developer
Solr, 110
Solution Explorer, 313, 323, 340, 416, 485, 488–489
__Sortorder, 31
__Source, 37, 49
source code management system (SCMS), 234, 263, 358
Source field, 118–119
Source property, 46–47
SPIF. *See* SharePoint Integration Framework
SQL Azure, 398
SQL Server. *See* Microsoft SQL Server

www.it-ebooks.info

SQL Server Database Engine service, 457
SQL Server Installation Center, 455
Squid, 223
SSO. *See* single sign-on
stack traces, 242
staging environment, 3
standard templates, 21, 31, 37, 53–54, 419
standard values, 21, 31, 49–53, 411
startup time, 214
`statusCode` attribute, 188
sticky sessions, 219
`StopMeasurements` processor, 229
`StreamReader`, 319
subdirectories, 451, 454
`__Subitems Sorting`, 51
SublayoutParameterHelper Sitecore Shared Source project, 90, 416
SublayoutRenderingType Sitecore Shared Source project, 179–183
sublayouts, 81–82, 85
  advantages, 414
  caching, 205
  components, 365
  data sources, 416
  defining class prototype, 175–178
  definition items, 365
  deploying, 365, 373
  exception handling, 174–180
  `GetCachingID()`, 208
  placeholders, 356
  XSL files, 83, 170
switching providers, 157–159
syndication, 118–120
syntax errors, in XSL renderings, 169–174
system outages, 196
System Types fields, 45
`System.Collections.Generic.IComparer`, 32
`system:debug` command, 253, 257
`System.Environment.StackTrace`, 242
`System.Exception`, 167
`System.IO`, 319
`System.IO.StreamReader`, 319
`System.Linq`, 33, 337
`System.Net`, 319
`System.Web.Application`, 75
`system:webedit`, 258
`System.Web.HttpApplication`, 75–76
`System.Web.HttpContext`, 309
`System.Web.HttpContext.Current.Response.ContentType`, 28
`System.Web.HttpContext.Current.Response.Redirect()`, 190
`System.Web.HttpContext.Current.Transfer()`, 190
`System.Web.HttpServerUtility.MapPath()`, 341
`System.Web.IHttpAsyncHandler`, 75
`System.Web.IHttpHandler`, 75, 76
`System.Web.UI.Page`, 76
`System.Web.UI.WebControls`, 83
`System.Web.UI.WebControls.Repeater`, 336
`System.Web.WebRequest`, 318
`System.Xml.Linq`, 345
`System.Xml.XPath`, 319
`System.Xml.XPath.XPathNavigator`, 319, 335
`System.Xml.XPath.XPathNodeIterator`, 321

T

tabs, 251
`TaskDatabaseAgent`, 279, 280
tasks, 278
Team Development for Sitecore (TDS), 354, 375
Telerik Rich Text Editor, 14
Template Builder, 12, 40
Template Manager, 11, 12, 43, 134, 410
templates, 40. *See also* data templates
  base, 21, 42
  branch, 30, 31
  command, 30, 246
  rendering parameters, 84–87
  standard, 21, 31, 37, 53–54, 419
test data
  creating from XML, 348–351
  creating through API, 347–348
  creating through HTTP, 344–347
  location of, 343–344

www.it-ebooks.info

setup and destruction, 342–343
working with, 342–351
test environments
content tree, 342
load balanced, 219
test fixtures, 314, 321, 324
test project, 311–317
test runner, 311, 314, 315
TestDriven.NET, 317
testing
automated, 307–308, 317–322
complexities, 309–310
control instantiation, 334–336
embedded test runners, 325–334, 347
invoking API, 336–338
marketing, 383, 387–388
over HTTP, 317–322
Sitecore API without HTTP context, 338–342
test repeatability, 342
web browser driver, 322–325
`TextFixture`, 313
threshold warnings, 229, 231
Trace group, 229
tracing, 229–231, 242
traffic dashboards, 386
traffic type analysis, 383
training, 403, 444
Tree Node Style command, 410
Treelist field, 44, 47
TreelistEx field, 44, 47, 48, 410
Tristate field, 45
troubleshooting, 225–234
`try...catch` blocks, 168–169, 175
Twitter, 157, 249
`type` attribute, 254
types, 247

U

UAT. *See* user acceptance testing
UI. *See* user interfaces
UI conventions, 12
`uiGetMasters` pipeline, 30, 31
unit tests, 308, 309
universal profile management, 383, 390
URL renderings, 82–83, 90
`UrlAgent`, 279
usability, 424–425
user acceptance testing (UAT), 3
user groups, 448
user interfaces (UI)
context site effects, 250
conventions, 424
definitions, 250–251
limiting, 425–427
logging in, 425
overriding, 246, 262–266
pipelines, 289
Sitecore framework, 250–266
XML files, 262–263, 265
User Manager, 11, 259
user profile management switching provider, 158–159
`UserResolver` processor, 192
users, 140–146. *See also* context user
anonymous, 145–146, 418
command access, 253
impersonating, 153
profiles, 143–145
roles, 146–152
virtual, 156–157
"Using Sitecore Query," 108
utility caches, 199

V

Validate command, 56
validation, 246
best practices, 407
data, 55–56
fields, 271–274
global, 406
information architecture, 406
items, 274–277
media library, 421

www.it-ebooks.info

validation *(continued)*
name, 27
process, 270–271
Validation section, 56
Varnish, 223
Vary by Data, 208
Vary by Device, 208
Vary by Login, 208
Vary By options, 207–208, 211, 227–228
Vary by Parameters, 208
Vary by Query String, 208
Vary by User, 208
`VaryBy` attribute, 207
Version Link field, 45
VersionManager Sitecore Shared Source project, 418
`VersionPublishingRestricted`, 260
versions
languages, 22–26
performance, 418
vertical scaling, 216
viewstate cache, 199
virtual directories, 451
virtual users, 156–157
virtual Windows accounts, 476–477
virtualization, 197, 222
cache size limits, 214
high availability, 224
visitor classification, 383
visitor profiling, 383, 388–389
Visual NUnit, 317
Visual Studio. *See* Microsoft Visual Studio

W

`Warn()`, 226
web browser driver, 322–325
web controls, 83–84, 85, 90, 98
advantages, 414–415
caching, 209–210
exception handling, 180–184
404 response, 194–195
XSL error handling, 170–174
Web Forms, 76, 79–80, 250, 262, 265
Web Forms for Marketers (WFFM), 15, 382, 392
web methods, 345–347
web services, 246
Web Services Description Language (WSDL), 345
Web-based Distributed Authoring and Versioning (WebDAV), 45
Web.config, 27, 30, 34, 35, 75, 120, 234–238
audit message settings, 225–226
configuration factory usage, 247–248
configuring installation from archive, 469, 472
Content Editor settings, 431, 433
data cache settings, 202
error management settings, 188
event handler insertion, 285–286
file media settings, 219
404 handler settings, 189–190
`getContentEditorWarnings` pipeline, 260
include files, 235–236, 248
item cache settings, 203
item path cache settings, 203–204
media cache settings, 204–205
multiple site management, 375
pipelines, 287–289
prefetch cache settings, 200–201
publishing instance settings, 221
scalability settings, 217
Sitecore API without HTTP context, 341
`twitter` custom attribute, 249
WebDAV, 14, 45, 66
WebDeploy. *See* Microsoft WebDeploy
WebEdit, 13, 427. *See also* Page Editor
`website` element, 254
WFFM. *See* Web Forms for Marketers
Wget, 317
wildcard items, 39
Windows 7, 189
Windows Authentication, 458
Windows Azure, 11
Windows Installer Package, 311
Windows Scheduler, 278
WinMerge, 234
Word Document field, 14
Workbox, 11, 367–368, 417
workflow engine, 237
workflows, 260, 367–371

www.it-ebooks.info

actions, 370–371
automating, 417–418
best practices, 417–418
commands, 370, 417
components, 367
locking, 368
rules engine triggering, 371
states, 369–370
WSDL. *See* Web Services Description Language

X

XHTML, 3, 318, 324
XHTML validation, 56
XML, 28, 43, 44
calling APIs without HTTP context, 340–341
presentation components, 262–263, 265
test data, 348–351
testing over HTTP, 318–321
web method testing, 345–347
XML-escaped HTML, 43
XmlImporter Sitecore Shared Source project, 71
`xmlns:patch`, 235
XPath, 47, 236, 318, 319, 321, 322, 324
XSL. *See* eXtensible Stylesheet Language
XSL caches, 199
XSL extension methods, 239
XSL rendering control, 335
XSL renderings, 82, 83, 85, 90
ad hoc reports, 439
caches, 199, 232–233
error handling, 169–174
`GetCachingID()`, 208
limitations, 414
traces, 230
XSL transformations, 173, 199, 232–233, 373
XslHelper Sitecore Shared Source project, 82

Y

Yahoo!, 447

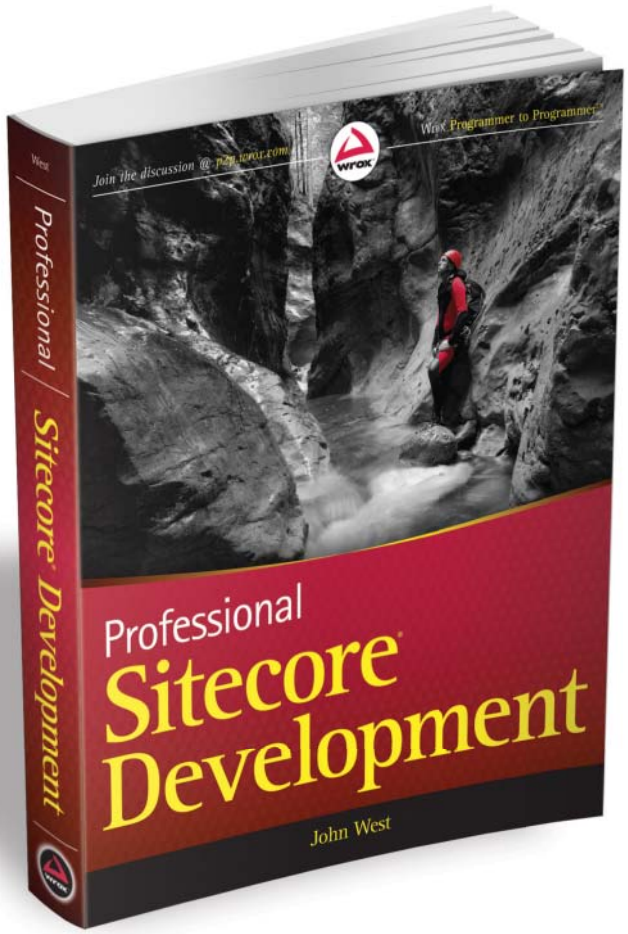
www.it-ebooks.info

www.it-ebooks.info

www.it-ebooks.info

Try Safari Books Online FREE for 15 days + 15% off for up to 12 Months*

Read this book for free online—along with thousands of others—with this 15-day trial offer.



With Safari Books Online, you can experience searchable, unlimited access to thousands of technology, digital media and professional development books and videos from dozens of leading publishers. With one low monthly or yearly subscription price, you get:

- Access to hundreds of expert-led instructional videos on today's hottest topics.
- Sample code to help accelerate a wide variety of software projects
- Robust organizing features including favorites, highlights, tags, notes, mash-ups and more
- Mobile access using any device with a browser
- Rough Cuts pre-published manuscripts

START YOUR FREE TRIAL TODAY!
Visit www.**safaribooksonline.com/wrox38** to get started.

*Available to new subscribers only. Discount applies to the Safari Library and is valid for first 12 consecutive monthly billing cycles. Safari Library is not available in all countries.

Safari Books Online

An Imprint of WILEY
Now you know.

www.it-ebooks.info



Programmer to Programmer™

Connect with Wrox.

Participate

Take an active role online by participating in our P2P forums @ **p2p.wrox.com**

Wrox Blox

Download short informational pieces and code to keep you up to date and out of trouble

Join the Community

Sign up for our free monthly newsletter at **newsletter.wrox.com**

Wrox.com

Browse the vast selection of Wrox titles, e-books, and blogs and find exactly what you need

User Group Program

Become a member and take advantage of all the benefits

Wrox on twitter

Follow @wrox on Twitter and be in the know on the latest news in the world of Wrox

Wrox on facebook

Join the Wrox Facebook page at **facebook.com/wroxpress** and get updates on new books and publications as well as upcoming programmer conferences and user group events

Contact Us.

We love feedback! Have a book idea? Need community support?
Let us know by e-mailing wrox-partnerwithus@wrox.com

www.it-ebooks.info

Related Wrox Books

Beginning ASP.NET 4: in C# and VB

ISBN: 978-0-470-50221-1

This step-by-step book walks you through building rich web sites with ASP.NET 4 and is packed with in-depth examples in both C# and VB. With hands-on exercises that take you right into the action, you'll learn how to build web sites firsthand while you get a good look at what goes on behind the scenes when you view an ASP.NET 4 page in your browser.

Professional ASP.NET 4 in C# and VB

ISBN: 978-0-470-50220-4

With this book, an unparalleled team of authors walks you through the full breadth of ASP.NET and the new and exciting capabilities of ASP.NET 4. The authors also show you how to maximize the abundance of features that ASP.NET offers to make your development process smoother and more efficient.

Professional ASP.NET Design Patterns

ISBN: 978-0-470-29278-5

This unique book takes good ASP.NET application construction one step further by emphasizing loosely coupled and highly cohesive ASP.NET web application architectural design. Each chapter addresses a layer in an enterprise ASP.NET application and shows how proven patterns, principles, and best practices can be leveraged to solve problems and improve the design of your code.

Testing ASP.NET Web Applications

ISBN: 978-0-470-49664-0

There are many different types of testing that can be performed on your ASP.NET web applications and this book shows you how to conduct many of these tests, including functionality of the application, security, load/stress, compliance, and accessibility testing.

www.it-ebooks.info