

Microsoft®

ASP.NET 4

George Shepherd



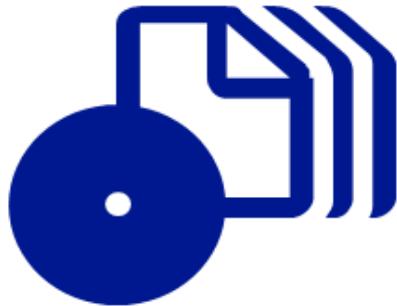
eBook + exercises

Step by Step



• • • • •

How to access your CD files



The print edition of this book includes a CD. To access the CD files, go to <http://aka.ms/627017/files>, and look for the Downloads tab.

Note: Use a desktop web browser, as files may not be accessible from all ereader devices.

Questions? Please contact: mspinput@microsoft.com

Microsoft Press

Microsoft

Microsoft® ASP.NET 4 Step by Step

George Shepherd

PUBLISHED BY

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

Copyright © 2010 by George Shepherd

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

Library of Congress Control Number: 2010925074

Printed and bound in the United States of America.

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

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

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

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

Microsoft, Microsoft Press, Access, ActiveX, DirectX, Expression, Expression Blend, Hotmail, IntelliSense, Internet Explorer, MS, MSDN, MS-DOS, MSN, SharePoint, Silverlight, SQL Server, Visual Basic, Visual C#, Visual Studio, Win32, Windows, Windows Live, Windows NT, Windows Server and Windows Vista are either registered trademarks or trademarks of the Microsoft group of companies. Other product and company names mentioned herein may be the trademarks of their respective owners.

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

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

Acquisitions Editor: Ben Ryan

Developmental Editor: Maria Gargiulo

Project Editor: Melissa von Tschudi-Sutton and Maria Gargiulo

Editorial Production: Waypoint Press, www.waypointpress.com

Technical Reviewer: Kenn Scribner; Technical Review services provided by Content Master, a member of CM Group, Ltd.

Cover: Tom Draper Design

Body Part No. X16-61997

*Dedicated to Sally Bronson Harrison and
Gene Harrison, my second mom and dad.*

Contents at a Glance

Part I Fundamentals

1	Web Application Basics	3
2	ASP.NET Application Fundamentals.....	25
3	The Page Rendering Model.....	59
4	Custom Rendered Controls.....	79
5	Composite Controls	101
6	Control Potpourri	119

Part II Advanced Features

7	A Consistent Look and Feel.....	143
8	Configuration	163
9	Logging In	181
10	Data Binding	207
11	Web Site Navigation.....	237
12	Personalization	257
13	Web Parts.....	267

Part III Caching and State Management

14	Session State	291
15	Application Data Caching	321
16	Caching Output.....	343

Part IV Diagnostics and Plumbing

17	Diagnostics and Debugging	363
18	The <i>HttpApplication</i> Class and HTTP Modules	385
19	HTTP Handlers.....	405

Part V Dynamic Data, XBAP, MVC, AJAX, and Silverlight

20	Dynamic Data	423
21	ASP.NET and WPF Content	433
22	The ASP.NET MVC Framework	449
23	AJAX	473
24	Silverlight and ASP.NET	513

Part VI Services and Deployment

25	Windows Communication Foundation	555
26	Deployment	575

Table of Contents

Acknowledgments	xvii
Introduction	xix

Part I Fundamentals

1 Web Application Basics	3
HTTP Requests	4
HTTP Requests from a Browser.....	5
Making HTTP Requests Without a Browser.....	6
Hypertext Markup Language.....	8
Dynamic Content.....	9
HTML Forms	10
Common Gateway Interface: Very Retro	11
The Microsoft Environment as a Web Server	12
Internet Information Services	12
Internet Services Application Programming Interface DLLs.....	13
Running Internet Information Services	14
Classic ASP: Putting ASP.NET into Perspective	18
Web Development Concepts.....	21
ASP.NET	22
Chapter 1 Quick Reference.....	23
2 ASP.NET Application Fundamentals.....	25
The Canonical Hello World Application.....	26
Mixing HTML with Executable Code.....	31
Server-Side Executable Blocks.....	33
The ASP.NET Compilation Model	41
Coding Options	43
ASP.NET 1.x Style	43
Modern ASP.NET Style	44

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

www.microsoft.com/learning/booksurvey/

The ASP.NET HTTP Pipeline	46
The IIS 5.x and IIS 6.x Pipeline.....	46
The IIS 7.x Integrated Pipeline.....	47
Tapping the Pipeline	48
Visual Studio and ASP.NET	50
Local IIS Web Sites	50
File System–Based Web Sites.....	50
FTP Web Sites.....	51
Remote Web Sites	51
Hello World and Visual Studio.....	52
Chapter 2 Quick Reference.....	58
3 The Page Rendering Model.....	59
Rendering Controls as Tags	59
Packaging the UI as Components.....	62
The Page Using ASP.NET.....	63
The Page's Rendering Model.....	64
The Page's Control Tree	66
Adding Controls Using Visual Studio	67
Layout Considerations.....	77
Chapter 3 Quick Reference.....	78
4 Custom Rendered Controls.....	79
The <i>Control</i> Class	79
Visual Studio and Custom Controls.....	81
A Palindrome Checker.....	88
Controls and Events.....	92
<i>HtmlTextWriter</i> and Controls	95
Controls and <i>ViewState</i>	97
Chapter 4 Quick Reference.....	100
5 Composite Controls	101
Composite Controls versus Rendered Controls.....	101
Custom Composite Controls	102
User Controls.....	110
When to Use Each Type of Control	117
Chapter 5 Quick Reference.....	117

6 Control Potpourri	119
Validation	119
How Page Validation Works	125
Other Validators	127
Validator Properties	128
Image-Based Controls	128
TreeView	132
MultiView	136
Chapter 6 Quick Reference	139
 Part II Advanced Features	
7 A Consistent Look and Feel	143
Managing User Interface Consistency	143
ASP.NET Master Pages	145
Themes	155
Skins	159
Chapter 7 Quick Reference	161
8 Configuration	163
Windows Configuration	164
.NET Configuration	164
Machine.Config	165
Configuration Section Handlers	165
Web.Config	167
Managing Configuration in ASP.NET 1.x	168
Managing Configuration in Later Versions of ASP.NET	169
Configuring ASP.NET from IIS	174
Chapter 8 Quick Reference	180
9 Logging In	181
Web-Based Security	182
Securing IIS	183
Basic Forms Authentication	184
ASP.NET Authentication Services	189
The <i>FormsAuthentication</i> Class	190
An Optional Login Page	191
Managing Users	194

ASP.NET Login Controls.....	200
Authorizing Users.....	203
Chapter 9 Quick Reference.....	206
10 Data Binding	207
Representing Collections Without Data Binding	207
Representing Collections with Data Binding.....	208
<i>ListControl</i> -Based Controls.....	209
<i>TreeView</i> Control	209
<i>Menu</i> Control	209
<i>FormView</i> Control	209
<i>GridView</i> Control	209
<i>DetailsView</i> Control.....	210
<i>DataList</i> Control.....	210
<i>Repeater</i> Control	210
Simple Data Binding	210
Accessing Databases	215
The .NET Database Story.....	215
Connections	215
Commands	217
Managing Results.....	218
ASP.NET Data Sources	221
Other Data-Bound Controls.....	226
LINQ	234
Chapter 10 Quick Reference.....	236
11 Web Site Navigation.....	237
ASP.NET Navigation Support.....	237
Navigation Controls.....	237
XML Site Maps	239
The <i>SiteMapProvider</i>	239
The <i>SiteMap</i> Class	239
The <i>SiteMapNode</i>	240
Using Navigation Controls	241
The <i>Menu</i> and <i>TreeView</i> Controls	241
The <i>SiteMapPath</i> Control	241
Site Map Configuration.....	242
Building Navigable Web Sites	243

Trapping the <i>SiteMapResolve</i> Event	247
Defining Custom Attributes for Each Node.....	248
Security Trimming	251
URL Mapping	251
URL Rewriting	255
Chapter 11 Quick Reference.....	256
12 Personalization	257
Personalizing Web Visits.....	257
Personalization in ASP.NET.....	258
User Profiles	258
Personalization Providers.....	258
Using Personalization	259
Defining Profiles in Web.Config	259
Using Profile Information.....	259
Saving Profile Changes	260
Profiles and Users.....	261
Chapter 12 Quick Reference.....	266
13 Web Parts.....	267
A Brief History of Web Parts.....	268
What Good Are Web Parts?.....	268
Developing Web Parts Controls.....	269
Web Parts Page Development.....	269
Web Parts Application Development.....	269
The Web Parts Architecture	269
<i>WebPartManager</i> and <i>WebZones</i>	270
Built-In Zones	270
Built-In Web Parts	271
Developing a Web Part.....	280
Chapter 13 Quick Reference.....	288
Part III Caching and State Management	
14 Session State	291
Why Session State?.....	292
ASP.NET and Session State	292
Introduction to Session State	293
Session State and More Complex Data	299

Configuring Session State	306
Turning Off Session State	307
Storing Session State <i>InProc</i>	307
Storing Session State in a State Server	307
Storing Session State in a Database	308
Tracking Session State	309
Tracking Session State with Cookies	309
Tracking Session State with the URL	310
Using <i>AutoDetect</i>	310
Applying Device Profiles	311
Session State Timeouts	311
Other Session Configuration Settings	311
The <i>Wizard</i> Control: An Alternative to Session State	312
Chapter 14 Quick Reference	320
15 Application Data Caching	321
Getting Started with Caching	321
Using the Data Cache	324
Impact of Caching	325
Managing the Cache	327
<i>DataSets</i> in Memory	328
Cache Expirations	331
Cache Dependencies	334
The SQL Server Dependency	336
Clearing the Cache	338
Chapter 15 Quick Reference	341
16 Caching Output	343
Caching Page Content	343
Managing Cached Content	346
Modifying the <i>OutputCache</i> Directive	346
The <i>HttpCachePolicy</i>	351
Caching Locations	352
Output Cache Dependencies	353
Caching Profiles	353
Caching User Controls	354
When Output Caching Makes Sense	357
Other Cache Providers	358
Chapter 16 Quick Reference	359

Part IV Diagnostics and Plumbing

17	Diagnostics and Debugging	363
	Page Tracing	363
	Tracing	364
	Trace Statements	367
	Application Tracing	370
	Enabling Tracing Programmatically	373
	The <i>TraceFinished</i> Event	373
	Piping Other Trace Messages.	374
	Debugging with Visual Studio	374
	Error Pages	378
	Unhandled Exceptions.	381
	Chapter 17 Quick Reference.	383
18	The <i>HttpApplication</i> Class and HTTP Modules	385
	The Application: A Rendezvous Point.	385
	Overriding <i>HttpApplication</i>	387
	HttpModules.	394
	Global.asax vs. <i>HttpModules</i>	404
	Chapter 18 Quick Reference.	404
19	HTTP Handlers.	405
	ASP.NET Request Handlers	405
	The Built-in Handlers.	407
	Handlers and <i>IHttpHandler</i>	410
	Handlers and Session State.	416
	Generic Handlers (ASHX Files)	417
	Chapter 19 Quick Reference.	419
Part V	Dynamic Data, XBAP, MVC, AJAX, and Silverlight	
20	Dynamic Data	423
	Dynamic Data Controls	424
	Dynamic Data Details	428
	Chapter 20 Quick Reference.	432

21 ASP.NET and WPF Content	433
Improving Perceived Performance by Reducing Round-Trips	433
What Is WPF?	434
How Does WPF Relate to the Web?	436
Loose XAML Files	437
XBAP Applications	438
WPF Content and Web Applications	442
What About Silverlight?	448
Chapter 21 Quick Reference	448
22 The ASP.NET MVC Framework	449
The Model-View-Controller (MVC) Architecture	449
ASP.NET and MVC	452
ASP.NET MVC vs. Web Forms	453
MVC and Testing	454
How MVC Plays with ASP.NET	455
Following the Request Path	455
Chapter 22 Quick Reference	472
23 AJAX	473
Rich Internet Applications	473
What Is AJAX?	474
ASP.NET and AJAX	475
Reasons to Use AJAX	476
Real-World AJAX	477
AJAX in Perspective	478
ASP.NET Server-Side Support for AJAX	478
<i>ScriptManager</i> Control	479
<i>ScriptManagerProxy</i> Control	479
<i>UpdatePanel</i> Control	479
<i>UpdateProgress</i> Control	480
<i>Timer</i> Control	480
AJAX Client Support	480
ASP.NET AJAX Control Toolkit	480
AJAX Control Toolkit Potpourri	481
Getting Familiar with AJAX	484
The Timer	490
Updating Progress	497

Extender Controls	501
The <i>AutoComplete</i> Extender	501
A Modal Pop-up Dialog-Style Component.....	508
Chapter 23 Quick Reference.....	512
24 Silverlight and ASP.NET	513
Web Applications Mature.....	514
What Is Silverlight?.....	515
Creating a Silverlight Application	517
Architecture.....	521
XAML	522
Constructing the Visual Tree	522
XAML and Namespaces	523
Compiling the Silverlight Application.....	524
Adding Silverlight Content to a Web Page	524
Using the Object Tag.....	524
Using the ASP.NET Silverlight Server-Side Control.....	525
Using the JavaScript Function	526
Controls and Events	526
Routed Events.....	526
Silverlight Controls and Class Members.....	527
Silverlight and Layout	528
Integrating with HTML	533
Animations	535
WCF Services and Silverlight	542
Chapter 24 Quick Reference.....	551
Part VI Services and Deployment	
25 Windows Communication Foundation	555
Distributed Computing Redux.....	555
A Fragmented Communications API.....	556
WCF for Connected Systems	556
WCF Constituent Elements.....	557
Endpoints	557
Channels	558
Behaviors.....	558
Messages.....	559

How WCF Plays with ASP.NET	560
Side-by-Side Mode	560
ASP.NET Compatibility Mode.....	561
Writing a WCF Service.....	561
Building a WCF Client	567
Chapter 25 Quick Reference.....	573
26 Deployment.....	575
Visual Studio Web Sites.....	576
HTTP Web Sites	576
FTP Web Sites.....	576
File System Web Sites	577
Precompiling.....	577
Precompiling for Performance	577
Precompiling for Deployment.....	578
Visual Studio 2010 Deployment Support	578
Chapter 26 Quick Reference.....	585
Index.....	587



What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

www.microsoft.com/learning/booksurvey/

Acknowledgments

The last time I wrote the acknowledgments for this book, I mentioned how my son, Ted, had written a Father's Day card for me in HTML. Ted is in college now, and I can remember his searching out and applying for schools during the last couple of years of high school. He did it almost entirely online, over the Web. How different that was from my experience applying to schools!

The Web permeates our social infrastructure. Whether you're a businessperson wanting to increase the visibility of your business, an avid reader trying to find an out-of-print book, a student fetching homework assignments from a school Web site, or any other producer or consumer of information, you touch the Internet.

Publishing a book is a huge effort. My name is on the lower right corner of the cover as the author, but I did only some of the work. I have so many people to thank for helping get this book out.

Thank you, Claudette Moore, for hooking me up with Microsoft Press again. Claudette has acted as my agent for all my work with Microsoft Press, handling the business issues so I can be free to write. Thank you, Maria Gargiulo, for managing the project. It's been great working with you. Thank you, Charlotte Twiss, for getting the code samples onto the CD. Thank you, Steve Sagman, for composing the pages so beautifully. Thank you, Christina Yeager, for copyediting the pages and making it appear that I can actually write coherent sentences, as well as for indexing the project. You all did a wonderful job on the editing, production, and layout. Thank you, Kenn Scribner, for providing the best technical objective eye I've ever worked with. Thank you, Ben Ryan, for accepting the book proposal and hiring me to create the book.

Thank you, Jeff Duntemann, for buying and publishing my first piece ever for *PC Tech Journal*. Thank you, JD Hildebrand, for buying my second writing piece ever, and for the opportunity to work with you all at Oakley Publishing. Thank you, Sandy Daston, for your support and guidance early in my writing career. Thank you to the folks at DevelopMentor for being an excellent group of technical colleagues and a great place for learning new technology. Thanks to my buds at Schwab Performance Technologies.

Thanks to my evil Java twin, Pat Shepherd, and his family, Michelle, Belfie, and Bronson. Thank you, Ted Shepherd, you're the best son ever. Thank you, George Robbins Shepherd

and Betsy Shepherd. As my parents, you guided me and encouraged me to always do my best. I miss you both dearly.

Finally, thank you, reader, for going through this book and spending time learning ASP.NET. May you continue to explore ASP.NET and always find new and interesting ways to handle HTTP requests.

—George Shepherd

Chapel Hill, NC

March, 2010

Introduction

This book shows you how to write Web applications using Microsoft ASP.NET 4, the most current version of the Microsoft HTTP request processing framework. Web development has come a long way since the earliest sites began popping up on the Internet in the early 1990s. The world of Web development offers several choices of development tools. During the past few years, ASP.NET has evolved to become one of the most consistent, stable, and feature-rich frameworks available for managing HTTP requests.

ASP.NET, together with Microsoft Visual Studio, includes a number of features to make your life as a Web developer easier. For example, Visual Studio offers several project templates that you can use to develop your site. Visual Studio also supports a number of development modes, including using Microsoft Internet Information Services (IIS) directly to test your site during development, using a built-in Web server, and developing your site over an FTP connection. With the debugger in Visual Studio, you can run the site and step through the critical areas of your code to find problems. With the Visual Studio Designer, you can develop effective user interfaces by dropping control elements onto a canvas to see how they appear visually. And when you are ready to deploy your application, Visual Studio makes it easy to create a deployment package. These are but a few of the features built into the ASP.NET framework when paired with Visual Studio.

The purpose of this book is to tell the story of ASP.NET development. Each section presents a specific ASP.NET feature in a digestible format with examples. The stepwise instructions yield immediate working results. Most of the main features of ASP.NET are illustrated here using succinct, easily duplicated examples. The examples are rich to illustrate features without being overbearing. In addition to showing off ASP.NET features by example, this book contains practical applications of each feature so that you can apply these techniques in the real world. After reading this book and applying the exercises you'll have a great head start into building real Web sites that include such modern features as AJAX, WCF services, custom controls, and master pages.

This book is organized so that you can read each chapter independently for the most part. With the exception of Chapter 1, "Web Application Basics," and the three chapters on server-side controls (Chapters 3 to 5), which make sense to tackle together, each chapter serves as a self-contained block of information about a particular ASP.NET feature. In addition, for the sake of completeness, Chapter 1 also includes information about how IIS and ASP.NET interact together.

Who This Book Is For

This book is targeted at several types of developers:

- **Those starting out completely new to ASP.NET** The text includes enough back story to explain the Web development saga even if you've developed only desktop applications.
- **Those migrating from either ASP.NET 1.x, 2.0, 3.x, or even classic ASP** The text explains how ASP.NET 4 is different from earlier versions of ASP.NET. It also includes references explaining differences between ASP.NET and classic ASP.
- **Those who want to consume ASP.NET how-to knowledge in digestible pieces** You don't have to read the chapters in any particular order to find the book valuable. Each chapter stands more or less on its own (with the exception of the first chapter, which details the fundamentals of Web applications—you might want to read it first if you've never ventured beyond desktop application development). You might find it useful to study the chapters about server-side controls (Chapters 3 to 5) together, but it's not completely necessary to do so.

Getting Started

If you've gotten this far, you're probably ready to begin writing some code.



Important Before beginning, make sure that:

- Visual Studio 2010 is installed on your computer.
As long as you've installed the development environment, you can be sure the .NET run-time support is installed as well.
- You have Administrator permissions on your computer.
See "Installing the C# Code Samples" later in this Introduction for more information.
- IIS is installed and running on your computer.
IIS is required to run the code samples for Chapters 1, 2, 9, and 26. To install IIS in Windows 7, click Start, and click Control Panel. In Control Panel, click Programs and Features, and click Turn Windows Features On or Off. In the Windows Features dialog box, expand Internet Information Services, select the checkboxes next to Web Management Tools and World Wide Web Services, and click OK.

If you attempt to install the code without IIS running, you might see an error message like the following. To bypass this error message, click Ignore to continue installation.



The first few code examples require nothing but a text editor and a working installation of IIS. To start, you can begin with some basic examples to illustrate the object-oriented nature and compilation model of ASP.NET. In addition to seeing exactly how ASP.NET works when handling a request, this is a good time to view the architecture of ASP.NET from a high level. Next, you progress to Web form programming and begin using Visual Studio to write code—which makes things much easier!

After learning the fundamentals of Web form development, you can see the rest of ASP.NET through examples of ASP.NET features such as server-side controls, content caching, custom handlers, output and data caching, and debugging and diagnostics, all the way to ASP.NET support for Web Services.

Finding Your Best Starting Point in This Book

This book is designed to help you build skills in a number of essential areas. You can use this book whether you are new to Web programming or you are switching from another Web development platform. Use the following table to find your best starting point in this book.

If you are	Follow these steps
New to Web development	<ol style="list-style-type: none">1. Install the code samples.2. Work through the examples in Chapters 1 and 2 sequentially. They ground you in the ways of Web development. They also familiarize you with ASP.NET and Visual Studio.3. Complete the rest of the book as your requirements dictate.
New to ASP.NET and Visual Studio	<ol style="list-style-type: none">1. Install the code samples.2. Work through the examples in Chapter 2. They provide a foundation for working with ASP.NET and Visual Studio.3. Complete the rest of the book as your requirements dictate.

If you are	Follow these steps
Migrating from earlier versions of ASP.NET	<ol style="list-style-type: none">1. Install the code samples.2. Skim the first two chapters to get an overview of Web development in the Microsoft environment and with Visual Studio 2010.3. Concentrate on Chapters 3 through 26 as necessary. You might already be familiar with some topics and might need only to see how a particular current feature differs from earlier versions of ASP.NET. In other cases, you might need to explore a feature that is completely new in ASP.NET 4.
Referencing the book after working through the exercises	<ol style="list-style-type: none">1. Use the index or the table of contents to find information about particular subjects.2. Read the Quick Reference section at the end of each chapter to find a brief review of the syntax and techniques presented in the chapter.

Conventions and Features in This Book

This book uses conventions designed to make the information readable and easy to follow. Before you start the book, read the following list, which explains conventions you'll see throughout the book and points out helpful features in the book that you might want to use.

Conventions

- Each chapter includes a summary of objectives near the beginning.
- Each exercise is a series of tasks. Each task is presented as a series of steps to be followed sequentially.
- “Tips” provide additional information or alternative methods for completing a step successfully.
- “Important” reader aids alert you to critical information for installing and using the sample code on the companion CD.
- Text that you type appears in bold type, like so:

```
class foo
{
    System.Console.WriteLine("HelloWorld");
}
```

- The directions often include alternative ways of accomplishing a single result. For example, you can add a new item to a Visual Studio project from either the main menu or by right-clicking in Solution Explorer.
- The examples in this book are written using C#.

Other Features

- Some text includes sidebars and notes to provide more in-depth information about the particular topic. The sidebars might contain background information, design tips, or features related to the information being discussed. They might also inform you about how a particular feature differs in this version of ASP.NET from earlier versions.
- Each chapter ends with a Quick Reference section that contains concise reminders of how to perform the tasks you learned in the chapter.

Prerelease Software

This book was reviewed and tested against the Visual Studio 2010 release candidate one week before the publication of this book. We reviewed and tested the examples against the Visual Studio 2010 release candidate. You might find minor differences between the production release and the examples, text, and screenshots in this book. However, we expect them to be minimal.

Hardware and Software Requirements

You need the following hardware and software to complete the practice exercises in this book:



Important The Visual Studio 2010 software is *not* included with this book! The CD-ROM packaged in the back of this book contains the code samples needed to complete the exercises. The Visual Studio 2010 software must be purchased separately.

- Windows 7; Windows Server 2003; Windows Server 2008; or Windows Vista
- Internet Information Services (included with Windows). You will need IIS 5.1 or later. IIS 7.5 is the latest release at the time of this writing.
- Microsoft Visual Studio 2010 Ultimate, Visual Studio 2010 Premium, or Visual Studio 2010 Professional
- Microsoft SQL Server 2008 Express (included with Visual Studio 2010) or SQL Server 2008 (SQL Server 2008 R2 is the latest release at the time of this writing)
- 1.6-GHz Pentium or compatible processor
- 1 GB RAM for x86
- 2 GB RAM for x64
- An additional 512 MB RAM if running in a virtual machine

- DirectX 9–capable video card that runs at 1024 × 768 or higher display resolution
- 5400-RPM hard drive (with 3 GB of available hard disk space)
- DVD-ROM drive
- Microsoft mouse or compatible pointing device
- 5 MB of available hard disk space to install the code samples

You also need to have Administrator access to your computer to configure Microsoft SQL Server 2008 Express.

Code Samples

The companion CD inside this book contains the code samples, written in C#, that you use as you perform the exercises in the book. By using the code samples, you won't waste time creating files that aren't relevant to the exercise. The files and the step-by-step instructions in the lessons also help you learn by doing, which is an easy and effective way to acquire and remember new skills.

Digital Content for Digital Book Readers

If you bought a digital-only edition of this book, you can enjoy select content from the print edition's companion CD. Visit <http://www.microsoftpressstore.com/title/9780735627017> and look for the Examples link to get your downloadable content.

Installing the C# Code Samples

Follow the steps here to install the C# code samples on your computer so that you can use them with the exercises in this book.



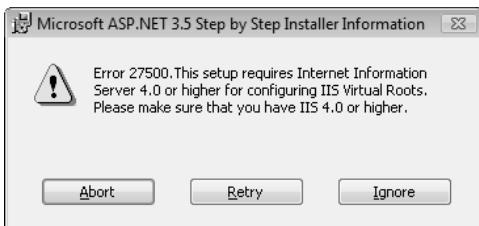
Important Before you begin, make sure that you have

- Administrator permissions on your computer.
- IIS installed and running on your computer.

Chapters 1, 2, 9, and 26 include information about using IIS, and their companion code samples require IIS. The code sample installer modifies IIS. Working with IIS requires that you have administration privileges on your machine. If you are using your own computer at home, you probably have Administrator rights. If you are using a computer in an organization and you do not have Administrator rights, please consult your computer support or IT staff.

To install IIS in Windows 7, click Start, and click Control Panel. In Control Panel, click Programs and Features, and click Turn Windows Features On or Off. In the Windows Features dialog box, expand Internet Information Services, select the checkboxes next to Web Management Tools and World Wide Web Services, and click OK.

If you attempt to install the code without IIS running, you might see an error message like the following. To bypass this error message, click Ignore to continue installation.



1. Remove the companion CD from the package inside this book and insert it into your CD-ROM drive.



Note A menu screen for the CD should open automatically. If it does not appear, open Computer on the desktop or the Start menu, double-click the icon for your CD-ROM drive, and then double-click StartCD.exe.

2. In the companion CD UI, select Code from the menu on the left. The InstallShield Wizard will guide you through the installation process.
3. Review the end-user license agreement. If you accept the terms, select the accept option, and then click Next.
4. Accept the default settings to install the code.

The code samples are installed to the following location on your computer:

\C\Microsoft Press\ASP.NET 4 Step by Step

Additionally, if you have IIS running and you open the Internet Information Services console, you will see that the installer creates a virtual directory named *aspnet4sbs* under the Default Web Site. Below the *aspnet4sbs* virtual directory, various Web applications are created.

Using the Code Samples

Each chapter in this book explains when and how to use any code samples for that chapter. When it's time to use a code sample, the book lists the instructions for how to open the files. Many chapters begin projects completely from scratch so that you can understand the entire development process. Some examples borrow bits of code from previous examples.

Here's a comprehensive list of the code sample projects:

Project	Description
Chapter 1	
HelloWorld.asp, Selectnoform.asp, Selectfeature.htm, Selectfeature2.htm, Selectfeature.asp	Several Web resources illustrating different examples of raw HTTP requests
WebRequestor	A simple application that issues a raw HTTP request
Chapter 2	
HelloWorld, HelloWorld2, HelloWorld3, HelloWorld4, HelloWorld5, partial1.cs, partial2.cs	Web resources illustrating compilation models and partial classes in ASP.NET
Chapter 3	
BunchOfControls.htm, BunchOfControls.asp, BunchOfControls.aspx	Web resources illustrating rendering control tags
ControlsORama	Visual Studio-based project illustrating Visual Studio and server-side controls
Chapter 4	
ControlsORama	Extends the example begun in Chapter 3. Illustrates creating and using rendered server-side controls
Chapter 5	
ControlsORama	Extends the example used in Chapter 4. Illustrates creating and using composite server-side controls and user controls
Chapter 6	
ControlPotpourri	Illustrates control validation, the <i>TreeView</i> , the <i>Image</i> , the <i>ImageButton</i> , the <i>ImageMap</i> , and the <i>MultiView/View</i> controls
Chapter 7	
MasterPageSite	Illustrates developing a common look and feel throughout multiple pages in a single Web application using master pages, themes, and skins

Project	Description
Chapter 8 ConfigORama	Illustrates configuration in ASP.NET. Shows how to manage the web.config file, how to add new configuration elements, and how to retrieve those configuration elements.
Chapter 9 SecureSite Login.aspx, OptionalLogin.aspx, Web.Config, Web.ConfigForceAuthentication, Web.ConfigForOptionalLogin	Illustrates Forms Authentication and authorization in a Web site Web resources for illustrating Forms Authentication at the very barest level
Chapter 10 DataBindORama	Illustrates data binding to several different controls, including the <i>GridView</i> . Illustrates the <i>DataSource</i> controls. Also illustrates loading and saving data sets as XML and XML schema
Chapter 11 NavigateMeSite	Illustrates ASP.NET navigation features
Chapter 12 MakeItPersonal	Illustrates ASP.NET personalization features
Chapter 13 UseWebParts	Illustrates using Web Parts in a Web application
Chapter 14 SessionState	Illustrates using session state in a Web application
Chapter 15 UseDataCaching	Illustrates caching data to improve performance
Chapter 16 OutputCache	Illustrates caching output to improve performance
Chapter 17 DebugORama	Illustrates debugging and tracing Web applications
Chapter 18 UseApplication	Illustrates using the global application object and HTTP modules as a rendezvous point for the application. Illustrates storing globally scoped data and handling application-wide events

Project	Description
Chapter 19	
CustomHandlers	Illustrates custom HTTP handlers, both as separate assemblies and as ASHX files
Chapter 20	
DynamicDataLinqToSQLSite	Illustrates how ASP.NET Dynamic works to create data-driven sites
Chapter 21	
XAMLOrama	Illustrates how to use loose XAML in a site
XBAPORama	Illustrates how to create an XAML-based Browser Application (XBAP)
Chapter 22	
MVCORama	Illustrates how to create and manage an MVC-based site, complete with a database
Chapter 23	
AJAXORama	Illustrates using AJAX to improve the end user experience
Chapter 24	
SilverlightSite	Illustrates how to include Silverlight content in an ASP.NET site
SilverlightLayout	Shows how Silverlight layout panels work
SilverlightAnimations	Illustrates using animations in Silverlight
SilverlightAndWCF	Shows how a Silverlight component can communicate to a Web site via WCF
Chapter 25	
WCFQuotesService	Illustrates how to create and consume an ASP.NET WCF service
Chapter 26	
DeployThisApplication	Illustrates the new ASP.NET Packaging system, which facilitates deployment

All these projects are available as complete solutions for the practice exercises (in case you need any inspiration).

Uninstalling the Code Samples

Follow these steps to remove the code samples from your computer:

1. In Control Panel, open Add Or Remove Programs.
2. From the list of Currently Installed Programs, select Microsoft ASP.NET 4 Step by Step.
3. Click Remove.
4. Follow the instructions that appear to remove the code samples.

Support for This Book

Every effort has been made to ensure the accuracy of this book and the contents of the companion CD. As corrections or changes are collected, they will be added to a Microsoft Knowledge Base article. Microsoft Press provides support for books and companion CDs at the following Web site:

<http://www.microsoft.com/learning/support/books/>

If you have comments, questions, or ideas regarding the book or the companion CD, or questions that are not answered by visiting the sites previously mentioned, please send them to Microsoft Press by sending an e-mail message to mspininput@microsoft.com.

Please note that Microsoft software product support is not offered through the preceding address.

We Want to Hear from You

We welcome your feedback about this book. Please share your comments and ideas through the following short survey:

<http://www.microsoft.com/learning/booksurvey>

Your participation helps Microsoft Press create books that better meet your needs and your standards.



Note We hope that you will give us detailed feedback in our survey. If you have questions about our publishing program, upcoming titles, or Microsoft Press in general, we encourage you to interact with us using Twitter at <http://twitter.com/MicrosoftPress>. For support issues, use only the e-mail address shown earlier.

Chapter 21

ASP.NET and WPF Content

After completing this chapter, you will be able to

- Understand the benefits of Windows Presentation Foundation (WPF) over traditional Windows user interfaces.
- Create an XAML-based browser application (XBAP) site.
- Add WPF-based content to an ASP.NET site.

The last 20 chapters demonstrate how ASP.NET makes Web development approachable by pushing most HTML rendering to the ASP.NET *Control* class and its descendants. In addition, the ASP.NET pipeline hides many of the details of a Web request so that you can focus on your part in development. The next few chapters show alternative paths for producing content for the end user, including information on ASP.NET support for AJAX, its implementation of the Model-View-Controller pattern, and how Microsoft Silverlight works. This chapter starts by discussing how you can render Extensible Application Markup Language (XAML)-based content to the browser.

Improving Perceived Performance by Reducing Round-Trips

Throughout the history of the Web, one main way developers have improved end-user experience has been to reduce the number of round-trips to the server. For a long time, the only way to do this was to employ client-side scripting in a Web page. That way, certain parts of the application were executed on the client's browser, which is usually much faster than making an entire round-trip.

Chapter 23, "AJAX," discusses AJAX, which represents a major improvement in Web-based user interfaces (UIs). AJAX adds many elements to Web-based user interfaces that have been available previously only to desktop applications. For example, the AJAX *AutoComplete* extender allows users typing text into a *TextBox* to select from options generated dynamically from a Web service. With the *ModalPopupExtender*, you can provide content in a pane that behaves like a standard Windows modal dialog box at run time.

However, scripting isn't the only way to push functionality to the browser. AJAX still relies fundamentally on HTML, and although HTML includes a huge set of tags that render to standard user interface elements that run in the browser, it stops there. Being able to run WPF content on a site changes that. WPF represents a new way to add rich user interfaces to a site, and it turns standard Web-based (and Windows-based) user interface programming

on its head. In this chapter, you see how WPF works and how it relates to the Internet and to browser applications. You revisit some of this when you look at Silverlight, a similar technology. For now, first look at WPF.

What Is WPF?

Windows-based user interface programming is based on an architecture that has remained fundamentally unchanged for more than a quarter century. Since back in the early 1980s through today, all applications have had the same basic underpinnings: The main application runs a message loop, picks up Windows messages off of the message queue, and deposits them into a window handler. Every window is responsible for rendering its own presentation—that is, every window, from the top-level window of the application to the most minor control in the window.

Nearly all Windows-based applications today use the Win32 application programming interface (API) at the lowest level. The classic Win32 API has worked well for a long time. However, its design is beginning to show its age. Because every window and control is responsible for its own rendering using the Win32 Graphics Device Interface (GDI, or the GDI+ interface, in the case of Windows Forms), fundamental user interface limitations are built into the design of the Windows operating system. The GDI and GDI+ interfaces have a huge array of functions. However, it takes a lot of work to do much more than basic drawing and text rendering. That is, special effects such as transformations, transparency, and video play integration are difficult to accomplish using the current Windows graphics interface. Windows does support a richer graphics-based interface named Direct X; however, using it is often beyond the scope of most Windows-based applications and is typically reserved for use by game programmers.

The limitations of the classic Windows API prompted Microsoft to develop a new programming interface: the Windows Presentation Foundation (WPF). With WPF, programming special effects for Windows-based applications (including presenting Web content, as described later) is very approachable. The WPF libraries are made up of a number of classes that work together very much like the Windows Forms classes do (on the surface at least; underneath the goings-on are very different from Windows Forms).

WPF represents a very rich programming interface for developing a user interface. Here's a short list of the kinds of features available through WPF (this is a broad summary and is not exhaustive):

- User interface elements that you can modify in all kinds of ways much more easily than you can using Win32 and subclassing
- Paths, shapes, and geometries for drawing two-dimensional presentations

- Transforms (scaling, translating, rotation, and skewing) that allow consistent and uniform modifications to all user interface elements
- Ability to manage the opacity of individual elements
- Built-in layout panels
- Brushes—image, video, and drawing brushes for filling areas on the screen
- Animations

WPF applications arrange the UI elements using layout panels. Rather than relying on absolute positioning (as is the case for Win32 applications) or flow layout (as is the case for ASP.NET pages), WPF introduces a number of layout options including the following:

- **Grid** Elements are placed in a table.
- **StackPanel** Elements are stacked vertically or horizontally.
- **Canvas** Elements are positioned absolutely.
- **DockPanel** Elements are positioned against the sides of the host.
- **WrapPanel** Elements are repositioned to fit when the host is resized.

The example that follows later uses the *Canvas*.

You craft a typical WPF application from files in very much the same way that you create an ASP.NET application. A stand-alone WPF application includes a main application object that runs the message loop and one or more windows, which are browser-based WPF applications made up of pages. WPF application components are typically composed from a markup file, just like ASP.NET pages are. WPF layouts are defined using Extensible Application Markup Language (XAML).

XAML files describe a WPF layout's logical tree, the collection of WPF user interface elements. A WPF application is made up of Common Language Runtime (CLR) classes underneath the façade of markup language, very much like the ASP.NET object model is. XAML files represent instructions for constructing a logical tree of visual elements. In the case of a stand-alone Windows application, the logical tree exists in a top-level window. In the case of a browser-based application, the logical tree exists in a browser pane. The following is a short XAML listing that displays "Hello World" in a button hosted in a browser pane:

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:sys="clr-namespace:System;assembly=mscorlib"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
    <Button Height="100" Width="100">Hello World</Button>
</Page>
```

The preceding code doesn't do a whole lot, but it is an example of the fundamental structure of a WPF page as expressed in XAML. When run, the XAML you see listed starts a browser session and displays a button with the string "Hello World" as its content (provided the XAML plug-in is installed). In a real application, instead of containing a single button with a string, the top-level WPF node can contain elaborate layouts using the different layout panels available in WPF. You see an example of this soon.

How Does WPF Relate to the Web?

What does all this mean for Web applications? Windows Internet Explorer and other browsers running under the Windows operating system are based on the classic Windows architecture. Browsers are responsible for rendering HTML using the graphic interface available to Windows: the Graphics Device Interface (GDI). Consequently, accomplishing special effects in browsers (and typical HTML) is just as difficult as it is with traditional Windows programs.

Web programming is based on submitting HTTP requests to a server, processing the requests, and sending back responses to the client. In that sense, any user interface–specific responses are constrained to whatever can be expressed in HTML. The Web is dynamic, and HTML is basically a document technology.

Is there another markup language that provides more than just simple tags that can be interpreted by an HTML browser? Yes, that's what XAML is when used in the context of a Web application.

Remember the previous code example? If the contents of the file are saved in an ASCII text file named `HelloWorld.xaml`, and you double click it in Windows Explorer, Internet Explorer loads and parses the XAML content. Figure 21-1 shows how it appears in Internet Explorer when you load the XAML file into the browser. Simply double-click the file name in Windows Explorer to see the application.

When adding WPF-style content directly to a Web site, you have three options: presenting the content through loose XAML files, creating an XAML-based browser application (XBAP), or using Silverlight. (Silverlight is described in more detail in Chapter 24, "Silverlight and ASP.NET.")

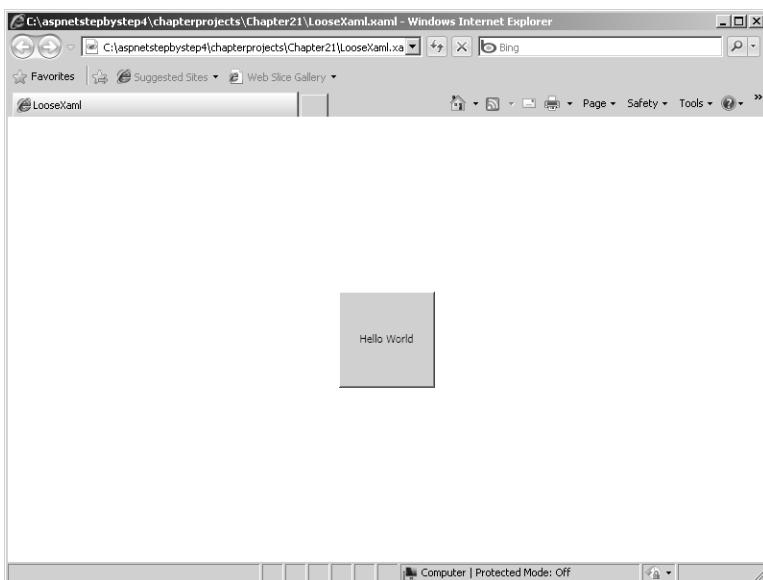


FIGURE 21-1 A button rendered as specified by XAML.

Loose XAML Files

As just shown, if you place a properly formatted XAML file in your site and make it available through a Web server, any browser capable of using the XAML plug-in (such as Internet Explorer) can pick it up and render it. This is one option for presenting WPF-based content from a Web site. This technique is useful for rendering semidynamic content—that is, for rendering anything expressible using pure XAML files.

The WPF programming model marries XAML layout instructions with accompanying code modules in very much the same way that ASP.NET does. Events generated from user interface elements are handled in the accompanying code. Deploying s as loose XAML files precludes adding event handlers and accompanying code.

However, WPF elements are dynamic in the sense that they can be animated, and user interface elements can be tied together using only XAML. That's why WPF content expressed only through XAML is semidynamic. You can hook up some interactive elements using only XAML, but there's a limit. For example, all through XAML you can render a list of names of images in a list box and allow users to select an image to zoom. You can attach slider controls to user interface elements so that the end user can change various aspects of the elements through the slider. However, you cannot implement event handlers for controls; that requires deploying a WPF application as an XBAP application.

XBAP Applications

XBAPs are another way to deploy WPF content over the Web. They're a bit more complex than loose XAML files are. In addition to expressing layout, XBAPs support accompanying executable code for each page. When you deploy a WPF application over the Web, the client receives the WPF visual layout and the accompanying code is downloaded to the client computer. Events occurring in the XBAP are handled on the client side.

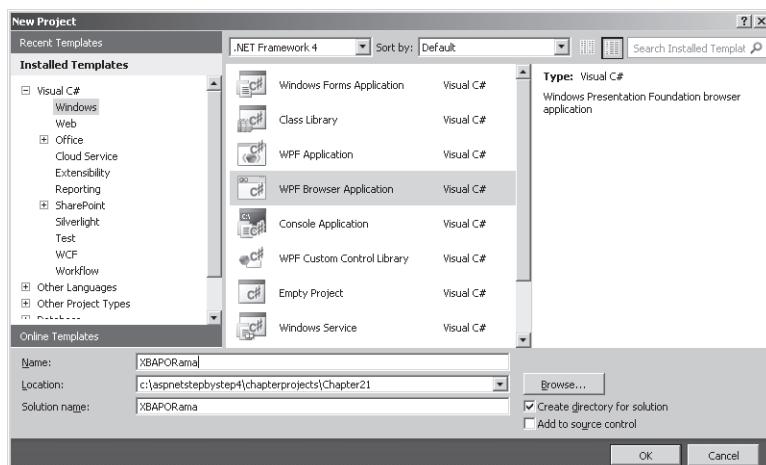
The upside of deploying an application as an XBAP is that it works in very much the same way that a Windows-based desktop application works (though with greatly reduced permissions and tightened security). For example, the application can handle mouse click events and can respond to control events all on the client side.

Although XBAPs are not related directly to ASP.NET, XBAP content can be hosted in ASP.NET-served pages in the same way that loose XAML content can be served. That is, you can make redirects to XBAP files or host XBAP files from within <iframe> HTML elements.

Microsoft Visual Studio includes a wizard for generating XBAPs that can present WPF content. In addition, the user interface elements contained in the WPF content can respond to events and messages the same way as any other desktop application can. When browsers surf to your XBAPs (which are ultimately deployed through Internet Information Services), they will have a very desktop-like experience in terms of user interface rendering and responsiveness, even though the application is running in a browser. The following exercise illustrates how to create an XBAP.

Creating an XBAP

1. Start Visual Studio and click File, New Project. Go to the Windows application templates and select WPF Browser Application. Name the Application *XBAPORama*, as shown here:



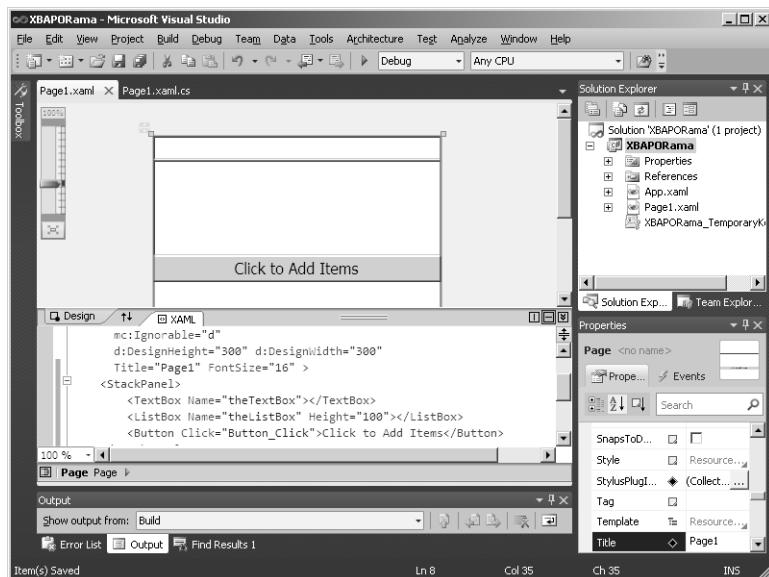
2. Visual Studio should have created for you a new XBAP that includes a page and an application XAML file set. The file names are Page1.xaml/Page1.xaml.cs and App.xaml/App.xaml.cs. This is very similar to the ASP.NET Web Form application structure in that there is a markup file that contains the bulk of the UI and a code file that implements functionality to be run on the client. Visual Studio should show the Page1.xaml file, which contains a *Grid* layout panel.
3. Change the layout panel from a *Grid* to a *StackPanel* so that it is simpler to work with. With a *StackPanel*, you can drop in controls and not worry about creating grid columns and rows:

```
<Page x:Class="XBAPORama.Page1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="300"
    Title="Page1">
    <StackPanel>
    </StackPanel>
</Page>
```

4. Modify the XAML a bit more. Change the *FontSize* property for the *Page* to **16**. Nest the following controls in the *StackPanel*: a *TextBox*, a *ListBox*, and a *Button*. WPF works very similarly to ASP.NET in that you can name controls in the markup file (the XAML file) and they will appear as programmatic elements in the code behind. Set the *Name* property for the *TextBox* to "theTextBox" and set the *Name* property of the *ListBox* to "theListBox" so that you can refer to them in the code files. Finally, set the *Height* property of the *ListBox* to 100 so that it will show up even if it is empty:

```
<Page x:Class="XBAPORama.Page1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="300"
    Title="Page1" FontSize="16">
    <StackPanel>
        <TextBox Name="theTextBox"></TextBox>
        <ListBox Name="theListBox" Height="100"></ListBox>
        <Button>Click to Add Items</Button>
    </StackPanel>
</Page>
```

The Designer should show all the controls in the *StackPanel* like this:



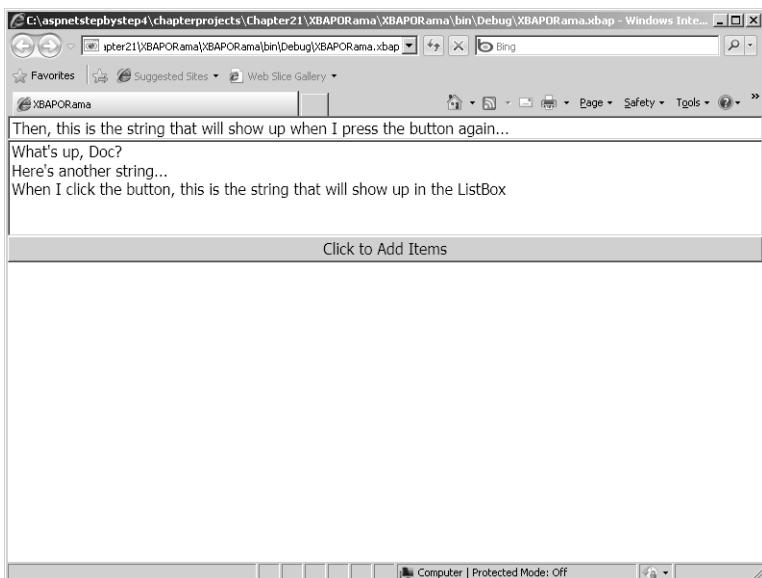
5. Double-click the button to add a handler. Visual Studio creates a handler for the button click. You can find the handler in the code file for the page. Because you didn't name the *Button*, Visual Studio gave the handler a default name of *Button_Click*. The method looks very much like the ASP.NET button click handlers except the second argument is a *RoutedEventArgs* instead of the .NET typical *EventArgs*.
6. Implement the handler by adding whatever is in the *TextBox* to the *ListBox*. It should feel almost like you are programming a Web Form—the code model is very similar:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace XBAPORama
{
    /// <summary>
    /// Interaction logic for Page1.xaml
    /// </summary>
    public partial class Page1 : Page
```

```
{  
    public Page1()  
    {  
        InitializeComponent();  
    }  
    private void Button_Click(object sender, RoutedEventArgs e)  
    {  
        this.listBox.Items.Add(this.textBox.Text);  
    }  
}
```

7. Press Ctrl+F5 from within Visual Studio to run the application in the browser. When you type text into the *TextBox* and click the *Button*, the code running on the client side will add the contents of the *TextBox* to the *ListBox*, as follows (notice the .xbap extension at the end of the file name in the URL):



Although this example does not strictly run in ASP.NET, it does show an alternative way of producing content. When you compiled the application, Visual Studio created a few files including XBAPORama.xbap and XBAPORama.exe. You can include this content as part of an ASP.NET site by including the XBAP, the EXE, and the manifest files that resulted from the compilation in a folder in an ASP.NET application. You do that shortly.

WPF Content and Web Applications

You can serve WPF content from an ASP.NET application in much the same way that ASP.NET serves other content. You can include loose XAML files in a Web application, or you can host some specific WPF content in an `<iframe>` HTML element. This exercise illustrates how you can use WPF content in an ASP.NET application.

Adding XAML content to a site

1. Create a new Empty ASP.NET Web Application project in Visual Studio. Name the project *XAMLOrama*.
2. Use Visual Studio to add a new text file to the project. Right-click the XAMLOrama project node in Visual Studio, and click Add, New Item. Select a text file type from the templates.
3. Rename the file so that it has an .xaml extension. This file shows a paper airplane drawing, so name the file *PaperAirplane.xaml*. The Visual Studio XAML designer might show an error right away because there's no content yet. This is not a problem because you add content in the next step.
4. Add some XAML content to the file, starting by defining the top-level layout node. Include the following XML namespaces and make the window 750 units wide:

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="750">

</Page>
```

All WPF layouts begin with a top-level node. In this case, the node is a *Page* so that it will show up in the client's browser.

5. Add a *Grid* to the page, and add two row definitions and two column definitions:

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="750">
    <Grid>
      <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition Height="100"/>
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition Width="25"/>
      </Grid.ColumnDefinitions>
    </Grid>
</Page>
```

6. Add WPF elements to the grid. Add a *Canvas* to the upper left corner of the *Grid*, and make the *Background SkyBlue*. Add two *Slider* controls to the *Grid*, too. The first *Slider* controls the X position of the airplane. Name the *Slider sliderX*. Put the slider into

row 1, and use the *ColumnSpan* to stretch the *Slider* across two columns. The maximum value of this slider should be 500. Orient the second *Slider* vertically and configure it to occupy column 1 in the *Grid*. Use the *RowSpan* to stretch the *Slider* across both rows. This slider controls the rotation of the airplane. Name this *Slider* *sliderRotate*. Its maximum value should be 360.

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="750">
    <Grid>
        <!-- Grid column and row definitions are here... -->
        <Canvas Background="SkyBlue" Grid.Row="0"
               Grid.Column="0">
            </Canvas>
            <Slider x:Name="sliderRotate" Orientation="Vertical"
                    Grid.Row="0"
                    Minimum="0" Maximum="360"
                    Grid.Column="1"></Slider>
            <Slider x:Name="sliderX" Maximum="500"
                    Grid.Column="0" Grid.Row="1"
                    Grid.ColumnSpan="2"></Slider>
        </Grid>
    </Page>
```

7. Add the airplane and connect it to the sliders using XAML data binding. Here's how: Create the airplane drawing using a WPF *Path*. The *Path* draws a series of line segments using a specific pen. Make the *Stroke* Black and set the the *StrokeThickness* to 3. The *Path* data should connect the following points. Move the cursor to 0,0, and then draw a line to 250,50, and then to 200,75 to 0,0. Then, move the cursor to 200,75 and draw a line to 190,115 and another line to 180,85 to 0,0. Next, move the cursor to 180,85 and draw a line to 140,105 and then to 0,0. Finally, move the cursor to 190,115 and draw a line to 158,93. Set the *Path*'s relationship to the *Top* of the *Canvas* as 200. Bind the *Path*'s relationship to the *Left* of the *Canvas* to *sliderX*'s *Value*. Finally, add a *RenderTransform* to the *Path* and include a *RotateTransform*. Bind the *RotateTransform*'s *Angle* to *sliderRotate*'s *Value*. Set the *Path*'s *RenderTransformOrigin* to .5, .5. Here's the *Path* code:

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="750">

    <Grid>
        <!-- Grid column and row definitions are here... -->
        <Canvas Background="SkyBlue" Grid.Row="0"
               Grid.Column="0">
            <Path Stroke="Black" StrokeThickness="2" Fill="White"
                  Data="M0,0 L250,50 L200,75 L0,0 M200,75 L190,115 L180,85
                        L0,0 M180,85 L140,105 L0,0 M190,115 L158,93"
                  RenderTransformOrigin=".5, .5"
                  Canvas.Top="200"
                  Canvas.Left="{Binding ElementName=sliderX, Path=Value}" >
```

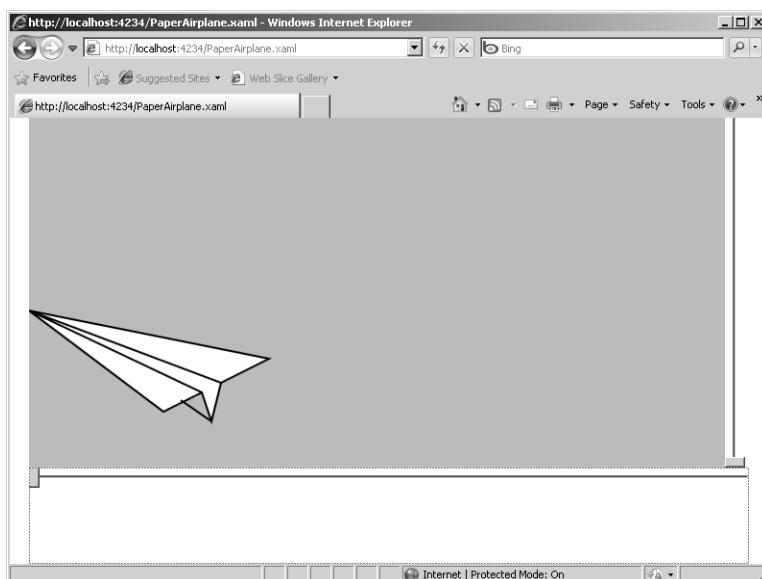
```

<Path.RenderTransform>
    <RotateTransform Angle=
        "{Binding ElementName=sliderRotate, Path=Value}"/>
</Path.RenderTransform>
</Path>
</Canvas>
<!-Sliders go here... -->
</Grid>
</Page>

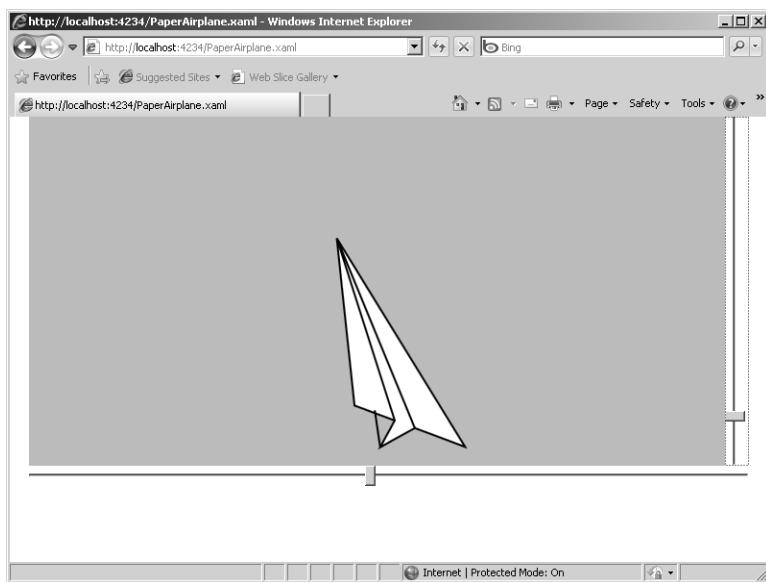
```

After setting up the *Canvas*, the *Path*, and the *Sliders* in the grid, you should see it appear in Visual Studio.

8. Add these references to the project: *WindowsBase*, *PresentationCore*, and *PresentationFramework* by right-clicking the References node in Solution Explorer and clicking Add Reference. Look in the .NET tab of the Add Reference dialog box to find these assemblies. Run the page. Because Visual Studio doesn't allow you to run loose XAML files directly, you need to navigate from another page. Add a new page to the application. Name it *Default.aspx*. Add a *Hyperlink* to the Default.aspx page and set the *NavigationUrl* property to *PaperAirplane.xaml*. Surf to the default page and click the hyperlink that loads the XAML file in the browser. It should appear like this:



9. Experiment by moving the sliders. Because the vertical slider controls the angle of rotation, moving it up causes the airplane to spin in a clockwise direction. Because the horizontal slider is connected to the *Path's Canvas.Left* property, moving the horizontal slider moves the plane along the x-axis, like this:



10. Integrate the new WPF content with some HTML. Add a new Page to the XAMLOrama file by right-clicking the XAMLOrama node in Solution Explorer and adding a new Web page. Name the page *PaperAirplane.aspx*. Add an *<iframe>* tag to the page in between the *<div>* tags that Visual Studio provides. Set the *<iframe>* *height* to **500** and the *width* to **750**. Finally, set the *<iframe>* *src* to **PaperAirplane.xaml**.

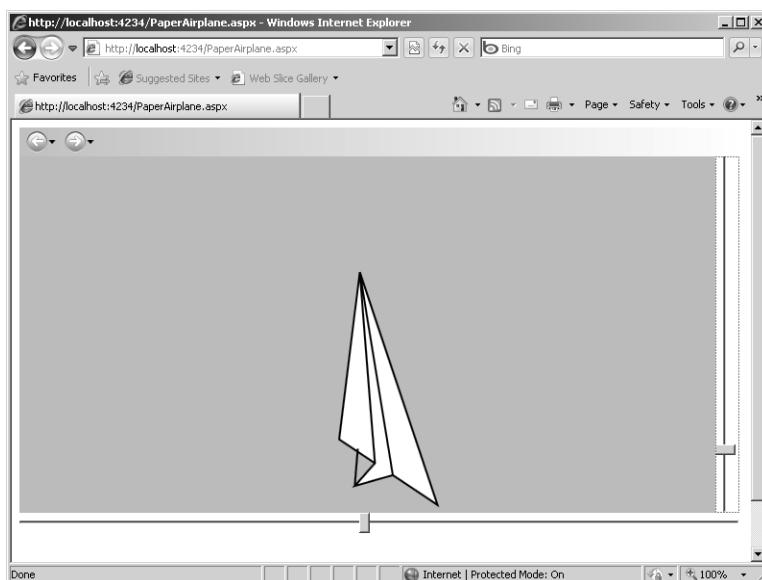
```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="PaperAirplane.aspx.cs"
Inherits="PaperAirplane" %>

<!DOCTYPE html PUBLIC "...">>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>

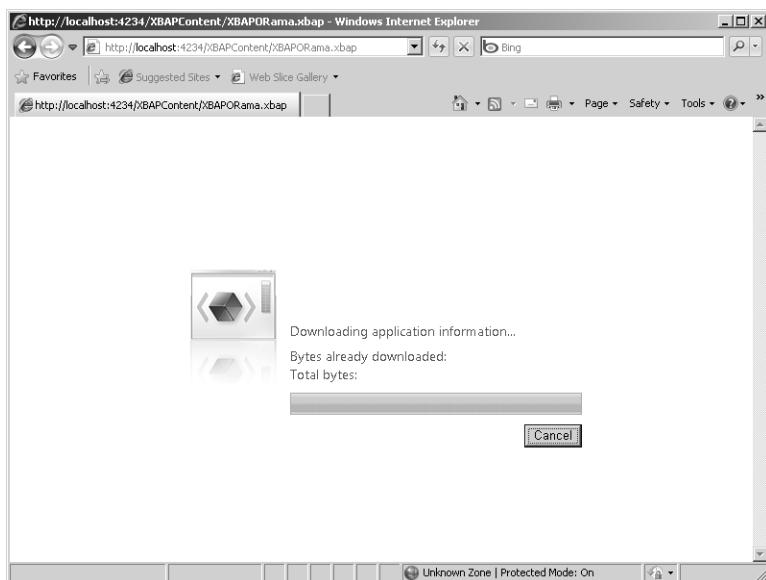
            <iframe height="500"
                width="750"
                src="paperairplane.xaml"></iframe> <br />
        </div>
    </form>
</body>
</html>
```

11. Run the page. The PaperAirplane.xaml content appears in a frame in the page. The XAML content has the same functionality in the frame as it did when it was run in the browser:



Because this is rendered from a typical ASP.NET page, you could include ASP.NET server controls along with the WPF content.

12. Add the XBAP content from the previous example to this site. First, create a new folder under the Project node in Solution Explorer. Name the folder *XBAPContent*. Right-click the new folder and click Add, Existing Item. Navigate to the previous example's bin directory (on my computer, it is C:\aspnetstepbystep4\chapterprojects\Chapter21\XBAPORama\XBAPORama\bin\Debug). Add XBAPORama.xbap, XBAPORama.exe, and XBAPORama.exe.manifest to this XBAPORama ASP.NET project.
13. Add a new link to the Default.aspx page. Set the *NavigationUrl* property to the XBAPORama.xbap file in the XBAPContent folder. Run the application and click the link that redirects to the XBAP content. You will see the XBAPORama.xbap content in the browser. The Web server downloads the XBAP content (you can see a little status bar in the browser, as shown in the following graphic). Try adding some items to the list box to ensure that it works.



Here is the XBAP content running from in the ASP.NET site.



This example illustrates how it is possible to integrate HTML with XAML-based content. You also saw that it is possible to include XBAP content in an ASP.NET site. Although these techniques lie somewhat outside of the usual ASP.NET pipeline, XBAP-based and XAML-based WPF content is still useful in many cases. A full investigation of WPF is beyond the scope of this book. WPF and XAML offer entirely new ways to present content to the end user. Because it is such new technology, the different ways it can be exploited are only now being invented and discovered.

What About Silverlight?

As a Web developer, you have probably been unable to avoid hearing the buzz about Silverlight. Until now, the only effective way to produce dynamic Web content has been through Macromedia Flash. Flash is a plug-in for rendering dynamic content over the Web (that is, animations). However, with the advent of WPF and its dynamic content capabilities, now there is a markup technology that rivals Flash in raw capability if you can find a way to deliver it to the browser. Although other dynamic content technologies certainly have worked, they have had some serious shortcomings for developers. Silverlight changes this.

Silverlight is a platform-independent WPF rendering engine. Without Silverlight, the only way to render WPF content in a browser is to run Internet Explorer or the Firefox browser with the XAML plug-in. Silverlight is packaged as an ActiveX Control for the Microsoft environment. For example, the Apple Safari browser is supported by Silverlight. Visual Studio 2010 includes full support for Silverlight applications. You visit Silverlight in Chapter 24.

Chapter 21 Quick Reference

To	Do This
Add an XAML file to your site	Right-click the project node in the Visual Studio Solution Explorer. Click Add New Item. Select Text File from the available templates. Be sure to name the file with an .xaml extension.
Declare a Page within the XAML file	At the top of the file, add a beginning <code><Page></code> tag and an ending <code></Page></code> tag. Using WPF within XAML requires the standard WPF namespace “ <code>http://schemas.microsoft.com/winfx/2006/xaml/presentation</code> ” and the keywords namespace “ <code>http://schemas.microsoft.com/winfx/2006/xaml</code> ” (which is often mapped to “ <code>x</code> ”).
Add a <i>Canvas</i> to the <i>Page</i>	Use the <code><Canvas></code> opening tag and the <code></Canvas></code> closing tag. Nest objects you’d like displayed in the canvas between the opening and closing tags.
Add content to the <i>Canvas</i>	Nest objects you’d like to appear on the canvas between the <code><Canvas></code> opening tag and the <code></Canvas></code> closing tag. Assign positions within the canvas using the <code>Canvas.Top</code> and <code>Canvas.Right</code> properties.
Add a <i>Grid</i> to a <i>Page</i>	Declare a <code><Grid></code> opening tag and a <code></Grid></code> closing tag on the page. Use the <i>Grid</i> ’s <code>RowDefinitions</code> and the <i>Grid</i> ’s <code>ColumnDefinitions</code> properties to define the rows and columns.
Add content to the <i>Grid</i>	Nest objects you’d like to appear on the canvas between the <code><Grid></code> opening tag and the <code></Grid></code> closing tag. Assign positions within the grid using the <code>Grid.Row</code> and <code>Grid.Column</code> properties.
Create an XAML-based browser application	Select File, New Project in Visual Studio. From the Windows application templates, choose WPF Browser Application. Visual Studio will create an XBAP application for you, starting with a simple page. Add WPF controls and handlers to the page. If you want to run the XBAP control from an ASP.NET site, just make sure the XBAP.exe, and manifest files are available to the ASP.NET Web Project.

Chapter 23

AJAX

After completing this chapter, you will be able to

- Understand the problem AJAX solves.
- Understand ASP.NET support for AJAX.
- Write AJAX-enabled Web sites.
- Take advantage of AJAX as necessary to improve the user's experience.

This chapter covers AJAX, possibly the most interesting feature added to ASP.NET recently. AJAX stands for Asynchronous JavaScript and XML, and it promises to produce an entirely new look and feel for Web sites throughout the world.

Rich Internet Applications

Software evolution always seems to happen in this typical fashion: Once a technology is grounded firmly (meaning the connections between the parts work and the architecture is fundamentally sound), upgrading the end user's experience becomes a much higher priority. Application technology is in this stage, and the general term for this kind of application is a Rich Internet Application (RIA). AJAX is one means of producing Rich Internet Applications. (Microsoft Silverlight is another popular means of creating RIAs.)

The primary reason for the existence of AJAX is to improve the standard HTTP GET/POST idiom with which Web users are so familiar. That is, the standard Web protocol in which entire forms and pages are sent between the client and the server is getting a whole new addition.

Although standard HTTP is functional and well understood by Web developers, it does have certain drawbacks—the primary one is that the user is forced to wait for relatively long periods while pages refresh. This has been a common problem in all event-driven interfaces. (The Windows operating system is one of the best examples.) AJAX introduces technology that shields end users from having to wait for a whole page to post.

Think back to the way HTTP typically works. When you make a request (using GET or POST, for example), the Web browser sends the request to the server, but you can do nothing until the request finishes. That is, you make the request and wait—watching the little progress indicator in the browser. When the request returns to the browser, you can begin using the

application again. The application is basically useless until the request returns. In some cases, the browser's window even goes completely blank. Web browsers have to wait for Web sites to finish an HTTP request in much the same way that Windows-based programs have to wait for message handlers to complete their processing. (Actually, if the client browser uses a multithreaded user interface such as Windows Internet Explorer, users can usually cancel the request—but that's all they can really do.) You can easily demonstrate this problem by introducing a call to *System.Threading.Thread.Sleep* inside the *Page_Load* method of an ASPX page. By putting the thread to sleep, you force the end user to wait for the request to finish.

The AJAX solution to this problem is to introduce some way to handle the request asynchronously. What if there were a way to introduce asynchronous background processing into a Web site so that the browser would appear much more responsive to the user? What if (for certain applications) making an HTTP request didn't stall the entire browser for the duration of the request, but instead seemed to run the request in the background, leaving the foreground unhindered and changing only the necessary portion of the rendered page? The site would present a much more continuous and smooth look and feel to the user. As another example, what if ASP.NET included some controls that injected script into the rendered pages that modified the HTML Document Object Model, providing more interaction from the client's point of view? Well, that's exactly what ASP.NET AJAX support is designed to do.

What Is AJAX?

AJAX formalizes a style of programming meant to improve the UI responsiveness and visual appeal of Web sites. Many AJAX capabilities have been available for a while now. AJAX consolidates several good ideas and uses them to define a style of programming and extends the standard HTTP mechanism that is the backbone of the Internet. Like most Web application development environments, ASP.NET takes advantage of HTTP capabilities in a very standard way. The browser usually initiates contact with the server using an HTTP GET request, followed by any number of POSTs. The high-level application flow is predicated upon sending a whole request and then waiting for an entire reply from the server. Although the ASP.NET server-side control architecture greatly improves back-end programming, users still get their information a whole page at a time. It operates almost like the mainframe/terminal model popular during the 1970s and early 1980s. However, this time the terminal is one of many modern sophisticated browsers and the mainframe is replaced by a Web server (or Web farm).

The standard HTTP round-trip has been a useful application strategy, and the Web grew up using it. While the Web was developing in the late 1990s, browsers had widely varying degrees of functionality. For example, browsers ranged all the way from the rudimentary

America Online Browser (which had very limited capabilities) to cell phones and personal digital assistants (PDAs), to more sophisticated browsers such as Internet Explorer and Netscape Navigator, which were rich in capability. For instance, Internet Explorer supports higher level features such as JavaScript and Dynamic HTML. This made striking a balance between usability of your site and the reach of your site very difficult prior to the advent of ASP.NET.

However, the majority of modern computing platforms can run a decent browser that can process client-side scripting. These days, most computing environments run a modern operating system, such as the Windows Vista or Windows 7 operating systems, or even Macintosh OS X. These environments run browsers fully capable of supporting XML and JavaScript. With so many Web client platforms supporting this functionality, it makes sense to take advantage of the capabilities. As you see later in this chapter, AJAX makes good use of these modern browser features to improve the user experience.

In addition to extending standard HTTP, AJAX is also a very clever way to use the Web service idiom. Web services are traditionally geared toward enterprise-to-enterprise business communications. However, Web services are also useful on a smaller scale for handling Web requests out of band. ("Out of band" simply means making HTTP requests using other methods instead of the standard page posting mechanism.) AJAX uses Web services behind the scenes to make the client UI more responsive than it is for traditional HTTP GETs and POSTs. This chapter describes how this works, especially in the section titled "Extender Controls" later in the chapter, which describes the ASP.NET AJAX Control Toolkit extender controls.

ASP.NET and AJAX

One of the primary changes AJAX brings to Web programming is that it depends on the browser taking an even more active role in the process. Instead of the browser simply rendering streams of HTML and executing small custom-written script blocks, AJAX includes some new client-script libraries to facilitate the asynchronous calls back to the server. AJAX also includes some basic server-side components to support these new asynchronous calls coming from the client. There's even a community-supported AJAX Control Toolkit available for the ASP.NET AJAX implementation. Figure 23-1 shows the organization of ASP.NET AJAX support.

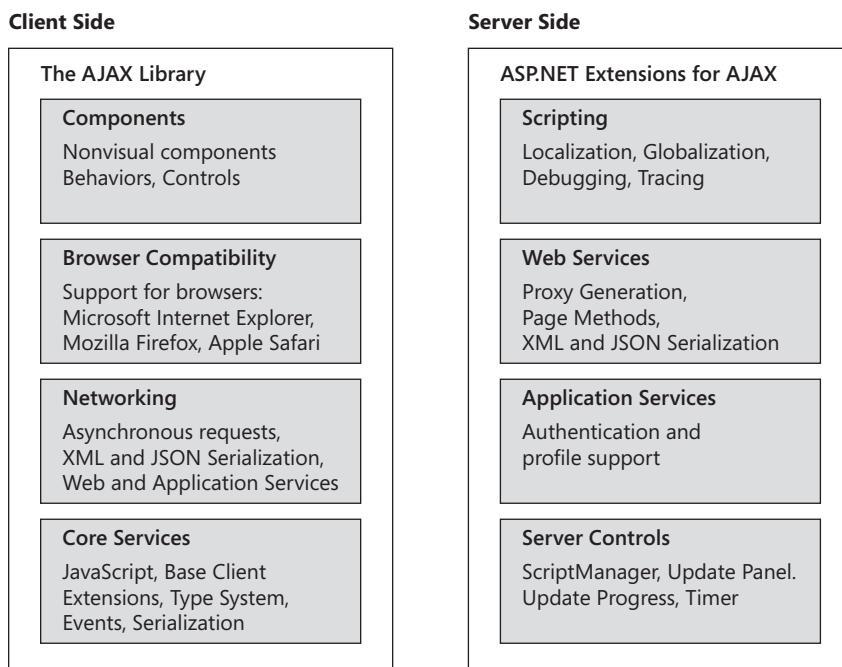


FIGURE 23-1 The conceptual organization of ASP.NET AJAX support layers.

Reasons to Use AJAX

If traditional ASP.NET development is so entrenched and well established, why would you want to introduce AJAX? At first glance, AJAX seems to introduce some new complexities into the ASP.NET programming picture. In fact, it seems to reintroduce some programming idioms that ASP.NET was designed to deprecate (such as overuse of client-side script). However, AJAX promises to produce a richer experience for the user. Because ASP.NET support for AJAX is nearly seamless, the added complexities are well mitigated. When building a Web site, there are a few reasons you might choose to enable your ASP.NET site for AJAX:

- AJAX improves the overall efficiency of your site by performing, when appropriate, parts of a Web page's processing in the browser. Instead of waiting for the entire HTTP protocol to get a response from the browser, you can push certain parts of the page processing to the client to help the client to react much more quickly. Of course, this type of functionality has always been available—as long as you're willing to write the code to make it happen. ASP.NET AJAX support includes a number of scripts so that you can get a lot of browser-based efficiency by simply using a few server-side controls.
- ASP.NET AJAX introduces to a Web site UI elements usually found in desktop applications, such as rectangle rounding, callouts, progress indicators, and pop-up windows

that work for a wide range of browsers (more browser-side scripting—but most of it has been written for you).

- AJAX introduces partial-page updates. By refreshing only the parts of the Web page that have been updated, the user's wait time is reduced significantly. This brings Web-based applications much closer to desktop applications with regard to perceived UI performance.
- AJAX is supported by most popular browsers—not just Internet Explorer. It works for Mozilla Firefox and Apple Safari, too. Although it still requires some effort to strike a balance between UI richness and the ability to reach a wider audience, the fact that AJAX depends on features available in most modern browsers makes this balance much easier to achieve.
- AJAX introduces a huge number of new capabilities. Whereas the standard ASP.NET control and page-rendering model provides great flexibility and extensibility for programming Web sites, AJAX brings in a new concept—the extender control. Extender controls attach to existing server-side controls (such as the *TextBox*, *ListBox*, and *DropDownList*) at run time and add new client-side appearances and behaviors to the controls. Sometimes extender controls can even call a predefined Web service to get data to populate list boxes and such (for example, the *AutoComplete* extender).
- AJAX improves on ASP.NET Forms Authentication and profiles and personalization services. ASP.NET support for authentication and personalization provides a great boon to Web developers—and AJAX just sweetens the offerings.

These days, when you browse different Web sites, you run into many examples of AJAX-style programming. Here are some examples:

- Colorado Geographic: <http://www.coloradogeographic.com/>
- Cyber Homes: <http://www.cyberhomes.com/default.aspx>
- Component Art: <http://www.componentart.com/>

Real-World AJAX

Throughout the 1990s and into the mid-2000s, Web applications were nearly a throwback to 1970s mainframe and minicomputer architectures. However, instead of a single large computer serving dumb terminals, Web applications consist of a Web server (or a Web farm) connected to smart browsers capable of fairly sophisticated rendering capabilities. Until recently, Web applications took their input from HTTP forms and presented output in HTML pages. The real trick in understanding standard Web applications is to see the disconnected and stateless nature of HTTP. Classic Web applications can show only a snapshot of the state of the application.

As this chapter describes, Microsoft supports standard AJAX idioms and patterns in the ASP.NET framework. However, AJAX is more a style of Web programming involving out-of-band HTTP requests than any specific technology.

You've no doubt seen sites engaging the new interface features and stylings available through AJAX programming. Examples include Microsoft.com, Google.com, and Yahoo.com. Very often while browsing these sites, you'll see modern features such as automatic page updates that do not require you to generate a postback explicitly. Modal-type dialog boxes that require your attention appear until you dismiss them. These are all features available through AJAX-style programming patterns and the ASP.NET extensions (for example, a rich set of AJAX server-side controls and extensions) for supporting AJAX.

If you're a long-time Microsoft environment Web developer, you might be asking yourself whether AJAX is something really worthwhile or whether you might be able to get much of the same type of functionality using a tried and true technology such as DHTML.

AJAX in Perspective

Any seasoned Web developer targeting Internet Explorer as the browser is undoubtedly familiar with Dynamic HTML (DHTML). DHTML is a technology that runs at the browser for enabling Windows desktop-style UI elements in the Web client environment. DHTML was a good start, and AJAX brings the promise of more desktop-like capabilities to Web applications.

AJAX makes available wider capabilities than DHTML does. With DHTML, primarily you can change the style declarations of an HTML element through JavaScript. However, that's about as far as it goes. DHTML is very useful for implementing such UI features as having a menu open when the mouse pointer rests on it. AJAX expands on this idea of client-based UI using JavaScript as well as out-of-band calls to the server. Because AJAX is based on out-of-band server requests (rather than relying *only* on a lot of client script code), AJAX has the potential for much more growth in terms of future capabilities than does DHTML.

ASP.NET Server-Side Support for AJAX

Much of ASP.NET support for AJAX resides in a collection of server-side controls responsible for rendering AJAX-style output to the browser. Recall from Chapter 3, "The Page Rendering Model," that the entire page-rendering process of an ASP.NET application is broken down into little bite-sized chunks. Each individual bit of rendering is handled by a class derived from *System.Web.UI.Control*. The entire job of a server-side control is to render output that places HTML elements in the output stream so that they appear correctly in the browser. For example, *ListBox* controls render a `<select>` tag. *TextBox* controls render an

<input type="text" /> tag. ASP.NET AJAX server-side controls render AJAX-style script and HTML to the browser.

ASP.NET AJAX support consists of these server-side controls along with client code scripts that integrate to produce AJAX-like behavior. The following subsections describe the most frequently used AJAX-oriented ASP.NET server controls: *ScriptManager*, *ScriptManagerProxy*, *UpdatePanel*, *UpdateProgress*, and *Timer*.

***ScriptManager* Control**

The *ScriptManager* control manages script resources for the page. The *ScriptManager* control's primary action is to register the AJAX Library script with the page so that the client script can use type system extensions. The *ScriptManager* also makes possible partial-page rendering and supports localization as well as custom user scripts. The *ScriptManager* assists with out-of-band calls back to the server. Any ASP.NET site wishing to use AJAX must include an instance of the *ScriptManager* control on any page using AJAX functionality.

***ScriptManagerProxy* Control**

Scripts on a Web page often require a bit of special handling in terms of how the server renders them. Typically, the page uses a *ScriptManager* control to organize the scripts at the page level. Nested components such as content pages and user controls require the *ScriptManagerProxy* to manage script and service references to pages that already have a *ScriptManager* control.

This is most notable in the case of master pages. The master page typically houses the *ScriptManager* control. However, ASP.NET throws an exception if a second instance of *ScriptManager* is found in a given page. So, what would content pages do if they needed to access the *ScriptManager* control that the master page contains? The answer is that the content page should house the *ScriptManagerProxy* control and work with the true *ScriptManager* control through the proxy. Of course, as mentioned, this also applies to user controls as well.

***UpdatePanel* Control**

The *UpdatePanel* control supports partial-page updates by tying together specific server-side controls and events that cause them to render. The *UpdatePanel* control causes only selected parts of the page to be refreshed instead of the whole page (as happens during a typical HTTP postback).

UpdateProgress Control

The *UpdateProgress* control coordinates status information about partial-page updates as they occur in *UpdatePanel* controls. The *UpdateProgress* control supports intermediate feedback for long-running operations.

Timer Control

The *Timer* control issues postbacks at defined intervals. Although the *Timer* control will perform a typical postback (posting the whole page), it is especially useful when coordinated with the *UpdatePanel* control to perform periodic partial-page updates.

AJAX Client Support

ASP.NET AJAX client-side support is centered around a set of JavaScript libraries. The following layers are included in the ASP.NET AJAX script libraries:

- The browser compatibility layer assists in managing compatibility across the most frequently used browsers. Whereas ASP.NET by itself implements browser capabilities on the server end, this layer handles compatibility on the client end (the browsers supported include Internet Explorer, Mozilla Firefox, and Apple Safari).
- The ASP.NET AJAX core services layer extends the usual JavaScript environment by introducing classes, namespaces, event handling, data types, and object serialization that are useful in AJAX programming.
- The ASP.NET AJAX base class library for clients includes various components, such as components for string management and for extended error handling.
- The networking layer of the AJAX client-side support manages communication with Web-based services and applications. The networking layer also handles asynchronous remote method calls.

The pièce de résistance of ASP.NET AJAX support is the community-supported Control Toolkit. Although all the features mentioned previously provide solid infrastructure for ASP.NET AJAX, AJAX isn't very compelling until you add a rich tool set.

ASP.NET AJAX Control Toolkit

The ASP.NET AJAX Control Toolkit is a collection of components (and samples showing how to use them) encapsulating AJAX capabilities. When you browse through the samples, you can get an idea of the kind of user experiences available through the controls and extenders.

The Control Toolkit also provides a powerful software development kit for creating custom controls and extenders. You can download the ASP.NET AJAX Control Toolkit from the ASP.NET AJAX Web site.

The AJAX Control Toolkit is a separate download and not automatically included with Microsoft Visual Studio 2010. The latest version is 3.0, which was made available at the end of September 2009. See <http://asp.net/ajax/ajaxcontroltoolkit/> for details. You can download the binaries or the source code. If you aren't interested in the source code, you only need to make a reference to the AjaxControlToolkit.dll assembly in your project.

If you want to build the toolkit yourself, follow these steps:

1. Download the toolkit source code.
2. After unzipping the Toolkit file, open the AjaxControlToolkit solution file in Visual Studio.
3. Build the AjaxControlKit project.
4. The compilation process produces a file named AjaxControlToolkit.dll in the AjaxControlToolkit\bin directory.
5. Right-click the Toolbox in Visual Studio, and click Choose Items on the menu. Browse to the AjaxControlToolkit.dll file in the AjaxControlToolkit\bin directory and include the DLL. This brings all the new AJAX controls from the toolkit into Visual Studio so that you can drop them in forms in your applications.



Note You can find a wealth of AJAX-enabled server-side controls and client-side scripts available through a community-supported effort. Although they are not quite officially part of the Microsoft AJAX release, the support includes the ASP.NET AJAX community-supported controls (mentioned previously) as well as support for client declarative syntax (XML script) and more. Go to <http://www.asp.net/ajax/> for more information. This site includes links to download the ASP.NET AJAX Control Toolkit, links to some interesting AJAX-enabled sites, video tutorials, and other useful downloads.

AJAX Control Toolkit Potpourri

A number of other extenders and controls are available through a community-supported effort. You can find a link to the AJAX Control Toolkit at <http://www.asp.net/ajax/>. This chapter discusses a few of the controls available from the toolkit. Table 23-1 lists the controls and extenders available through this toolkit.

TABLE 23-1 The ASP.NET Control Toolkit

Component	Description
<i>Accordion</i>	This extender is useful for displaying a group of panes one pane at a time. It's similar to using several <i>CollapsiblePanels</i> constrained to allow only one to be expanded at a time. The <i>Accordion</i> is composed of a group of <i>AccordionPane</i> controls.
<i>AlwaysVisibleControl</i>	This extender is useful for pinning a control to the page so that its position remains constant while content behind it moves and scrolls.
<i>Animation</i>	This extender provides a clean interface for animating page content.
<i>AsyncFileUpload</i>	This control is for uploading a file asynchronously in the background.
<i>AutoComplete</i>	This extender is designed to communicate with a Web service to list possible text entries based on what's already in the text box.
<i>Calendar</i>	This extender is targeted for the <i>TextBox</i> control providing client-side date-picking functionality in a customizable way.
<i>CascadingDropDown</i>	This extender is targeted toward the <i>DropDownList</i> control. It functions to populate a set of related <i>DropDownList</i> controls automatically.
<i>CollapsiblePanel</i>	This extender is targeted toward the <i>Panel</i> control for adding collapsible sections to a Web page.
<i>ConfirmButton</i>	This extender is targeted toward the <i>Button</i> control (and types derived from the <i>Button</i> control) and is useful for displaying messages to the user. The scenarios for which this extender is useful include those requiring confirmation from the user (for example, where linking to another page might cause the end user to lose state).
<i>DragPanel</i>	This is an extender targeted toward <i>Panel</i> controls for adding the capability for users to drag the <i>Panel</i> around the page.
<i>DropDown</i>	This extender implements a Microsoft SharePoint-style drop-down menu.
<i>DropShadow</i>	This extender is targeted toward the <i>Panel</i> control that applies a drop shadow to the <i>Panel</i> .
<i>DynamicPopulate</i>	This extender uses an HTML string returned by a Web service or page method call.
<i>FilteredTextBox</i>	This extender is used to ensure that an end user enters only valid characters in a text box.
<i>HoverMenu</i>	This extender is targeted for any <i>WebControl</i> that has an associated pop-up window for displaying additional content. The pop-up window is activated when the user rests the mouse pointer on the targeted control.

Component	Description
<i>ListSearch</i>	This extender searches items in a designated <i>ListBox</i> or <i>DropDownList</i> based on keystrokes as they're typed by the user.
<i>MaskedEdit</i>	This extender is targeted toward <i>TextBox</i> controls to constrain the kind of text that the <i>TextBox</i> will accept by applying a mask.
<i>ModalPopup</i>	This extender mimics the standard Windows modal dialog box behavior. With the <i>ModalPopup</i> , a page can display content of a pop-up window that focuses attention on itself until it is dismissed explicitly by the end user.
<i>MutuallyExclusiveCheckBox</i>	This extender is targeted toward the <i>CheckBox</i> control. The extender groups <i>CheckBox</i> controls using a key. When a number of <i>CheckBox</i> controls all share the same key, the extender ensures that only a single check box will appear selected at a time.
<i>NoBot</i>	This control attempts to provide CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart)-like bot/spam detection and prevention without requiring any user interaction. Although a noninteractive approach might be bypassed more easily than one requiring actual human interaction can be, this implementation is invisible.
<i>NumericUpDown</i>	This extender is targeted toward the <i>TextBox</i> control to create a control very similar to the standard Windows Edit control with the Spin button. The extender adds "up" and "down" buttons for incrementing and decrementing the value in the <i>TextBox</i> .
<i>PagingBulletedList</i>	This extender is targeted toward the <i>BulletedList</i> control. The extender enables sorted paging on the client side.
<i>PasswordStrength</i>	This extender is targeted toward the <i>TextBox</i> control to help when end users type passwords. Whereas the typical <i>TextBox</i> hides only the actual text, the <i>PasswordStrength</i> extender also displays the strength of the password using visual cues.
<i>PopupControl</i>	This extender is targeted toward all controls. Its purpose is to open a pop-up window for displaying additional relevant content.
<i>Rating</i>	This control renders a rating system from which end users rate something using images to represent their choice (stars are common).
<i>ReorderList</i>	This ASP.NET AJAX control implements a bulleted, data-bound list with items that can be reordered interactively.
<i>ResizableControl</i>	This extender works with any element on a Web page. Once the <i>ResizableControl</i> is associated with an element, the user can resize that control. The <i>ResizableControl</i> puts a handle on the lower right corner of the control.
<i>RoundedCorners</i>	The <i>RoundedCorners</i> extender can be applied to any Web page element to turn square corners into rounded corners.

Component	Description
<i>Seadragon</i>	The <i>Seadragon</i> control renders an image along with buttons for zooming in and out, going to full screen, and panning.
<i>Slider</i>	This extender is targeted to the <i>TextBox</i> control. It adds a graphical slider that the end user can use to change the numeric value in the <i>TextBox</i> .
<i>SlideShow</i>	This extender controls and adds buttons users can use to move between images individually and to play the slide show automatically.
<i>Tabs</i>	This server-side control manages a set of tabbed panels for managing content on a page.
<i>TextBoxWatermark</i>	<i>TextBoxWatermark</i> extends the <i>TextBox</i> control to display a message while the <i>TextBox</i> is empty. When the <i>TextBox</i> contains some text, the <i>TextBox</i> appears as a typical <i>TextBox</i> .
<i>ToggleButton</i>	This extender extends the <i>CheckBox</i> to show custom images reflecting the state of the <i>CheckBox</i> .
<i>UpdatePanelAnimation</i>	This extender provides a clean interface for animating content associated with an <i>UpdatePanel</i> .
<i>ValidatorCallout</i>	<i>ValidatorCallout</i> extends the validator controls (such as <i>RequiredFieldValidator</i> and <i>RangeValidator</i>). The callouts are small pop-up windows that appear near the UI elements containing incorrect data to direct user focus to them.

Getting Familiar with AJAX

Here's a short example to help get you familiar with AJAX. It's a very simple Web Forms application that shows behind-the-scenes page content updates with the *UpdatePanel* server-side control. In this exercise, you create a page with labels showing the date and time that the page loads. One label is outside the *UpdatePanel*, and the other label is inside the *UpdatePanel*. You can see how partial-page updates work by comparing the date and time shown in each label.

Implementing a simple partial-page update

1. Create a new Web site project named *AJAXORama*. Make it an empty, file system-based Web site. Visual Studio 2010 creates AJAX Enabled projects right from the start. Make sure the default.aspx file is open.
2. Add a *ScriptManager* control to the page by dragging one from the Toolbox onto the page. (It is under the *AJAX Extensions* tab in the Toolbox instead of the normal control tab.) Using the AJAX controls requires a *ScriptManager* to appear prior to any other AJAX controls on the page. By convention, the control is usually placed outside the DIV

Visual Studio creates for you. After placing the script manager control on your page, the <body> element in the *Source* view should look like this:

```
<body>
    <form id="form1" runat="server">
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <div>

        </div>
    </form>
</body>
```

3. Drag a *Label* control onto the Default.aspx form. In the Properties pane, give the *Label* control the name *LabelDateTimeOfPageLoad*. Then, drop a *Button* on the form as well. Give it the text *Click Me*. Open the code-behind file (default.aspx.cs) and update the *Page_Load* handler so that the label displays the current date and time:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        this.LabelDateTimeOfPageLoad.Text = DateTime.Now.ToString();
    }
}
```

4. Run the page and generate some postbacks by clicking the button a few times. Notice that the label on the page updates with the current date and time each time you click the button.
5. Add an *UpdatePanel* control to the page. (You can find this control alongside the *ScriptManager* control in the AJAX node in the Visual Studio *Toolbox*.) Then, drop another *Label* from the *Toolbox* into the content area of the *UpdatePanel*. Name the label *LabelDateTimeOfButtonClick*.
6. Add some code to the *Page_Load* method so that the label shows the current date and time:

```
using System;
using System.Data;
using System.Configuration;
```

```

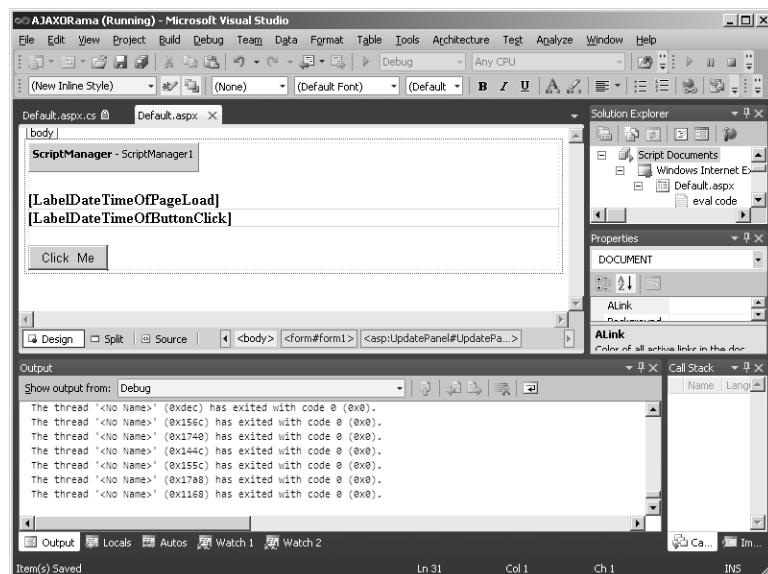
using System.Web;
using System.Web.Security;

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

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        this.LabelDateTimeOfPageLoad.Text = DateTime.Now.ToString();
        this.LabelDateTimeOfButtonClick.Text =
            DateTime.Now.ToString();
    }
}

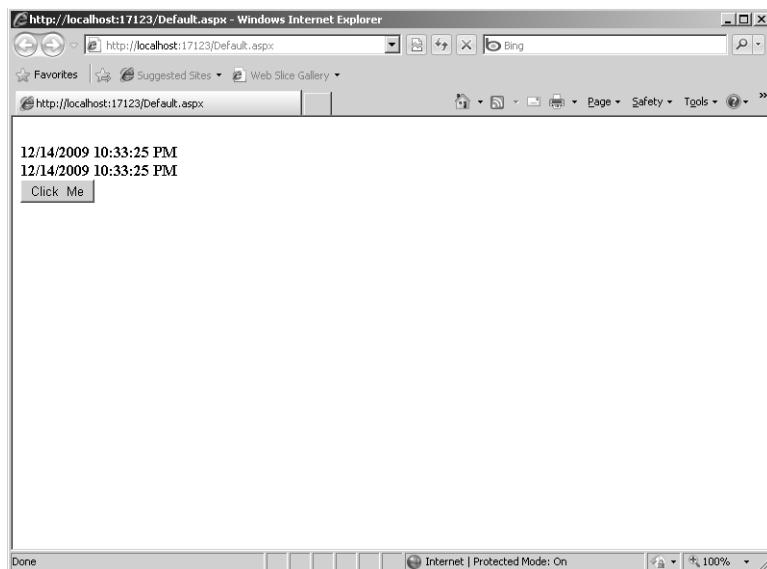
```

The following graphic shows the *UpdatePanel*, *Button*, and *Labels* as displayed in the Visual Studio Designer (there are some line breaks in between so that the page is readable):



- Run the page and generate some postbacks by clicking the button. Both labels should be showing the date and time of the postback (that is, they should show the same time). Although the second label is inside the *UpdatePanel*, the action causing the postback is happening outside the *UpdatePanel*.

The following graphic shows the Web page running without the *Button* being associated with the *UpdatePanel*:



8. Now delete the current button from the form and drop a new button into the *UpdatePanel1* control. Add a *Label* to the *UpdatePanel1* as well. Name the new label *LabelDateTimeOfButtonPress*. Look at the Default.aspx file to see what was produced:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ScriptManager
            ID="ScriptManager1" runat="server" /><br/>
        <asp:Label ID="LabelDateTimeOfPageLoad"
            runat="server"></asp:Label> <br/>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">

            <ContentTemplate>
                <asp:Label ID="LabelDateTimeOfButtonPress"
                    runat="server">
                </asp:Label><br/>
```

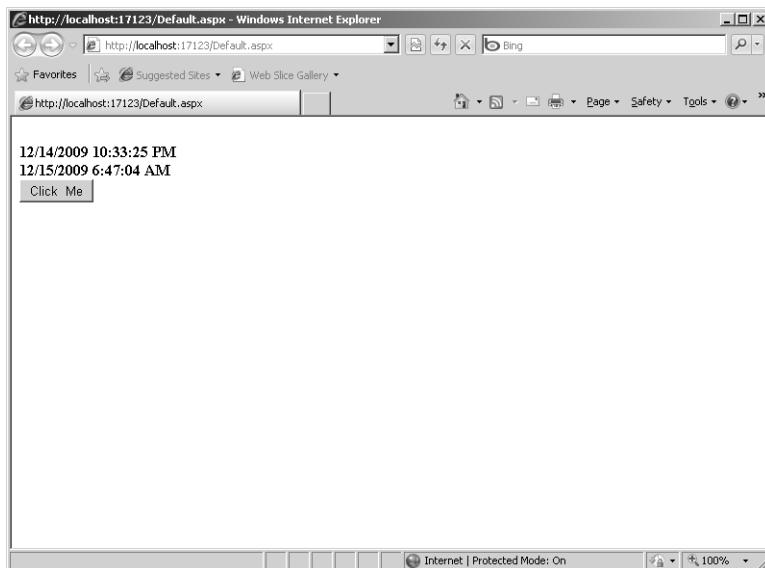
```

<asp:Button ID="Button1"
            runat="server" Text="Click Me" />
        </ContentTemplate>
    </asp:UpdatePanel>
</form>
</body>
</html>

```

The new *Button* should now appear nested inside the *UpdatePanel* along with the new *Label*.

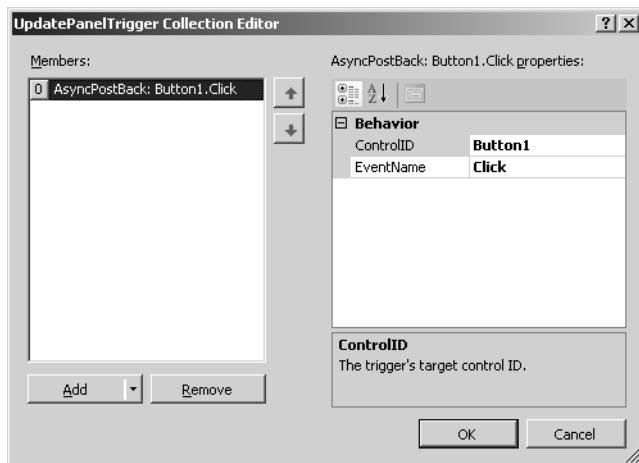
- Run the page and generate some postbacks by clicking the button. Notice that only the label showing the date and time enclosed in the *UpdatePanel* is updated. This is known as a *partial-page update* because only part of the page is actually updated in response to a page action, such as clicking the button. Partial-page updates are also sometimes referred to as *callbacks* rather than postbacks. The following graphic shows the Web page running with the *Button* being associated with the *UpdatePanel*:



- Add an *UpdatePanel* trigger. Because the second label and the button are both associated with the single *UpdatePanel*, only the second *Label* is updated in response to the postback generated by the button. If you could set up partial-page updates based only on elements tied to a single *UpdatePanel*, that would be fairly restrictive. As it turns out, the *UpdatePanel* supports a collection of triggers that generate partial-page updates. To see how this works, you need to first move the button outside the *UpdatePanel* (so that the button generates a full normal postback). The easiest way is simply to drag a button onto the form, making sure it lands outside the *UpdatePanel*.

Because the button is outside the *UpdatePanel* again, postbacks generated by the button are no longer tied solely to the second label, and the partial-page update behavior you saw in step 9 is again nonfunctional.

11. Update the *UpdatePanel's Triggers* collection to include the *Button's Click* event. With the Designer open, select the *UpdatePanel*. Go to the properties Window and choose *Triggers*.
12. Add a trigger and set the control ID to the button's *ID* and the event to *Click* as shown in the following graphic:



(Note that the handy drop-down lists for each property assist you with this selection.) Run the page. Clicking the button should now generate a callback (causing a partial-page update) in which the first label continues to show the date and time of the original page load and the second label shows the date and time of the button click. Pretty cool!

Async Callbacks

As you know by now, standard Web pages require the browser to instigate postbacks. Many times, postbacks are generated by clicking a *Button* control (in ASP.NET terms). However, you can enable most ASP.NET controls to generate postbacks as well. For example, if you'd like to receive a postback whenever a user selects an item in a *DropDownList*, just flip the *AutoPostBack* property to *true*, and the control will generate the normal postback whenever the selected item changes.

In some cases, an entire postback is warranted for events such as when the selected item changes. However, in most cases generating postbacks is distracting for users and

leads to very poor performance of your page. That's because standard postbacks refresh the whole page.

ASP.NET AJAX support introduces the notion of the *asynchronous* postback by using JavaScript running inside the client page. The *XMLHttpRequest* object posts data to the server—making an end run around the normal postback. The server returns data as XML, JSON, or HTML and has to refresh only part of the page. The JavaScript running in the page replaces old HTML in the Document Object Model with new HTML based on the results of the asynchronous postback.

If you've done any amount of client-side script programming, you can imagine how much work doing something like this can be. Performing asynchronous postbacks and updating pages usually requires a lot of JavaScript.

The *UpdatePanel* control you just used in the preceding exercise hides all of the client-side code and also the server-side plumbing. Also, because of the well-architected server-side control infrastructure in ASP.NET, the *UpdatePanel* maintains the same server-side control model you're used to seeing in ASP.NET.

The Timer

In addition to causing partial-page updates through an event generated by a control (such as a button click), AJAX includes a timer to cause regularly scheduled events. You can find the *Timer* control alongside the other standard AJAX controls in the Toolbox. By dropping a *Timer* on a page, you can generate automatic postbacks to the server.

Some uses for the *Timer* include a "shout box"—like an open chat where a number of users type in messages and they appear near the top like a conversation. Another reason you might like an automatic postback is if you wanted to update a live Web camera picture or to refresh some other frequently updated content.

The *Timer* is very easy to use—simply drop it on a page that hosts a *ScriptManager*. The default settings for the timer cause the timer to generate postbacks every minute (every 60,000 milliseconds). The *Timer* is enabled by default and begins firing events as soon as the page loads.

Here's an exercise using the *Timer* to write a simple chat page that displays messages from a number of users who are logged in. The conversation is immediately updated for the user typing in a message. However, users who have not refreshed since the last message don't get to see it—unless they perform a refresh. The page uses a *Timer* to update the conversation automatically. At first, the entire page is refreshed. Then, the chat page uses an *UpdatePanel* to update only the chat log (which is the element that changes).

Using the *Timer* to create a chat page

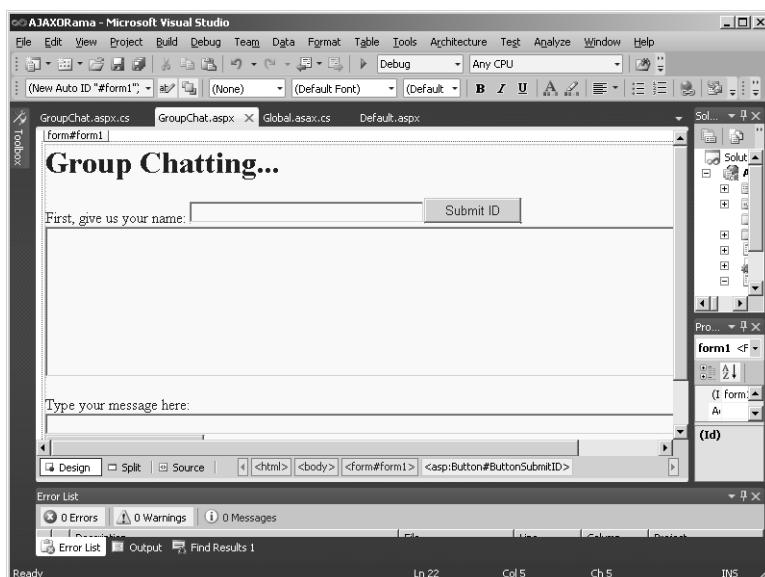
1. Open the AJAXORama application if it's not already open. The first step is to create a list of chat messages that can be seen from a number of different sessions. Add a global application class to the project by right-clicking in Solution Explorer and clicking Add New Item. Choose Global Application Class as the type of file to add. This adds files named Global.asax and Global.asax.cs to your Web site.
2. Update the *Application_Start* method in Global.asax.cs to create a list for storing messages and add the list to the application cache.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;

namespace AJAXORama
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            // Code that runs on application startup
            List<string> messages = new List<string>();
            HttpContext.Current.Cache["Messages"] = messages;
        }
        // other generated code is here...
    }
}
```

3. Create a chat page by adding a new page to the Web site and calling it *GroupChat.aspx*. This will hold a text box with messages as they accumulate, and it also gives users a means of adding messages.
4. When the messages are coming in, it would be very useful to know who sent which messages. This page forces users to identify themselves first; then, they can start adding messages. First, type in the text **Group Chatting...** after the *ScriptManager*. Give it a large font style with block display so that it's on its own line. After that, type in the text **First, give us your name:**. Then, drag a *TextBox* control from the Toolbox onto the page. Give the *TextBox* the ID *TextBoxUserID*. Drop a *Button* on the page so that the user can submit his or her name. Give it the text *Submit ID* and the ID *ButtonSubmitID*.
5. Drop another *TextBox* onto the page. This one will hold the messages, so make it large (800 pixels wide by 150 pixels high should do the trick). Set the *TextBox*'s *TextMode* property to *MultiLine*, and set the *ReadOnly* property to *True*. Give the *TextBox* the ID *TextBoxConversation*.

6. Drop one more *TextBox* onto the page. This one will hold the user's current message. Give the *TextBox* the ID *TextBoxMessage*.
7. Add one more *Button* to the page. This one enables the user to submit the current message and should include the text *Add Your Message*. Be sure to give the button the ID value *ButtonAddYourMessage*. The following graphic shows a possible layout of these controls:



8. Open the code-beside file GroupChat.aspx.cs for editing. Add a method that retrieves the user's name from session state:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class GroupChat : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    protected string GetUserID()
    {
        string strUserID =
            (string) Session["UserID"];
        return strUserID;
    }
}

```

9. Add a method to update the UI so that users may type messages only after they've identified themselves. If the user has not been identified (that is, the session variable is not there), *disable* the chat conversation UI elements and *enable* the user identification UI elements. If the user has been identified, *enable* the chat conversation UI elements and *disable* the user identification UI elements:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class GroupChat : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    // other code goes here...
    void ManageUI()
    {
        if (GetUserID() == null)

        {
            // if this is the first request, then get the user's ID
            TextBoxMessage.Enabled = false;
            TextBoxConversation.Enabled = false;
            ButtonAddYourMessage.Enabled = false;

            ButtonSubmitID.Enabled = true;
            TextBoxUserID.Enabled = true;
        }
        else
        {
            // if this is the first request, then get the user's ID
            TextBoxMessage.Enabled = true;
            TextBoxConversation.Enabled = true;
            ButtonAddYourMessage.Enabled = true;

            ButtonSubmitID.Enabled = false;
            TextBoxUserID.Enabled = false;
        }
    }
}

```

10. Add a *Click* event handler for the *Button* that stores the user ID (*ButtonSubmitID*). The method should store the user's identity in session state and then call *ManageUI* to enable and disable the correct controls:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

```

```

using System.Web.UI;
using System.Web.UI.WebControls;

public partial class GroupChat : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    // other page code goes here...
    protected void ButtonSubmitID_Click(object sender, EventArgs e)
    {
        Session["UserID"] = TextBoxUserID.Text;
        ManageUI();
    }
}

```

- 11.** Add a method to the page for refreshing the conversation. The code should look up the message list in the application cache and build a string that shows the messages in reverse order (so the most recent is on top). Then, the method should set the conversation *TextBoxConversation's* *Text* property to the new string (that is, the *text* property of the *TextBox* showing the conversation):

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class GroupChat : System.Web.UI.Page
{
    // other page code goes here...
    void RefreshConversation()
    {
        List<string> messages = (List<string>)Cache["Messages"];
        if (messages != null)
        {
            string strConversation = "";

            int nMessages = messages.Count;

            for(int i = nMessages-1; i >=0; i--)
            {
                string s;

                s = messages[i];
                strConversation += s;
                strConversation += "\r\n";
            }

            TextBoxConversation.Text =
                strConversation;
        }
    }
}

```

12. Add a *Click* event handler for adding your message by double-clicking the *Button* for adding your message (the lower button on the form) and adding a *Click* event handler to respond to the user submitting his or her message (*ButtonAddYourMessage*). The method should grab the text from the user's message *TextBoxMessage*, prepend the user's ID to it, and add it to the list of messages held in the application cache. Then, the method should call *RefreshConversation* to make sure the new message appears in the conversation *TextBox*:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class GroupChat : System.Web.UI.Page
{
    // Other code goes here...
    protected void ButtonAddYourMessage_Click(object sender,
                                                EventArgs e)
    {
        // Add the message to the conversation...
        if (this.TextBoxMessage.Text.Length > 0)
        {
            List<string> messages = (List<string>)Cache["Messages"];
            if (messages != null)
            {
                TextBoxConversation.Text = "";
                string strUserID = GetUserID();

                if (strUserID != null)
                {
                    messages.Add(strUserID +
                        ": " +
                        TextBoxMessage.Text);
                    RefreshConversation();
                    TextBoxMessage.Text = "";
                }
            }
        }
    }
}

```

- 13.** Update the *Page_Load* method to call *ManageUI* and *RefreshConversation*:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

using System.Xml.Linq;
using System.Collections.Generic;

public partial class GroupChat : System.Web.UI.Page
{
    // Other code goes here...
    protected void Page_Load(object sender, EventArgs e)
    {
        ManageUI();
        RefreshConversation();
    }
}

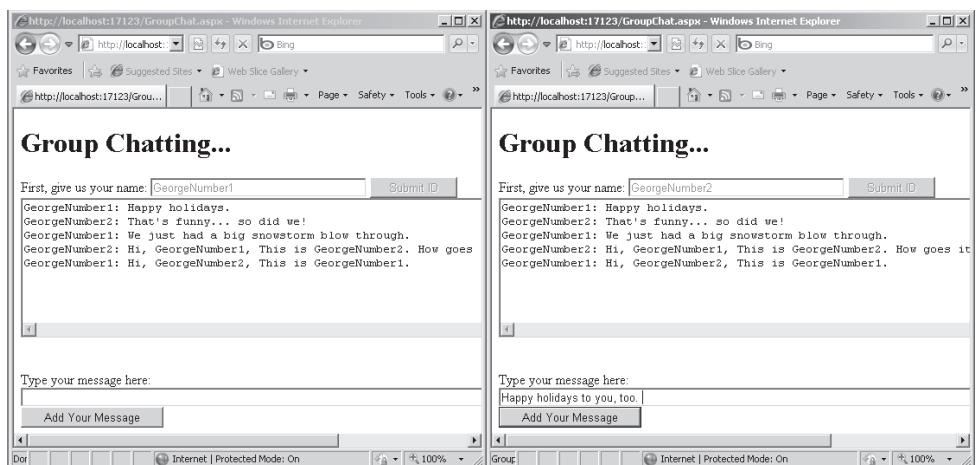
```

- 14.** Now run the page to see how it works. After you've identified yourself, you can start typing in messages—and you'll see them appear in the conversation *TextBox*. Try browsing the page using two separate browsers. Do you see an issue? The user typing a message gets to see the message appear in the conversation right away. However, other users involved in the chat don't see any new messages until after they submit messages of their own. You can solve this issue by dropping an *AJAX Timer* onto the page.
- 15.** Drag a *ScriptManager* from the *AJAX controls* onto the page. Then, drag a *Timer* from the *AJAX controls* onto the page. Although the *AJAX Timer* starts generating postbacks automatically, the default interval is 60,000 milliseconds, or once per minute. Set the *Timer's Interval* property to something more reasonable, such as 10,000 milliseconds (or 10 seconds).

Now run both pages and see what happens. You should see the pages posting back automatically every 10 seconds. However, there's still one more issue with this scenario. If you watch carefully enough, you'll see that the whole page is refreshed—even though the user name is not changing. During the conversation, you're really only interested in seeing the conversation *TextBox* updated. You can fix this by putting in an *UpdatePanel*.

- 16.** Drag an *UpdatePanel* from the *AJAX controls* onto the page. Position the *UpdatePanel* so that it can hold the conversation text box. Move the conversation text box so that it's positioned in the *UpdatePanel*. Modify the *UpdatePanel's* triggers so that it includes the *Timer's Tick* event. Now run the chat pages, and you should see only the

conversation text box being updated on each timer tick. The following graphic shows the new layout of the page employing the *UpdatePanel*:



The ASP.NET AJAX *Timer* is useful whenever you need regular, periodic posts back to the server. You can see here how it is especially useful when combined with the *UpdatePanel* to do periodic partial-page updates.

Updating Progress

A recurring theme when programming any UI environment is keeping the user updated about the progress of a long-running operation. If you're programming Windows Forms, you can use the *BackgroundWorker* component and show progress updating using the *Progress* control. Programming for the Web requires a slightly different strategy. ASP.NET AJAX support includes a component for this—the ASP.NET AJAX *UpdateProgress* control.

UpdateProgress controls display during asynchronous postbacks. All *UpdateProgress* controls on the page become visible when any *UpdatePanel* control triggers an asynchronous postback.

Here's an exercise for using an *UpdateProgress* control on a page.

Using the *UpdateProgress* control

1. Add a new page to the AJAXORama site named *UseUpdateProgressControl.aspx*.
2. Drag a *ScriptManager* from the Toolbox onto the page.

3. Drag an *UpdatePanel* onto the page. Give the panel the ID *UpdatePanelForProgress* so that you can identify it later. Add the text **This is from the update panel**, and then add a *Button* to the update panel that will begin a long-running operation. Give it the ID *ButtonLongOperation* and the text *Activate Long Operation*.
4. Add a *Click* event handler for the button. The easiest way to create a long-running operation is to put the thread to sleep for a few seconds, as shown here. By introducing a long-running operation, you have a way to test the *UpdateProgress* control and see how it works when the request takes a long time to complete.

```
public partial class UseUpdateProgressControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

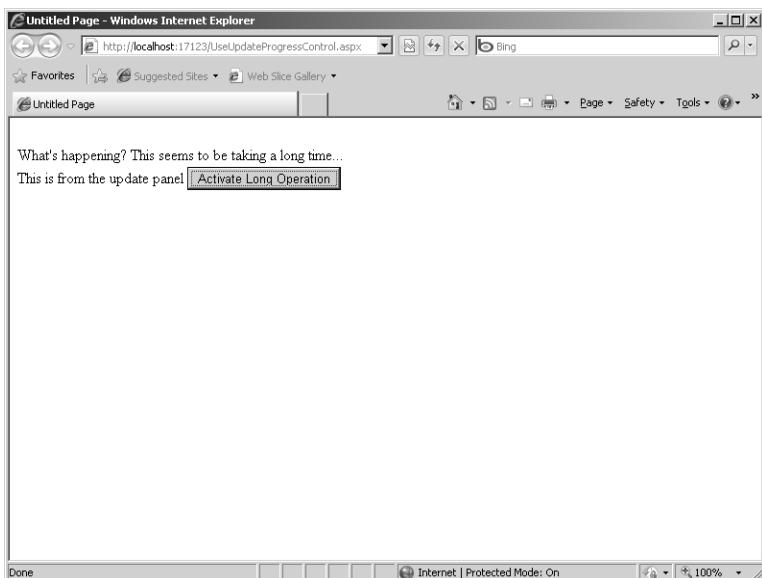
    }

    protected void
        ButtonLongOperation_Click(object sender,
                                  EventArgs e)
    {
        // Put thread to sleep for five seconds
        System.Threading.Thread.Sleep(5000);
    }
}
```

5. Now add an *UpdateProgress* control to the page. An *UpdateProgress* control must be tied to a specific *UpdatePanel*. Set the *UpdateProgress* control's *AssociatedUpdatePanelID* property to the *UpdatePanelForProgress* panel you just added. Note that you can simply use the provided dropdown to select this ID. Also change the *DisplayAfter* value to be 100 (indicating the progress indication should begin 100 milliseconds after the refresh begins).
6. Add a *ProgressTemplate* to the *UpdateProgress* control—this is where the content for the update display is declared. Add a *Label* to the *ProgressTemplate* so that you can see it when it appears on the page:

```
<asp:UpdateProgress ID="UpdateProgress1"
    runat="server"
    AssociatedUpdatePanelID="UpdatePanelForProgress"
    DisplayAfter="100">
    <ProgressTemplate>
        <asp:Label ID="Label1" runat="server"
            Text="What's happening? This takes a long time...">
        </asp:Label>
    </ProgressTemplate>
</asp:UpdateProgress>
```

7. Run the page to see what happens. When you click the button that executes the long-running operation, you should see the *UpdateProgress* control show its content automatically. This graphic shows the *UpdateProgress* control in action:



8. Finally, no asynchronous progress updating UI technology is complete without a means to cancel the long-running operation. If you wish to cancel the long-running operation, you can do so by inserting a little of your own JavaScript into the page. You need to do this manually because there's no support for this using the wizards. Write a client-side script block and place it near the top of the page—inside the `<head>` tag. The script block should get the instance of the `Sys.WebForms.PageRequestManager`. The `PageRequestManager` class is available to the client as part of the script injected by the ASP.NET AJAX server-side controls. The `PageRequestManager` has a method named `get_isInAsyncPostBack()` that you can use to figure out whether the page is in the middle of an asynchronous callback (generated by the *UpdatePanel*). If the page is in the middle of an asynchronous callback, use the `PageRequestManager's` `abortPostBack()` method to quit the request. Add a *Button* to the *ProgressTemplate* and assign its `OnClientClick` property to make a call to your new `abortAsyncPostBack` method. In addition to setting the `OnClientClick` property to the new abort method, insert `return false;` immediately after the call to the abort method, as shown in the following code. (Inserting `return false;` prevents the browser from issuing a postback.)

```
<%@ Page Language="C#"
  AutoEventWireup="true"
  CodeFile="UseUpdateProgressControl.aspx.cs"
  Inherits="UseUpdateProgressControl" %>
```

```

<!DOCTYPE html PUBLIC
  ...
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>

  <script type="text/javascript">
    function abortAsyncPostback()
    {
      var obj =
        Sys.WebForms.PageRequestManager.getInstance();
      if(obj.get_isInAsyncPostBack())
      {
        obj.abortPostBack();
      }
    }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <div>

      <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>

      </div>
      <asp:UpdateProgress ID="UpdateProgress1"
        runat="server"
        AssociatedUpdatePanelID="UpdatePanelForProgress"
        DisplayAfter="100">
        <ProgressTemplate>
          <asp:Label ID="Label1" runat="server"
            Text="What's happening? This takes a long time...">
          </asp:Label>
          <asp:Button ID="Cancel" runat="server"
            OnClientClick="abortAsyncPostback(); return false;">
            Text="Cancel" />
        </ProgressTemplate>
      </asp:UpdateProgress>
      <asp:UpdatePanel ID="UpdatePanelForProgress" runat="server">
        <ContentTemplate>
          This is from the update panel
          <asp:Button ID="ButtonLongOperation"
            runat="server"
            onclick="ButtonLongOperation_Click"
            Text="Activate Long Operation" />
        </ContentTemplate>
      </asp:UpdatePanel>

    </form>
  </body>
</html>

```



Caution *Caveat Cancel:* As you can see, canceling an asynchronous postback is completely a client-side affair. Canceling a long-running operation on the client end is tantamount to disconnecting the client from the server. Once the client is disconnected from the server, the client will never see the response from the server.

Also, although the client is happy that it could cancel the operation, the server might *never know* that the client canceled. So, the big caveat here is to plan for such a cancellation by making sure you program long-running blocking operations carefully so that they don't spin out of control. Although Microsoft Internet Information Services (IIS) 6 and IIS 7 should eventually refresh the application pool for such runaway threads, it's better to depend on your own good programming practices to make sure long-running operations end reasonably nicely.

ASP.NET AJAX support provides a great infrastructure for managing partial-page updates and for setting up other events such as regular timer ticks. The next section looks at the ASP.NET AJAX extender controls.

Extender Controls

The *UpdatePanel* provides a way to update only a portion of the page. That's pretty amazing. However, AJAX's compelling features have a very broad reach. One of the most useful features is the extender control architecture.

Extender controls target existing controls to extend functionality in the target. Whereas controls such as the *ScriptManager* and the *Timer* do a lot in terms of injecting script code into the page as the page is rendered, the extender controls often manage the markup (HTML) in the resulting page.

The following subsections discuss the ASP.NET AJAX extender controls. The first one is the *AutoComplete* extender.

The *AutoComplete* Extender

The *AutoComplete* extender attaches to a standard ASP.NET *TextBox*. As the end user types text in the *TextBox*, the *AutoComplete* extender calls a Web service to look up candidate entries based on the results of the Web service call. The following example borrows a component from Chapter 15, "Application Data Caching"—the quotes collection containing a number of famous quotes by various people.

Using the *AutoComplete* extender

1. Add a new page to AJAXORama. Because this page will host the *AutoComplete* extender, name it *UseAutocompleteExtender*.

2. Add an instance of the *ScriptManager* control to the page you just added.
3. Borrow the *QuotesCollection* class from Chapter 15. Remember, the class derives from *System.Data.Table* and holds a collection of famous quotes and their originators. You can add the component to AJAXORama by right-clicking the project node, selecting Add Existing Item, and locating the *QuotesCollection.cs* file associated with the *UseDataCaching* example in Chapter 15.
4. Add a method to retrieve the quotes based on the last name. The method should accept the last name of the originator as a string parameter. The *System.Data.DataView* class you use for retrieving a specific quote is useful for performing queries on a table in memory. The method should return the quotes as a list of strings. There might be none, one, or many, depending on the selected quote author. You use this function shortly.

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Collections.Generic;

///<summary>
/// Summary description for QuotesCollection
///</summary>
public class QuotesCollection : DataTable
{
    public QuotesCollection()
    { }

    public void Synthesize()
    {
        this.TableName = "Quotations";
        DataRow dr;

        Columns.Add(new DataColumn("Quote", typeof(string)));
        Columns.Add(new DataColumn("OriginatorLastName", typeof(string)));
        Columns.Add(new DataColumn(@"OriginatorFirstName",
            typeof(string)));

        dr = this.NewRow();
        dr[0] = "Imagination is more important than knowledge.";
        dr[1] = "Einstein";

        dr[2] = "Albert";
        Rows.Add(dr);
    }
}
```

```
// Other quotes added here...
}

public string[]
GetQuotesByLastName(string strLastName)
{
    List<string> list = new List<string>();

    DataView dvQuotes = new DataView(this);
    string strFilter = String.Format("OriginatorLastName = '{0}'", strLastName);
    dvQuotes.RowFilter = strFilter;

    foreach (DataRowView drv in dvQuotes)
    {
        string strQuote =
            drv["Quote"].ToString();

        list.Add(strQuote);
    }

    return list.ToArray();
}
}
```

5. Add a class named *QuotesManager* to the project. The class manages caching.

The caching example from which this code is borrowed stores and retrieves the *QuotesCollection* during the *Page_Load* event. Because the *QuotesCollection* will be used within a Web service, the caching has to happen elsewhere. To do this, add a public static method named *GetQuotesFromCache* to retrieve the *QuotesCollection* from the cache:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

/// <summary>
/// Summary description for QuotesManager
/// </summary>
public class QuotesManager
{
    public QuotesManager()
    {

    }

    public static QuotesCollection GetQuotesFromCache()
    {
        QuotesCollection quotes;
```

```

        quotes =
            (QuotesCollection)HttpContext.Current.Cache["quotes"];

        if (quotes == null)
        {
            quotes = new QuotesCollection();
            quotes.Synthesize();
        }
        return quotes;
    }
}

```

6. Add an XML Web Service to your application. Right-click the project and add an ASMX file to your application. Name the service *QuoteService*. You can remove the *WebService* and *WebServiceBinding* attributes, but be sure to adorn the XML Web Service class with the *[System.Web.Script.Services.ScriptService]* attribute by uncommenting it (Visual Studio put it in for you). That way, it is available to the *AutoComplete* extender later on. The *AutoCompleteExtender* uses the XML Web Service to populate its drop-down list box.
7. Add a method to get the last names of the quote originators—that's the method that populates the drop-down box. The method should take a string representing the text already typed in as the first parameter, an integer representing the maximum number of strings to return. Grab the *QuotesCollection* from the cache using the *QuoteManager*'s static method *GetQuotesFromCache*. Use the *QuotesCollection* to get the rows from the *QuotesCollection*. Finally, iterate through the rows and add the originator's last name to the list of strings to be returned if it starts with the prefix passed in as the parameter. The Common Language Runtime (CLR) *String* type includes a method named *StartsWith* that's useful to figure out whether a string starts with a certain prefix. Note that you also have to add *using* statements for generic collections and data as shown:

```

using System;
using System.Linq;
using System.Web;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

using System.Data;

[System.Web.Script.Services.ScriptService]
public class QuoteService : System.Web.Services.WebService
{

```

```

[WebMethod]
public string[]
GetQuoteOriginatorLastNames(string prefixText,
                            int count)

{
    List<string> list = new List<string>();

    QuotesCollection quotes =
        QuotesManager.GetQuotesFromCache();

    prefixText = prefixText.ToLower();

    foreach (DataRow dr in quotes.Rows)
    {
        string strName =
            dr["OriginatorLastName"].ToString();

        if (strName.ToLower().StartsWith(prefixText))
        {
            if (!list.Contains(strName))
            {
                list.Add(strName);
            }
        }
    }

    return list.GetRange(0,
                        System.Math.Min(count, list.Count)).ToArray();
}
}

```

8. Now drop a *TextBox* on the *UseAutocompleteExtender* page to hold the originator's last name to be looked up. Give the *TextBox* an ID of *TextBoxOriginatorLastName*.
9. Drag an *AutoCompleteExtender* from the *AJAX Toolbox* and add it to the page. Set its ID to be *AutoCompleteExtenderForOriginatorLastName*. Point the *AutoComplete TargetControlID* to the *TextBox* holding the originator's last name, *TextBoxOriginatorLastName*. Make the *MinimumPrefix* length 1, the *ServiceMethod* *GetQuoteOriginatorLastNames*, and the *ServicePath* *quoteservice.asmx*. This wires up the *AutoComplete* extender so that it takes text from the *TextBoxOriginatorLastName TextBox* and uses it to feed the XML Web Service *GetQuoteOriginatorLastNames* method.

```

<cc1:AutoCompleteExtender
    ID="AutoCompleteExtenderForOriginatorLastName"
    TargetControlID="TextBoxOriginatorLastName"
    MinimumPrefixLength="1"
    ServiceMethod="GetQuoteOriginatorLastNames"
    ServicePath="quoteservice.asmx"
    runat="server">
</cc1:AutoCompleteExtender>

```

10. Add a *TextBox* to the page to hold the quotes. Name the *TextBox* *TextBoxQuotes*.
11. Update the *Page_Load* method. It should look up the quotes based on the name in the text box by retrieving the *QuotesCollection* and calling the *QuotesCollection GetQuotesByLastName* method:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Text;

public partial class UseAutocompleteExtender :
System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        QuotesCollection quotes =
            QuotesManager.GetQuotesFromCache();
        string[] quotesArray =
            quotes.GetQuotesByLastName(TextBoxOriginatorLastName.Text);

        if (quotesArray != null && quotesArray.Length > 0)
        {
            StringBuilder str = new StringBuilder();
            foreach (string s in quotesArray)
            {
                str.AppendFormat("{0}\r\n", s);
            }
            this.TextBoxQuotes.Text = str.ToString();
        }
        else
        {
            this.TextBoxQuotes.Text = "No quotes match your request.";
        }
    }
}

```

12. To make the page updates more efficient, drop an *UpdatePanel* onto the page. Put the *TextBox* for holding the quotes in the *UpdatePanel*. This causes only the *TextBox* showing the quotes to be updated instead of performing a whole-page refresh. Add a button following the originator's last name *TextBox* with the ID *ButtonFindQuotes*.
13. Add two *asynchPostBack* triggers to the *UpdatePanel*. The first trigger should connect the *TextBoxOriginatorLastName TextBox* to the *TextChanged* event. The second trigger should connect the *ButtonFindQuotes* button to the button's *Click* event.

The following graphic shows the layout of the page using the *AutoCompleteExtender* in action:



14. Run the page. As you type originator names into the *TextBox*, you should see a drop-down list appear containing candidate names based on the *QuotesCollection*'s contents.

The *AutoComplete* extender is an excellent example of the capabilities that ASP.NET AJAX support includes. Internet Explorer has had an autocomplete feature built in for quite a while. Internet Explorer remembers often-used names of HTML input text tags and recent values that have been used for them. For example, when you go online to buy an airline ticket and then go back to buy another one later, watch what happens as you type in the Web address. The Internet Explorer autocomplete feature makes available a drop-down list below the address bar that shows the last few addresses you've typed in that begin with the same text you began typing in the text box.

The ASP.NET *AutoComplete* extender works very much like this. However, the major difference is that the end user sees input candidates generated by the Web site rather than simply a history of recent entries. Of course, the Web site could mimic this functionality by tracking a user's profile identity and store a history of what a particular user has typed in to a specific input field on a page. The actual process of generating autocomplete candidates is completely up to the Web server, giving a whole new level of power and flexibility in programming user-friendly Web sites.

A Modal Pop-up Dialog-Style Component

AJAX provides another interesting feature that makes Web applications appear more like desktop applications: the *ModalPopup* extender. Historically, navigating a Web site involves users walking down the hierarchy of a Web site and climbing back out. When users provide inputs as they work with a page, the only means available to give feedback about the quality of the input data has been the validation controls. In addition, standard Web pages have no facility to focus users' attention while they type in information.

Traditional desktop applications usually employ modal dialog boxes to focus user attention when gathering important information from the end user. The model is very simple and elegant: The end user is presented with a situation in which he or she must enter some data and then click OK or Cancel before moving on. After dismissing the dialog box, the end user sees exactly the same screen he or she saw right before the dialog box appeared. There's no ambiguity and no involved process where the end user must walk up and down some arbitrary page hierarchy.

This example shows how to use the pop-up dialog extender control. You create a page with some standard content and then have a modal dialog-style pop-up window appear right before the page is submitted.

Using a *ModalPopup* extender

1. Add a new page to AJAXORama to host the pop-up extender. Call it *UseModalPopupExtender*.
2. As with all the other examples using AJAX controls, drag a *ScriptManager* from the Toolbox onto the page.
3. Add a title to the page (the example here uses "ASP.NET Code of Content"). Give the banner some prominence by surrounding it with `<h1>` and `</h1>` tags. You can simply replace the existing `<div>` tag with the `<h1>` tag.
4. Drag a *Panel* from the Toolbox onto the page to hold the page's normal content.
5. Add a *Button* to the *Panel* for submitting the content. Give the *Button* the ID *ButtonSubmit* and the text *Submit* and create a button *Click* event handler. You need this button later.
6. Place some content on the panel. The content in this sample application uses several check boxes that the modal dialog pop-up examines before the page is submitted.

```
<h1>ASP.NET Code Of Conduct </h1>

<asp:Panel ID="Panel1" runat="server"
    style="z-index: 1;left: 10px;top: 70px;
    position: absolute;height: 213px;width: 724px;
    margin-bottom: 0px;">
```

```

<asp:Label ID="Label1" runat="server"
    Text="Name of Developer;"></asp:Label>
    &nbsp;<asp:TextBox ID="TextBox1"
    runat="server"></asp:TextBox>

<br />
<br />
<br />
As an ASP.NET developer, I promise to
<br />
<input type="checkbox" name="Check" id="Checkbox1"/>
<label for="Check1">Use Forms Authentication</label>
<br />
<input type="checkbox" name="Check" id="Checkbox2"/>
<label for="Check2">Separate UI From Code</label>
<br />
<input type="checkbox" name="Check" id="Checkbox3"/>
<label for="Check3">Take Advantage of Custom Controls</label>
<br />
<input type="checkbox" name="Check" id="Checkbox4"/>
<label for="Check4">Use AJAX</label>
<br />
<input type="checkbox" name="Check" id="Checkbox5"/>
<label for="Check5">Give MVC a try</label>
<br />
<input type="checkbox" name="Check" id="Checkbox6"/>
<label for="Check6">Give Silverlight a try</label>
<br />

<asp:Button ID="ButtonSubmit" runat="server" Text="Submit"
    onclick="ButtonSubmit_Click" />
<br />
</asp:Panel>

```

7. Add another *Panel* to the page to represent the pop-up. Give this *Panel* a light yellow background color so that you'll be able to see it when it comes up. It should also have the ID *PanelModalPopup*.
8. Add some content to the new *Panel* that's going to serve as the modal pop-up. At the very least, the pop-up should have *OK* and *Cancel* buttons. Give the *OK* and *Cancel* buttons the ID values *ButtonOK* and *ButtonCancel*. You need them a bit later, too.

```

<asp:Panel ID="PanelModalPopup" runat="server"
    BorderColor="Black"
    BorderStyle="Solid"
    BackColor="LightYellow" Height="72px"
    Width="403px">
<br />
<asp:Label
    Text="Are you sure these are the correct entries?"
    runat="server">
</asp:Label>
    &nbsp;&nbsp;&nbsp;

```

```

<asp:Button ID="ButtonOK"
    runat="server"
    Text="OK" />
    &nbsp;&nbsp;
<asp:Button ID="ButtonCancel"
    runat="server" Text="Cancel" />
    <br />
</asp:Panel>

```

9. Add a script block to the ASPX file. You need to do this by hand. Write functions to handle the *OK* and *Cancel* buttons. The example here examines check boxes to see which ones have been selected and then displays an alert to show which features have been chosen. The *Cancel* handler simply displays an alert indicating that the *Cancel* button was clicked:

```

<script type="text/javascript">

    function onOk() {
        var optionsChosen;
        optionsChosen = "Options chosen: ";

        if($get('Checkbox1').checked)
        {
            optionsChosen =
                optionsChosen.toString() +
                "Use Forms Authentication ";
        }

        if($get('Checkbox2').checked)
        {
            optionsChosen =
                optionsChosen.toString() +
                "Separate UI From Code ";
        }

        if($get('Checkbox3').checked)
        {
            optionsChosen =
                optionsChosen.toString() +
                "Take Advantage of Custom Controls ";
        }

        if($get('Checkbox4').checked)
        {
            optionsChosen =
                optionsChosen.toString() +
                "Give AJAX a try ";
        }
        alert(optionsChosen);
    }

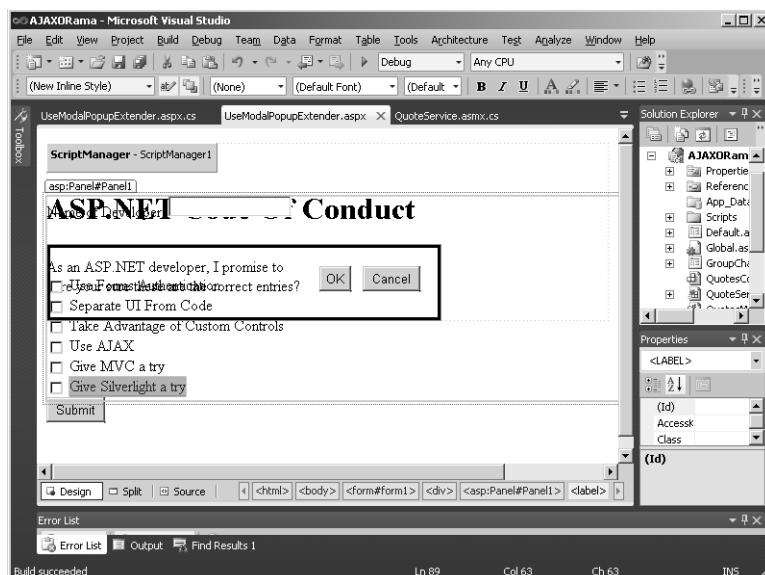
    function onCancel() {
        alert("Cancel was pressed");
    }
</script>

```

10. Drag the *ModalPopup* extender from the Toolbox onto the page.
11. Add the following markup to the page to set various properties on the new *ModalPopup* extenders. This sets the *OkControlID* property to *ButtonOK* and the *CancelControlID* property to *ButtonCancel*. It also sets the *OnCancelScript* property to *onCancel()* (the client-side Cancel script handler you just wrote). Set *OnOkScript="onOk()"* (the client-side OK script handler you just wrote). Finally, the following markup sets the *TargetControlID* property to *ButtonSubmit*:

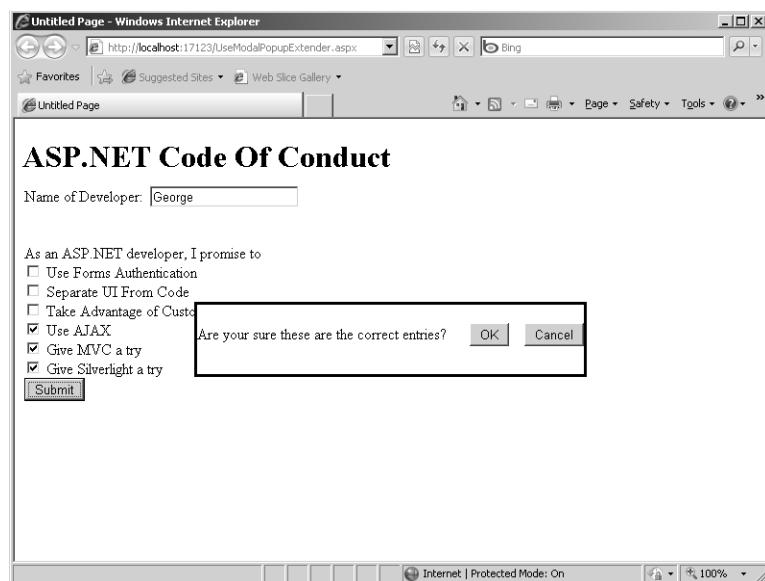
```
<cc1:ModalPopupExtender
    ID="ModalPopupExtender1"
    runat="server"
    OkControlID="ButtonOK"
    CancelControlID="ButtonCancel"
    OnCancelScript="onCancel()"
    OnOkScript="onOk()"
    TargetControlID="ButtonSubmit"
    PopupControlID="PanelModalPopup"
    runat="server"
    DynamicServicePath="" Enabled="True">
</cc1:ModalPopupExtender>
```

This graphic shows the layout of the page using the *ModalPopup* extender in Visual Studio 2010:



12. Run the page. When you click the Submit button, the *Panel* designated to be the modal pop-up window is activated. (Remember, the Submit button is the *TargetControlID* of the *ModalPopup* Extender.) When you dismiss the pop-up window by clicking OK or

Cancel, you should see the client-side scripts being executed. The following graphic image shows the *ModalPopup* extender displaying the modal pop-up window:



Chapter 23 Quick Reference

To	Do This
Enable a Web site for AJAX	Normal Web sites generated by Visual Studio 2010's template are AJAX-enabled by default. However, you must add a <i>ScriptManager</i> to a page before using any of the AJAX server-side controls.
Implement partial page updating in your page	From within an ASP.NET project, select an <i>UpdatePanel</i> from the toolbox. Controls that you place in the <i>UpdatePanel</i> will trigger updates for only that panel, leaving the rest of the page untouched.
Assign arbitrary triggers to an <i>UpdatePanel</i> (that is, trigger partial page updates using controls and events not related to the panel)	Modify an <i>UpdatePanel</i> 's trigger collection to include the new events and controls. Highlight the <i>UpdatePanel</i> from within the Visual Studio designer. Select the <i>Triggers</i> property from within the property editor. Assign triggers as appropriate.
Implement regularly timed automatic posts from your page	Use the AJAX <i>Timer</i> control, which will cause a postback to the server at regular intervals.
Use AJAX to apply special UI nuances to your Web page	After installing Visual Studio 2008, you can create AJAX-enabled sites, and use the new AJAX-specific server-side controls available in the AJAX toolkit. Select the control you need. Most AJAX server-side controls may be programmed completely from the server. However, some controls require a bit of JavaScript on the client end.

Index

Symbols

404 errors, 378
<% and %> tags, 19, 31
<body> tag, 147
<Canvas>/</Canvas> tags, 448
[DataContract] attribute, 573
<deny users="*"/> node, 204
<form>/</form> tags, 9–11
 action attribute, 10
 method attribute, 10–11
<Grid>/</Grid> tags, 448
<iframe> element, 442, 445
 tag, 128
<input type=image /> tag, 129
<object> tags, 524–525
[OperationContract] attribute, 573
<option> tag, 207
<Page>/</Page> tags, 448
[ScriptableMember] attribute, 534
[ScriptableType] attribute, 534
<select>and </select> tags, 9, 207
[ServiceContract] attribute, 573

A

ABC endpoints definition, 557
abortPostBack() method, 499
absolute expirations, for cached data, 331–333
absolute positioning, 77, 150
AcceptVerbs attribute, 469, 471
access
 managing, 181. *See also* security
 speeds of, 321
access rules, 198
 creating, 203–204
Accordion extenders, 482
AccountController, 457
action attribute, 10
ActionResult, 460, 469
Active Data Objects (ADO), 215
Active Server Pages (ASP), 18–21
 code processing, 46
 control state, loss of, 97
 dynamic content support, 60, 61
 execution model, 25
 locked files in, 42

Response object, 32
script blocks, 31
ActiveViewIndex property, 137
ActiveX controls, for Web-based GUIs, 62
Add Application Setting dialog box, 178
add attribute, 406
Add Connection String dialog box, 177
Add New Access Rule link, 203
Add New Item dialog box, 53
Add Reference dialog box, 398
Add Service Reference command, 547
Add Service Reference dialog box, 568
Add Style Rule dialog box, 156–157
administrators, user access control, 182
ADO (Active Data Objects), 215
ADO.NET, 215–221
 database connection classes, 216
 database provider factories, 216–217
 database scalability and, 219
 result set management, 218–221
ADO.NET objects, data-bound controls, session state and, 299–305
AJAX (Asynchronous Java and XML), 433, 474–475
AJAX-style programming examples, 477
ASP.NET and, 475–478
async callbacks, 489–490
authentication support, 477
AutoComplete extender, 433, 501–507
base class library, 480
benefits of, 476–477
browser compatibility layer, 480
browser support, 477
client-side support, 480–484
core services layer, 480
 vs. DHTML, 478
extender control architecture, 477, 501–512
ModalPopup extender, 433, 508–512
networking layer, 480
partial-page updates, 477, 484–489
personalization support, 477
progress updating, 497–501
in the real world, 477–478
RIAs, creating with, 473
server-side support, 478–480
style of programming, 474–475
Web service idiom use, 475
Web sites, enabling for, 512
 for Web UIs, 62
AJAX Control Toolkit, 475, 480–481
building, 481
community-supported effort, 481
controls and extenders, 482–484
AJAX Library scripts, registering with page, 479
AJAX script libraries, 480
Alexander, Christopher, 451
allowAnonymous attribute, 262, 266
allowCustomSqlDatabase setting, 311
AlternateRowStyle property, 228
AlternateText property, 130
AlwaysVisibleControl extenders, 482
Animation extenders, 482
animations
 rendering, 448
 in Silverlight, 535–542
Anonymous Authentication mode, 183
anonymousIdentification element, 262, 266
anonymous personalization, 262
anonymous profiles, 261
 tracking, 266
AnonymousTemplate template, 200

anonymous users
 denying access of, 198–199
 personalization support for, 262

App_Code directory, adding to projects, 210

App_Data directory, 53

AppDomain state, shared, 560

AppearanceEditorPart, 271, 276

appearance of pages, output caching and, 357

AppendCacheExtension, 351

Application_AuthenticateRequest event, 391

Application_BeginRequest event, 391

Application_BeginRequest handler, 393

application configuration settings accessing, 173–174
 adding to *web.config*, 170–173
 managing, 177–178

application data cache, 389–390
 vs. application state, 389

application dictionary, 386
 accessing, 404
 indexing data in, 389
 storing data in, 389

Application directive, 387

Application_End event, 390

Application_EndRequest handler, 393

Application_Error event, 390

application event handlers, 387–388

application object, 386
 event handling abilities, 390–391

Application objects, 389
 in Silverlight projects, 521

application performance. *See also* performance
 nesting controls and, 102
 view state management and, 100

application pooling, 31

applications. *See also* Web applications
 access management, 181. *See also* security
 cache, 80
 console application, 6, 567
 debugging, 375–377, 383
 Debug version, 580
 distributable, 557
 interactive applications, 10–11

isolation of, 31

Release version, 580

RIAs, 473–474

settings in *web.config*, 581

Silverlight applications, 517–524, 533–534, 551

virtual directories for, 13

WPF applications, 435

XBAPs, 438–441, 448, 513

Application_Start event, 388, 389, 390

application state, 385, 389–390
 accessing, 49
 vs. application data cache, 389
 managing, 388–389
 MVC model management of, 450
 storing in modules, 400–403

application tracing, 370–374
 enabling, 325, 383

application-wide cache, 49

application-wide data repository for, 48. *See also* *HttpApplication* objects

application-wide events, 385, 386, 391–392
 exposing, 47
 handling, 49, 387, 390–391

App_Themes directory
 creating, 156

App.xaml.cs files, 439

App.xaml files, 439

ArrayList objects, adding to controls, 97–98

ASCX files, applying *OutputCache* directive, 354, 359

ASHX files, 417–419

ASP. *See* Active Server Pages (ASP)

ASPClassic handler, 18

ASP.DLL ISAPI DLL, 19

.asp extension, 19

ASP.NET
 AJAX and, 475–480
 browser capability information, 95
 evolution of, 22–23
 IIS compatibility, 35
 object-oriented execution model, 25
Page model, 64–66. *See also* pages
 precompiling, 577–578
 request handling facility, 407–410

server-side control architecture, 59, 62

subdirectories in, 41

syntax, 25

technologies underlying, 3

Visual Studio and, 50–58

WCF and, 560–561

XBAP support, 438

ASP.NET 1.0, 22
 custom control/user control approach to UI, 143–144

ASP.NET 1.x, configuration management, 163, 168–169

ASP.NET 2.0, 22–23
 configuration management, 169

ASP.NET 3.5, 23

ASP.NET 4, 23

ASP.NET applications. *See also* applications; Web applications
 converting HTML files to, 29
aspnet_compiler utility, 578, 585

ASP.NET configuration, 163, 167–174. *See also* application configuration settings; configuration
 configuring from IIS, 174–180
 keywords, 166
 schema for, 167–168

ASP.NET Configuration Settings pane, 180

ASP.NET Configuration tab, 163, 174–180

aspnet_isapi.dll
 mapping extensions to, 419
 requests routed to, 38

ASP.NET MVC Site template, 456

ASP.NET pipeline, 46–49
 accessing, 40, 48–50
 HTTP modules and, 395–396
 integrated version, 47
 WCF services access to, 561

aspnet_regsql.exe, 195, 288, 336

ASP.NET Server Control template, 82, 100, 103

ASPNETStepByStepExamples, 52

ASPNETStepByStep Web site, 27

ASP.NET tags, *ID* attribute, 74

ASP.NET test sites, for Silverlight content, 517–518

ASP.NET Web Forms, controls collection in, 80

ASP.NET Web Site template, 53

ASP.NET worker process, 31

asp: prefix, 86

.aspx extension, 29
ASPX files
 code-behind directives, 43–44
 compiling, 41, 417
 integrating with assemblies, 30–31
 mapping virtual URLs to, 251–255
ASPX pages
 breakpoints, inserting, 375
 page tracing, setting, 364
assemblies. *See also .NET assemblies*
 binary composite controls as, 117
 integrating with .aspx files, 30–31
 pages compiled into, 41
 precompiled, 44–45, 577
 refreshing, 93
 reverse compiling, 41
 storage in temporary directories, 41, 42
 strong names for, 166
 viewing, 41–42
AssociatedUpdatePanelID property, 498
 asterisks, in output caching parameters, 357
AsyncFileUpload extender, 482
asynchPostBack triggers, 506
 asynchronous background processing, 474
Asynchronous Java and XML. *See* AJAX (Asynchronous Java and XML)
 asynchronous postbacks, 489–490
 canceling, 499–501
 asynchronous proxy methods, Visual Studio generated, 568
 asynchronous service references, 568
 attributes, of HTML tags, 9
Authenticate event, 200
Authenticate method, 190
 authentication, 119, 189–194
 AJAX support for, 477
 definition of, 181
 managing in configuration files, 168
 manual, 190, 206
 Passport authentication, 189
 WCF services side-by-side mode and, 560
 Windows authentication, 189

authentication cookies
 installing, 190
 invalidating, 206
 setting manually, 192
 value of, viewing, 188
 verifying, 206
authentication keyword, 166
Authentication page (ASP.NET Configuration Settings pane), 180
 authorization, 119, 182, 184, 203–206
 access rules, 198
 definition of, 181
 WCF services side-by-side mode and, 560
AutoComplete extenders, 433, 482, 501–507
AutoDetect, 310
AutoEventWireup attribute, 145
automaticSaveEnabled attribute, 260
AutoPostBack property, 212
.axd extension, file mappings for, 409

B

background processing, asynchronous, 474
BackgroundWorker component, 497
 backing files, 334
 banners on master pages, 153–154
 base classes, building pages based on, 144
 base class library, 480
BeginRequest event, intercepting, 392–393
BehaviorEditorPart, 272
 behaviors (WCF), 558–559
 binary composite controls, 101, 103–110
 advantages of, 117
 default properties, 113
 disadvantage of, 117
 global assembly cache, adding to, 117
 mappings to file extensions, 36
 UI functionality, grouping in, 117
 bindings (WCF endpoint), 558
 bitmaps. *See also* images
 clickable areas, defining, 130

BorderColor property, 134
BorderStyle property, 134
 breakpoints, inserting, 375
 breaks (
), 86
BrowseDisplayMode, 274
 browser compatibility layer, 480
 browser definition capability files, 165

Browser object, 95
browsers
 AJAX support, 477
 HTML interpreting capabilities, 95
 HTTP requests from, 5–6, 29
 jobs of, 7
 modern features of, 475
 pushing functionality to, 433–434
 XAML plug-in, 448
 variations in, 475
 viewing HTML source in, 30
 version-independent rendering code, 100
 built-in handlers, 407–410. *See also* HTTP handlers
BulletedList control, 209
BunchOfControls page, 59, 60
 control tree, 66
 raw HTML, 65–66
 source code for, 61, 63
Button_Click handlers, 440
 buttons
 adding graphics to, 129
 adding to page, 73
Click handlers for, 115
 event handlers for, 74–75

C

CacheControl property, 353
 cached data. *See also* data caching
 expirations, 331–333
 flushing, 334
 managing, 163, 346–354
 SQL Server dependency, 336–338
 varying on query string parameters, 348–351
 cache dependencies, 334–336. *See also* data cache
CacheDependency objects, 341, 353
Cache.Insert method, 341
Cache.NoAbsoluteExpiration policy, 335

Cache.NoSlidingExpiration policy, 332, 335
Cache objects
 accessing, 321
 Insert method, 327–328
CacheProfile attribute, 347
Cache property, 341, 359
CacheWithFileDependency method, 336, 340
caching. See also *data caching*; *output caching*
 page content, 343–346
 user controls, 359
caching profiles, 353–354
Calendar extenders, 482
callback delegates, 341
callbacks, 488. See also *postbacks*
 asynchronous, 489–490
 initiating, 338–340
Call Stack window, 377
Canvas, 435, 442–443, 528–529
 adding, 448
 adding content to, 448
Canvas.Left property, 444
Canvas.Right property, 448
Canvas.Top property, 448
CascadingDropDown extenders, 482
Cascading Style Sheets (CSS), 155
Cassini Web server, 12, 13
CatalogDisplayMode, 274
CatalogPart controls, 271
catalogs, adding Web Parts to, 284–285
CatalogZone, 277, 288
cc1: prefix, 86
ChangePassword control, 200
channels (WCF), 558
chat pages, creating, 491–497
CheckBoxList control, 209
child controls
 adding, 102
 of composite controls, 109
 unique IDs for, 103
Choose Toolbox Items dialog box, 83
ChtmlTextWriter class, 95
Class1 class, 396, 411
Class1.cs file, 396
classic ASP, 18–21. See also *Active Server Pages (ASP)*
 file inclusion method, 143
Classic mode (IIS), 37
ClassName attribute, 145
Click handlers, adding, 115
 client authentication by IIS, 36.
 See also *authentication*
ClientBin directory, 519
clients
 infrastructure on, 62
 JavaScript support, 125
 sending output to, 49
client-side controls, 59
client-side scripting, 433
client-side validation, 125
closing tags (HTML), 9
CLR object dictionary, 321. See also *data cache*
code behind, 43–44
code beside, 43–45
 accessing controls with, 64
 derivation of, 64
code blocks, server-side, 33–34
code execution, server-side, 31
CodeFile directive, 45
collaboration sites, Web Parts and, 268
CollapsiblePanel extenders, 482
collections
 attaching to controls, 208, 210–215
 binding to controls, 236
 member variables, exposing, 211–212
 rendering as tables, 98
 representing as UI elements, 207
 representing with data binding, 208–210
 representing without data binding, 207–208
ColumnDefinitions property, 448, 545
CommandBuilder, 221
commands, database commands, 217–218
Common Gateway Interface (CGI), 11–12
Common Language Runtime (CLR) classes, *IHttpHandler* implementation, 25
communications API, fragmented, 556
CompareValidator controls, 120
 adding to Web Forms, 127
compilation model, 35–42
CompilerOptions attribute, 145
compiler tracing, 374
Component Object Model (COM), ActiveX support, 62
CompositeControl class, deriving from, 103
composite controls, 79. See also *controls*
ArrayList, adding, 107–108
building, 103–110
Button event handler, 104
categories of, 101
child members, 102
control tree, 109
CreateChildControls method override, 105
event hookup, 105
fields, adding, 103
LiteralControl, 105
for login, 102
nesting, 102
RenderContents method, removing, 103
vs. rendered controls, 101–102
rendering, 102
running, 108
*StripNonAlphanumeric*s method, 103–104
Table control, adding, 107–108
Text property, 103
utility of, 102
Composite pattern, 452
CompositeType class, 563
config.sys, editing, 414
configuration. See also *application configuration settings*
 browser definition capability files, 165
configuration section handlers, 165
 keywords, 166
machine.config comments, 165
managing in *ASP.NET 1.x*, 168–169
managing in later versions of *ASP.NET*, 169–174
of site maps, 242–243
configuration files, 163
authentication, managing in, 168
 for deployment, 582
 listing, 164
 for .NET applications, 167
 retrieving settings from, 180
 for Web applicaitons, 167
ConfigurationManager.
AppSettings collection, 178
ConfigurationManager class, 173, 180

configuration section handlers, 165–166
 configuration settings
 for ASP.NET applications, 180
 retrieving, 180
 for subdirectories under virtual
 directories, 180
 using IIS ASP.NET Configuration
 tool, 180
 using Web Site Administration
 Tool, 180
ConfirmButton extenders, 482
ConnectDisplayMode, 274
 connecting to databases, 215–217
 connection strings, adding, 177
 Console Application projects, 6,
 567
 content
 dynamic content, 9–18
 interactive content, 520
 semidynamic content, 437
 static content, generalized, 257
ContentPlaceholder controls,
 145–147
Context.ClearError, 382
 contracts (WCF endpoints), 558
 Visual Studio-generated
 placeholders for, 562
Control class. *See System.Web.*
 UI.Control class
 control flow, with Forms
 Authentication, 185
Controller class, 453
 controllers, adding to MVC
 projects, 460–461, 464, 472
 controls, 77. *See also* composite
 controls; user controls
 adding to control tree, 64
 adding to Toolbox, 100
 adding with Visual Studio, 67–78
 appearance properties,
 managing, 159–160
 custom, 3, 81–88, 100, 101
 data-bound. *See also* data-
 bound controls
 declaring on page, 85
 default event handlers, 78
 event management for, 100
 extender, 477, 501–512
 identifying text, 85
 layout options, 77, 78
 naming, 121
 nesting, 102
 new instances of, 93
 placing on pages, 84
 populating with query results,
 221–226
 properties, changing, 78, 86,
 100
 rendering as tags, 59
 run-time availability, 75
 selecting, 85–86
 selection handlers, 213
 Silverlight controls, 516,
 526–527
 skins for, 159–160
 state of, 62
 tag prefix, mapping to, 85
 view state management, 93,
 97–100
ControlToAssociate property, 121
ControlToValidate property, 120
 setting, 122–123
 control trees
 construction of, 80
 control instances, adding, 64
 controls in, 87, 88
 viewing, 365
 walking, 67–68
cookieName setting, 311
 cookies
 session identifier as, 310
 session state, tracking with,
 309–310
 Set Auth cookie, 206
CookiesSupported method, 190
 core services layer, 480
 Create Application Settings link,
 170
CreateChildControls method,
 overriding, 105
Create Package command, 583
CreateUserWizard control, 201
CssClass property, 77
CurrentNode property, 240
Current property, 49–50, 387
 custom attributes, for nodes in
 web.sitemap, 248–250
 custom cache providers, 358
CustomControlLib assembly,
 composite controls in, 110
CustomControlLib directory, 83
CustomControlLib project, 82–88
 custom controls, 3. *See also*
 composite controls;
 controls
 creating, 81–88, 100
 types of, 101
 custom error pages, 378–381
customErrors section (web.config),
 379
 custom handlers. *See also* HTTP
 handlers
 creating, 419
 mapping extensions to,
 414–415
 as separate assemblies, 419
 session state and, 416
 Trace handler, 407–409
 writing, 411–415
 custom providers for session
 state, 311
CustomValidator controls, 120,
 127–128

D

data access, 324
 expense of, 323. *See also* data
 caching
DataAdapter, 219–220
 database access, expense of, 323
 database lookups, 324
 Database Markup Language
 (DBML) source files, 458
 database queries
 configuring, 222–223
 inline, 234–236
 using a *DataSet*, 220
 using data readers, 218
 database query results
 IDataReader, for iterating
 through, 218–219
 populating controls with,
 221–226
 databases
 commanding, 217–218
 connecting to, 215–217, 222
 deployment/redeployment, 579
 limits of connectivity, 219
 managing result sets, 218–221
 provider pattern and, 216
 random access to, 221
 SQL support, 217–218
 Tables collection, 219
 database tables, wrapper classes
 for, 472
 database technologies, ADO.NET
 providers for, 216
DataBind, 208
 data binding
 collections, representing with,
 208–210

data binding (*continued*)
 simple method, 210–215
TreeView support of, 135
 data-bound controls, 208
 ADO.NET objects, session state, and, 299–305
AutoPostBack property, 212
DownList control, 210, 232–233
DataSourceID property, 208
DataSource property, 208, 236
DetailsView control, 210, 230–232
FormView control, 209, 228–230
GridView control, 209, 226–228
ListControl base class, 209
Menu control, 209
Repeater control, 210
TreeView control, 209
 value associated with, 212
 data cache
 accessing, 341
 benefits of, 321–323
 cache dependencies, 334–336
 callback delegates, 341
 clearing, 338–340
 controlling cache entries, 327
 deleting items from, 341
 inserting items in, 341
 inserting items with dependencies, 341
 inserting items with expiration times, 341
 managing, 327, 331–333
 retrieving data from, 503–504
 searching, 324
 SQL Server dependency, 336–338
 using, 324–325
 data caching, 321–341, 386
 backing files, 334
 impact of, 325–327
 output caching, 343–358
 removal callbacks, 338–340
 data collection, multistage, 312–320
DataContract attribute, 543
DataContractFormat attribute, 543
DownList control, 210, 232–233, 236
DataMember attribute, 543
 data members, storing, 535
 data models, for MVC sites, 458
 data providers, managing, 170
 data readers, 218–221
 holding, 328
DataSet class, 219–221
CommandBuilder, 221
DataSets, 328–331
 Data Source Configuration dialog box, 245
DataSource controls, 208, 221–226
 attaching to data-bound controls, 221
DataSourceID property, 208
DataSourceMode property setting, 223
DataSource property, 208, 236
 data sources
 attaching data-bound controls to, 208
 configuring, 224
DataSourceMode property, 223
 for navigation controls, 256
DataTable arrays, 219
 displaying, 236
 DataTables, 302, 328–331
 in-memory, 328–330
 synthesizing programmatically, 328–329
 DataTextFields, 236
 DataValueFields, 236
DbProviderConfigurationHandler, 216
 Debug Description attribute, 145
 debugging, 374–377
 page tracing, 363–370
 preparing Web sites for, 383
 starting, 375
 Visual Studio support for, 374
 Web applications, 383
 Debug, Step Into command, 376
 Debug, Step Over command, 376
 DeclarativeCatalogPart, 271, 277, 278
 declarative data binding, 208, 221. *See also* data binding
 DetailsView support of, 210
 FormView support of, 209
 GridView support of, 209
 Menu support of, 209
 TreeView support of, 209
 Decrypt method, 190
 default.aspx.cs files, 53
 default.aspx files, 53
 for MVC applications, 455
 default configuration settings, 167
 defaultRedirect attribute, 378
 delegates for event handlers, 105
 DeleteProfile method, 260
 dependencies
 for cached items, 334–336
 in output caching, 353
 dependency properties, 535
 deployment, 575–586
 configuration files for, 582
 precompiling for, 578
 Visual Studio support for, 578–585
 deployment packages, creating, 583
 DesignDisplayMode, 274
 Designer (Visual Studio)
 controls, support for, 110
 event handlers, wiring, 105
 user controls, support for, 110, 111, 117
 visual development in, 67
 Design mode
 placing controls in, 84
 switching to, 78
 Design Patterns: Elements of Reusable Object-Oriented Software (Gamma, Helm, Johnson, Vlissides), 451
 Design tab (Visual Studio), 54
 Design view (Visual Studio), 68
 inserting text in, 69
 DetailsView control, 210, 230–232
 device profiles, session state and, 311
 DHTML (Dynamic HTML), 478
 directories, for Web application files, 26
 discretionary access control lists (DACLs), 182
 display names, resolving to URLs, 237
 Display property, 128
 Dispose method, 397, 404
 distributable applications, indirection layer for, 557
 distributed computing solutions, fragmented nature of, 555–556
<div> tags, 69
 attributes, setting, 69
 DLLs, 14
 ISAPI DLLs, 13–14, 19
 DockPanel, 435
 Document Object Model (DOM) in Silverlight applications, 533–534, 551
 Silverlight interaction with, 516, 533–534

DragPanel extenders, 482
DropDown extenders, 482
DropDownList boxes, 9, 209
 adding items to, 72–73
 editing items in, 72–73
 rendering, 207–208
DropShadow extenders, 482
Duration attribute, 344, 347
dynamic content, 9–18
 rendering, 60, 448
Dynamic Data model,
 DynamicValidator controls, 128
Dynamic HTML (DHTML), 478
DynamicPopulate extenders, 482

E

EditDisplayMode, 274
EditorZone, 276, 288
Empty Web site template, 146
Enabled key, 371
Enabled property, 240
EnableViewState Explicit attribute, 145
Encrypt method, 190
endpoints (WCF), 557–558
 address, 558
 bindings, 558
 contracts, 558
 loosely coupled, 557
 wire encoding, 557
EndRequest event, intercepting, 392–393
environment variables, 164
Error events, handling, 380–382
ErrorMessage property, 125
 setting, 122
error messages
 management of, 124, 128
 from validator controls, 120
error pages, 378–381
Errors window (Visual Studio), 57
event handlers
 adding, 74
 adding with text wizard, 104–105
 creation of, 93
 delegates for, 105
 event handling, 390–393
events
 AJAX *Timer* control, 490–497
 application-wide, 385, 391–392
 exposing, 92–94
 firing, 92–93
 handlers for, creating, 93

managing, 100
responding to, 94
Silverlight events, 526–527
Events button (Properties pane), 93
exceptions handling, 381–382
executable blocks, 21
executable code
 adding as a script block, 34–35
 adding inline, 32–33
 compiling, 46
 managing, 43–46
 marking, 31, 179
ExecuteReader, 218
execution model, 33
 compilation model, 35
 object-oriented, 25
execution tags <% %>, 31
expirations
 absolute, 331–332
 on cached data, 331–333
 sliding, 333–334
Expression Blend, 516, 527
extender controls, 477, 501–512
eXtensible Application Markup Language. *See XAML*

F

fields, validating, 120–128
file mappings
 for .axd extension, 409
 for HTTP handlers, 419
 for virtual directories, 17–18
file name extensions
 .asp, 19
 .aspx, 29
 .axd, 409
 mappings to binary components, 36
 mappings to ISAPI DLLs, 13
 .master, 145
 .xaml, 448
 .xbap, 441
file system, loading Web applications from, 12
File System Web sites, 50–51, 577, 585
 creating, 58
 precompiling, 578
FilteredTextBox extenders, 482
Flash, 514
Flex, 514
folders, adding to a project, 203
Forbidden handler, 409–410

<form> tags, *runat* attribute, 63–64
Forms Authentication, 36, 184–189
 enabling, 196
 target file, 188
 using, 206
FormsAuthentication class, 188, 189, 190
 methods of, 190
 Set Auth cookie, 206
 SignOut method, 206
FormsCookieName method, 190
FormsCookiePath method, 190
Forms keyword, 166
FormView control, 209, 228–230, 236
free-form layout, 209
FrontPage 2002 Server Extensions for Web site development, 51
FTP Web sites, 51, 576, 585
 creating, 58

G

Gamma, Erich, 451
GDI and GDI+ interfaces, 434, 436
Generic Handler template, 417, 419
GetAuthCookie method, 190
GetAverageLengthOfRequest method, 401
GetElementById, 551
get_isInAsyncPostBack method, 499
GetProductsCompletedEventArgs argument, 548–549
GetPropertyValue method, 260
GetRedirectUrl method, 190
GET requests, 4, 10–11, 35
GetResponse method, 6–7
GetVaryByCustomString method, 351
global application class, adding, 338
Global Application Class template, 338, 388
Global.asax.cs files, 338, 387
 global event handlers, 390
Global.asax files, 387
 vs. HTTP modules, 404
 server-side script block, 387
global assemblies
 sharing, 419
 signing and deploying, 117

global assembly cache, binary composite controls in, 117
 global configuration files, viewing, 180
 global state making thread safe, 389–390
 storing in modules, 400–404
 graphical user interfaces (GUIs). *See also* UIs (user interfaces), modular, 62
 Graphics Device Interface, 434, 436
Grid.Column property, 448
Grid layout panel, 435, 528, 551 adding content to, 448
 adding to Page, 448
Grid.Row property, 448
Grid ShowGridLines property, 528
GridView controls, 209, 226–228, 236
AlternateRowStyle property, 228
PagerSettings property, 228
PagerStyle property, 228
 group element, 261
 GUI components packaging, 62, 63
 server-side, 62

H

handler mappings, 23, 38–39
 handlers. *See also* event handlers for requests, 62
HashPasswordForStoringInConfigFile method, 190
 header information, viewing, 5
 HEAD requests, 4
HelloWorld2.aspx file, 32 output of, 33
HelloWorld3.aspx file, 34
HelloWorld4Code.cs file, 43
HelloWorld5Code.cs file, 44
HelloWorld.aspx.cs file, 56
HelloWorld.aspx file, 29, 30, 57
HelloWorld.htm file, 28 converting to ASP.NET application, 29
HelloWorld Web application building, 26–30 in Visual Studio, 52–58
 Helm, Richard, 451
hideSkiplink style, 244
 hierarchical data binding, 209
HomeController, 457
 host pages, subscription to events, 92

HotSpot Collection Editor, 130
HotSpotMode property, 130
 hot spots defining, 130
 editing, 130–131
HotSpots property, 130
HoverMenu extenders, 482
 HTML (Hypertext Markup Language), 8–9 in ASP content, 60–62 changes between versions, 95 integrating Silverlight content with, 533–534 over a disconnected protocol, 67 and page, layer of indirection between, 62 rendering, 95 tables, rendering in, 96 test pages for Silverlight content, 517–518 versions of, 8 XAML content, integrating, 442–447
Html32TextWriter class, 95
Html.BeginForm, 470
HtmlDocument class, 533
GetElementById, 533
GetElementByName, 533
AttachEvent method, 551
GetProperty method, 551
 SetProperty method, 551
 HTML files converting to ASP.NET applications, 29 creating, 28
 HTML forms, 10–11 postbacks, 10
 HTML markup, rendering, 105
HtmlPage class
 Document property, 551
 RegisterCreatableType, 551
 RegisterScriptableObject, 551
 HTML source, viewing, 30
 HTML streams, renderings of, 8–9
 HTML tags attributes of, 9
<form>, 9
** tag, 128
<input type=image /> tag, 129
<option> tag, 207
<select> and *</select>*, 9, 207 sending to browser, 59 views, adding to, 472
HtmlTextWriter class, 80, 90, 95, 95–96
AddStyleAttribute, 96
RenderBeginTag, 96 tag-rendering methods, 100 using, 96 versions of, 95
 HTTP 1.0, 4
 HTTP 1.1, 4
HttpApplication, 38, 46–48, 386–387, 391–392 dictionary, 386 overriding, 387–393
HttpApplicationState, 80
Lock method, 389
HttpBrowserCapabilities object
 SupportsRedirectWithCookie property, 311
HttpCachePolicy class, 351–352
 SetCacheability method, 352–353
HttpContext, 386
 Cache property, 341
 Current property, 387
 Profile property, 259
 RewritePath method, 251
User.Identity.Name key, 261 for WCF applications, 561
HttpContext objects, 38, 46, 48–49, 291 properties in, 49
 HTTP GET/POST idiom drawbacks of, 473 improving, 473. *See also* AJAX
HttpHandler interface, 50
 HTTP handlers, 50, 405–420. *See also* custom handlers
 add attribute, 406
 built-in, 407–410
 creating, 419
 file mapping, 419
Forbidden handler, 409–410 generic handlers, 417–419 lifetime of, 48 request handlers, 405–407 session state and, 416 *Trace* handler, 407–409 *type* element, 406 *validate* attribute, 406 verbs, 406
httpHandler section (*web.config* file), 419
 HTTP (Hypertext Transfer Protocol), 4
HttpModules, 46, 49
 HTTP modules, 394–403 creating, 395 features handled by, 395

- HTTP modules (*continued*)
 vs. Global.asax, 404
 implementing, 396–397
 project-level references to, 398
 storing state in, 400–403
 viewing, 398–400
- HTTP requests/responses, 4–7. *See also* requests
 from a browser, 5–6
 endpoint of, 48
 headers with, 95
 handlers for, 17–18
 header information, 5
 inbound port for, 12
 information about, 49
 listening for, 12–13
 managing, 48–50
 MVC handling, 450. *See also* MVC Framework
 object representing, 80
 payload of, 100
 routing of, 35–42
 _VIEWSTATE field, 100
 without a browser, 6
- HttpResponse* objects,
 CacheControl property, 353
- HttpRuntime*
AppDomainAppPath property, 566
 Cache property, 573
http://schemas.microsoft.com/winfx/2006/xaml namespace, 448
http://schemas.microsoft.com/winfx/2006/xaml/presentation namespace, 448, 523
- HTTPS (Secure HTTP), inbound port for, 12
- HTTP.SYS, 39
- HttpVerbs.Post* enumeration, 468, 472
- HTTP Web sites, 52, 576, 585
 creating, 58
- HttpWorkerRequest* class, 46
- HyperLink* controls, 282
 adding to master pages, 204
NavigationUrl method, 204
- hyperlink Web Parts, creating, 281–288
- |
- ICollection* interface, 208
IDataReader interface, 218–219
ID attribute, 74
- identity configuration elements, 311
- IEnumerable* interface, 208
- IHierarchicalDataSource* interface, 238, 241
- IHierarchicalEnumerable* interface, 238, 241
- IHttpHandler* interface, 50, 406, 410–411, 419
 implementation of, 25, 46, 48, 405
- IHttpModule*, 397, 404
- IIS Content View, 14–15
- IIS Features View, 14–16
- IIS (Internet Information Services), 12–13
 Anonymous Authentication mode, 183
 application pooling, 31
 ASP.NET, configuring from, 174–180
 ASP.NET pipeline and, 46–47
 ASP.NET, working with, 39, 40
 Classic mode, 37
 client authentication, 36
 C++/native core server API, 16
 custom handlers, configuring for, 414–415
 Default Web Site node, 26–27
 directory space, 13
 Handler Mappings page, 414
 hosting Web sites in, 174
 HTTP modules and, 395
 Integrated mode, 37–38
 loading Web applications with, 12
 output caching, 36
 port 80, listening on, 35
 request handling, 36–42
 resources, surfing to, 23
 running, 14–18
 securing, 183–184
 security context of, 182, 183
 security identity, viewing, 183
 session state configuration page, 307
 Trace.axd request handling, 409
 URL Rewrite Module, 255
 virtual directories of, 13, 26
 Windows authentication support, 183
- IIS management console starting, 23
- IListSource* interface, 208
- image-based controls, 128–132
- ImageButton* controls, 129
- OnClientClick* property, 129
- PostBackUrl* property, 129
- Image controls, 128–130
- ImageUrl* property, 129, 230, 249
- ImageMap* controls, 129
 adding to Web Forms, 130
AlternateText property, 130
HotSpotMode property, 130
ImageUrl property, 130
- images
 hot spots in, defining, 130
 managing and organizing, 129
 tooltips for, 130–131
- ImageUrl* property, 129, 130, 249
- Impersonation* property, 559
- implicit properties, 211
- ImportCatalogPart*, 271
- INamingContainer* interface, 103
- indexer notation, 341
- indexers, for *Session* object, 293
- Index* method, 455, 460
 index views, generating with, 464
- Inherits* attribute, 44, 145
- InitializationComplete* handler, 274–275
- initialization files (.ini files), 164
- Init* method, 397, 404
- InitOutputCache* method, 352
- in process, storing session state, 306
- Insert* method
Cache object, 327–328
 overloads, 328, 338
- installers, for Web applications, 585
- Integrated mode (IIS)
 handler mappings, 37–38
 module mappings, 36, 37
- interactive applications, 10–11
- interactive content, 520
- Internet Information Services.
See IIS (Internet Information Services)
- Internet Information Services (IIS) Manager
 Application Settings pane, 177–178
 ASP.NET Configuration Settings pane, 180
 Connections pane, 176
 Connection Strings pane, 176–177
 Features View pane, 176
 opening, 176

Internet Services Application Programming Interface, 13–14, 19, 32
inventory binding code, 322–323
 ISAPI DLLs, 13–14, 19, 32
IsapiFilterModule module, 16
IsapiModule module, 16
IService1.cs file, 562
IsReusable property, 410–411, 419
IsValid property, 125, 126
 item changed event handlers, 223–224
ItemCommand handler, 301
Items collections, 207–208
IValidator interface, 125

J

Java applets, for GUI components packaging, 62
 JavaScript
 client-side validation and, 125
 managed code, accessing in, 551
 Silverlight components, accessing with, 533
 JavaScript libraries
 for AJAX client-side support, 480
 Silverlight.createObjectEx helper function, 526
 Java Virtual Machine, Java applet support, 62
 Johnson, Ralph, 451

K

keywords for configuration, 166
keywords namespace, 448

L

<label> element, 121
 labels
 adorning, 70
 ControlToAssociate property, 121
 editing content, 71
 for validator controls, 121
Language attribute, 63, 145
 Language Disassembler (ILDASM), 41
 viewing assemblies in, 41–42
 Language Integrated Query (LINQ), 215, 234–236

language syntax, choosing, 53–54
 layout
 Silverlight schemes, 528–533
 options for controls, 77, 78
LayoutEditorPart, 272
 layout panels, 435
 Silverlight, 528–533
LayoutRoot, 545
 lineage of pages, 56
 line breaks, rendering, 105
 LINQ (Language Integrated Query), 215, 234–236
 queries, constructing, 234–235
LinqDataSource control, 208
 LINQ To SQL template, 472
 LINQ to SQL wrapper classes, 458
ListBox controls, 209
 adding items to, 150–152
ListControl base class, 209, 215
ListItem Collection Editor dialog box, 72
ListSearch extenders, 483
LiteralControl for line breaks, 105
 literal text, rendering, 105
localhost, 27
 local IIS Web sites, 50
 creating, 53
localOnly key, 371
 Locals window, 376–377
LocalSystem, access rights, 31
Location attribute, 347, 352–353, 359
location element, 168, 180
Lock method, 389
LoggedInTemplate template, 200
 logical trees, in Windows-based vs. browser-based application, 435
 login composite controls, 102
 login controls, 119, 200–203
LoginName control, 200
 login pages
 basic page, 186–188
 creating, 201
 with Forms Authentication, 185
 optional, 191–194
 Visual Studio-created, 201
LoginStatus control, 200
LoginUrl method, 190
LoginView control, 200
LogOnUserControl, 457
 long-running operations
 canceling, 499–501
 updating progress of, 497–501
 look and feel, 143–144

master pages, 145–155
 skins, 159–160
 themes, 155–159
 loopbacks, 308
 loose XAML files, 436, 437, 438, 442, 444

M

machine.config files, 165
 provider keys in, 216–217
 configuration section handlers, 165–166
 default .NET configuration settings, 167
 location of, 166
 updating, 167
machine.config.comments file, 165
 Macromedia Flash, 448
MainPage class, 522, 528
MainPage constructor, Silverlight visual tree in, 522–523
MainPage.xaml.cs file, 521
MainPage.xaml file, 521
 Manage Access Rules link, 203
 managed code, 16–18
 accessing in JavaScript, 534, 551
ManageForm method, 412
ManageUI, 493, 496
 mangled URLs, 261
 marker interfaces, 416
MaskedEdit extenders, 483
Master attribute, 146
 master configuration files, 394–395
.master extension, 145
MasterPage directives, attributes of, 145–146
MasterPage.master file, 146
 <body> tag, 147
 master pages, 145–155
 absolute positioning on, 150
 adding content, 149–150
 .aspx pages, similarities to, 147
 automatic, 179
 banners on, 153–154
 content of, 145
 menus, adding to, 152
 ScriptManager control on, 479
SiteMapPath control, adding, 246
 Visual Studio-generated code for, 146–147, 149
 Web Forms based on, 244

- Master Page template, 146
 master web.config file
 HTTP handlers in, 406–407
 httpModules section, 394–395
 member variables, 74
 accessing, 64
 exposing, 211–212, 236
Menu controls, 209, 237–238, 241, 256
 data source for, 245
 MaximumDynamicDisplayLevels property, 246
 StaticDisplayLevels property, 246
 menu items
 adding, 152
 NavigateUrl property, 238
 menus
 adding to master pages, 152
 hideSkiplink style, 244
 messages (WCF), 559
 message traffic, MVC
 management of, 450
method attribute, 10–11
 method calls, HTTP requests as, 405–406
 methods, stepping into and over, 376
 Microsoft SharePoint, 268
Microsoft.SharePoint.
 WebPartPages.WebPart, 267
 Microsoft Silverlight. *See* Silverlight
 Microsoft Visual Basic Controls (VBXs), 62
 Microsoft Web platform, 12. *See also* IIS (Internet Information Services)
ModalPopup extenders, 433, 483, 508–512
 configuring, 511
mode keyword, 166
 Model-View-Controller (MVC)
 software development pattern, 449, 452. *See also* MVC framework
 ASP.NET and, 452–453
 Modify Style dialog box, 70, 157

tag settings, 69
 module mappings, 23
 viewing, 16–17
 modules. *See* HTTP modules
mostRecent key, 371
 multistage data collection, 312–320
MultiView controls, 136–138
 ActiveViewIndex property, 137
 adding Views to, 136
MutuallyExclusiveCheckBox extenders, 483
 MVC framework
 application state management, 450
 architecture, 449–452
 controllers, adding, 472
 message traffic management, 450
 postback events handling, 453
 postbacks processing, 468
 request path, 455–456
 testing and, 454
 UI management, 450
 views, creating, 472
 view templates, 453
 Visual Studio wiring of, 455
 vs. Web Forms, 453
 MVC Site template, 456
 MVC Web sites
 creating, 456–462, 472
 data model for, 458
 delete views, 470–472
 details views, 466
 edit views, 469
 index views, 466
 links, adding, 465
 navigation, 461
 Site.css file, 457
 Site.master file, 457
 updating entries, 463–472
 view code, 459
- ## N
- namespaces, XAML and, 523–524
NavigateUrl property, 238
 navigation, 237–240. *See also* site maps
 security trimming and, 251
 URL mapping and, 251–255
 URL rewriting and, 255
 navigation controls, 237–239
 adding, 256
 data source for, 256
 pointing to site map, 243
 using, 241–243
 navigation requests
 intercepting, 256
 redirecting with URL mapping, 251–255
- redirecting with URL rewriting, 255
 navigation structure, in-memory representation of, 239–240
NavigationUrl method, 204
 navigation URLs, setting, 153
 nesting of controls, 102
.NET assemblies. *See also* assemblies
 pages compiled into, 41
.NET configuration, machine config, 165
.NET configuration directory, 167
.NET Globalization page (ASP.NET Configuration Settings pane), 180
.NET run-time parameters, 165
.NET databases, 215–221
.NET developers, Silverlight and, 516
.NET Framework
 configuration, 164–174
 LINQ extensions, 234–235
 WCF, 555
 XML configuration files, 164
.NET Framework 3.5, implicit properties, 211
.NET interfaces, [ServiceContract] attribute, 573
 networking layer, 480
NetworkService account, 31
 New Data Source command, 245
 New Project dialog box, 6
 New Solution Configuration dialog box, 580
 New Style dialog box, 77
NextResult method, 219
NoBot extenders, 483
 nonalphanumeric characters, stripping out, 89, 90
 nonexistent URLs, mapping to ASPX files, 251–255
NoStore attribute, 347
NumericUpDown extenders, 483
- ## O
- ObjectDataSource* control, 208
 object type arguments, 548
 Observer pattern, 452
OnClientClick property, 129
 One-Click Publish, 579
OnEndRequest handler, 401
OnMenuItemDataBound event, 249–250

on/off attributes, 378
 opening tags (HTML), 9
OperationBehavior attribute, 559
OperationBehaviorAttribute attribute, 573
 operations, updating on progress of, 497–501
<option> tags, 67
 out-of-band request handling, 475, 478, 479
 output cache dependencies, 353 managing, 346–354
OutputCache directive
Duration attribute, 344
Location attribute, 352–353, 359 modifying, 346–351 parameters of, 347–348 placing on page, 343
Shared property, 354 syntax, 344
VaryByCustom parameter, 347
VaryByHeader attribute, 359
VaryByParam attribute, 344, 359
outputCacheProfile elements, 353, 359
OutputCacheProvider elements, 358
outputCacheSettings section, 353 output caching, 163, 343–358, 359 alternate providers for, 358 by IIS, 36 caching profiles, 353–354 effective strategies for, 357–358 locations for content, 352–353 page appearance and, 357 setting up, 343 of user controls, 354–357 output caching parameters, asterisks in, 357

P

Package/Publish Settings, 583
_Page objects, 66
Page1.xaml.cs files, 439
Page1.xaml files, 439
PageCatalogPart, 271
Page class, 30. See *System.Web.UI.Page* class contents of, 514 request handling, 405 server-side validation management, 125
Trace property, 363–370

UI processing, 405
Validate method, 126
 page content based on different browsers, 347 cached, locations of, 352–353 caching, 343–346
Page directive, 30, 63 adding, 29
 page elements, themes for, 155–161
Page_Load events, handling, 63
Page_Load method
CacheWithFileDependency(), 336 calling, 323 updating properties in, 249 in *Wizard* controls, 317
 page loads, costs of, 325. *See also* data caching
pageOutput key, 371
 page refreshes, 473
 page-rendering process bypassing, 343. *See also* output caching composite controls and, 102
PageRequestManager class, 499
abortPostBack() method, 499
get_isInAsyncPostBack method, 499
PagerSettings property, 228
PagerStyle property, 228
 pages. *See also* Web Parts pages:
 Web pages absolute positioning on, 150 browsing to, 28–29 compiling into assemblies, 41 content placeholders, 145 controls, adding, 80 controls, declaring on, 85 controls, placing on, 84 control tree for, 64, 66–67 events management, 100 initialization of, 80
IsValid property, 126 layout options, 77, 78 lineage of, 56 loading of, 80 master pages, 145–155 properties of, 69
Register directives, 144 rendering contents of, 80 state of, encoding, 97 styles, adding, 77 unloading of, 80 user controls, adding, 115
 XAML files, declaring in, 448
 Pages And Controls page (ASP.NET Configuration Settings pane), 180
 page tracing, 66, 363–370 configuring, 370–371 enabling, 383 turning on, 364
 page validation, 125–127
PageView controls, 136
PagingBulletedList extenders, 483
 partial classes, 45
 partial-page updates, 477, 484–489 implementing, 512 triggers for, 488 *UpdatePanel* support of, 479
 Passport authentication, 189
PasswordRecovery control, 200
 passwords, hard-coded, 191, 194
PasswordStrength extenders, 483
 PATH environment variable, 164
 A *Pattern Language* (Alexander), 451
 Patterns movement, 451
 performance
 AJAX and, 476 application performance, 100, 102 data caching and, 321 improving, 433–434 nesting controls and, 102 output caching and, 345–346 precompiling for, 577–578 request processing chain, plugging into and, 394 round-trips to database and, 323 view state management and, 100
 per-request dictionary, 49
 personal information management, 257–258. *See also* personalization; user profiles
Personalizable property, 283
 personalization, 257–266, 271 AJAX support for, 477 anonymous, 262 personalization providers, 258–259 user profiles, 258. *See also* user profiles using, 259–265 Web Parts and, 272
PersonalizationProvider class, 258

personalization providers, 258–259
physical directories, mappings to URLs, 13
PopupControl extenders, 483
port 80, 26
 monitoring, 12
 requests on, 35
port 443, 12
port 42424, 308
portals, 268, 269
 Web Parts and, 268
postback events, MVC handling
 of, 453, 468, 469
postbacks, 10
 from ASP.NET controls, 489–490
 asynchronous, 490
 canceling, 499–501
 timed and automatic, 480, 490,
 496–497, 512
PostBackUrl property, 129
postprocessing, 46
POST requests, 4, 10–11
posts, maintaining state between,
 97–100
precompiled assemblies, 577
precompiling, 44, 577–578, 585
 for deployment, 578
 for performance, 577
preprocessing, 40, 46
PresentationCore reference, 444
PresentationFramework reference,
 444
ProcessRequest method, 410–411,
 419
ProcessRequest method
 (*IHttpHandler*), 50
ProductsServiceClient class, 548
ProfileBase class, 260
profile information
 saving, 260
 using, 259–260
Profile property, 259
profile providers, 258
profiles
 accessing properties of, 266
 anonymous, 261
 defining settings, 266
 deleting, 260
 grouping and nesting, 261
profile submission handler,
 262–264
 saving, 260, 264
 using, 259–260
profile schemas, defining, 259
Program.cs file, 7

Progress control, 497
ProgressTemplate, 498
progress updating, 497–501
project templates
 ASP.NET Web site, 53
 Empty Web Site, 52
 HTTP site, 52
Project Wizard (Visual Studio), 50
properties
 implicit properties, 211
 of validation controls, 128
Properties pane, 71
 Events button, 93
PropertyGridEditorPart, 272, 286
protocol channels, 558
provider factories, 216–217
provider pattern, 216
 personalization providers,
 258–259
Provider property, 240
providers for output caching, 358
Providers property, 240
ProxyPartManager, 270
publishing Web applications, 583,
 585
PUT requests, 4

Q

query string parameters, varying
 cached content on, 348–351
query strings, 11

R

radio button controls, selection
 handlers, 225
RadioButtonList control, 209
RangeValidator controls, 120, 127
Rating extenders, 483
Read method, 219
Really Simple Syndication (RSS),
 557
record sets, disconnected, 219
RedirectFromLoginPage method,
 190
RedirectToAction method, 472
References node, 444
reflection for view code,
 generating, 459
RefreshConversation, 495, 496
regenerateExpiredSessionId
 setting, 311
RegisterCreatableType method,
 534

Register directive, 85
 TagPrefix attribute, 86
Register directives, 144
RegisterScriptableObject method,
 534
registry, editing, 164
Regular Expression Editor,
 126–127
regular expressions, for validation,
 126–127
RegularExpressionValidator
 control, 120
 adding to Web Forms, 126–127
remoteOnly attribute, 378
remote Web sites, 51
removal callbacks, setting up,
 338–340
remove instruction, 168
RenderContents method, 80, 85,
 100
 HtmlTextWriter methods, using
 in, 96
overriding, 83, 100, 102
removing, 103
rendered controls, 101. *See also*
 controls
 vs. composite controls, 101–102
rendering code, browser version
 independent, 100
rendering controls, 79. *See also*
 controls
Render method, 288
ReorderList extenders, 483
Repeater control, 210
request duration, tracking,
 400–403
request handlers, 405–407
request handling facility, 407–410
requestLimit key, 371
request paths of MVC
 applications, 455–456
request pipeline, 62–63
request processing, inserting
 functionality in, 394–403
Request property (*HttpContext*
 and *Page*), 95
request-response pattern for
 WCF messages, 559
requests. *See also* HTTP requests/
 responses
 asynchronous handling of, 474.
 See also AJAX
 authenticating manually, 206
 Authentication tickets for, 189
 context information, viewing,
 366

requests (continued)

- handlers for, 63
- out-of-band handling, 475
- routing tables for, 454
- time stamping, 396–398
- waiting for, 473–474

RequiredFieldValidator controls, 120

- adding to Web Forms, 122

RequireSSL method, 190

ResizableControl extenders, 483

resources

- associating with user roles, 203
- DACLs of, 182
- surfing to from IIS, 23
- virtual directories for, 13

Response class *HttpCachePolicy*, 351

Response object, 21, 32, 75

- Cache* property, 359

responses, 5, 7. *See also* HTTP requests/responses

- Authentication* tickets for, 189
- generating, 22
- header information, 5

Response.Write, 75

result sets, managing, 218–236

reverse compiling, 41

RewritePath method, 251

Rich Internet Applications (RIAs), 473–474

- Silverlight generation of, 513

RootNode property, 240

RootVisual property, 521, 551

RoundedCorners extenders, 483

round-trips to server, 474–475

- client-side validation and, 125
- reducing, 433–434

RoutedEventArgs argument, 440

RoutedEventArgs parameter, 521

routed events, 526–527

RouteTable class, 455

routing policies, changing, 453

routing tables, 454

RowDefinitions property, 448, 545

runat=server attribute, 35, 63–64, 66

runaway threads, avoiding, 501

runtime, enabling session state, 416

S

scalability, application dictionary size and, 389–390

ScriptableMember attribute, 551

ScriptableType attribute, 551

script blocks, executable, 34–35

scripting issues, 515

ScriptManager controls, 479

- adding to page, 484–485, 512

ScriptManagerProxy controls, 479

<script> tags, 31

Seadragon extenders, 484

security

- authentication services of ASP.NET, 189–194
- authorizing users, 203–206
- configuring, 206
- Forms Authentication*, 184–189
- IIS, securing, 183–184
- login controls, 200–203
- of Web applications, 31
- user access, managing, 194–199
- Windows security, 182

security identity of IIS, 183

security trimming, 251

securityTrimmingEnabled attribute, 251

<select> tags, 67

Select A Single Provider For All Site Management Data link, 195

SelectedNodeChanged events, 135

SelectionChanged event handler, 546

selection controls, selected item management, 97

selection handlers, adding, 213

semidynamic content, 437

ServerControl1 control, 82–84

servers

- CGI support, 11–12
- code execution on, 31
- executable script blocks on, 34–35
- GUI componentization on, 62
- requests to, 6–9

server-side controls, 59

- adding to page, 78
- attaching collections to, 210–215
- composite controls, 79
- for layer of indirection, 62
- for literal text and HTML markup, 105

Items collections, 207–208

navigation controls, 237–239

rendering controls, 79

ScriptManager control, 479

ScriptManagerProxy control, 479

Silverlight control, 525–526

size, changing, 78

style template support, 159

Timer control, 480

UpdatePanel control, 479

UpdateProgress control, 480

validator controls, 120, 125

view state management, 97–100

- vs. Web Parts, 267

server-side script blocks, 63

server-side validation, 125–126

- See also* validation

Service1.svc.cs file, 562

Service1.svc file, 562

ServiceBehavior attribute, 559

ServiceBehaviorAttribute attribute, 573

Service class request handling, 405

ServiceContract attribute, 544

service contracts

- behaviors, 558–559
- creating, 564, 573
- implementing, 564, 573

ServiceModel Metadata Utility Tool, 573

service references

- adding, 568
- adding to Silverlight projects, 547

services. *See also* WCF services

- contract and policy based, 557
- explicit boundaries between, 557

Services control panel, 308

Session_End event, 391

session identifiers as cookies, 309–311

Session member, 260

Session objects, 291

- indexer for, 293
- inserting and retrieving data from, 293, 296–297
- objects stored in, 299
- role of, 292

Session_Start event, 391

session state, 292, 386

- accessing, 320
- accessing specific values in, 320
- adding objects to, 294

ADO.NET objects, data-bound controls, and, 299–305

ASP.NET support of, 292–293

configuring, 306–308

custom handlers and, 416

custom providers for, 311

session state (*continued*)
 device profiles and, 311
 disabling, 306, 307, 320
 enabling, 163
 managing, 163, 180
 retrieving data from, 492–493
 storing in a database, 306, 308
 storing in a SQL Server database, 308, 311, 320
 storing in a state server, 306, 307–308, 320
 storing *InProc*, 306, 307, 320
 timeouts, 311, 320
 tracking with cookies, 309–310, 320
 tracking with URLs, 310–311, 320
 trying, 293–297
sessionState configuration settings, 166
 Session State management feature (ASP.NET Configuration Settings pane), 180
sessionString member variable, 295, 298
Set Auth cookie, 206
SetAuthCookie method, 192
SetCacheability method, 351–353
SetETag, 351
SetExpires, 351
SetLastModified, 351
SetMaxAge, 351
SetPropertyValue method, 260
SetRevalidation, 351
SetTargetProperty method, 540
SetValidUntilExpires, 351
SetVaryByCustom, 351
Shared attribute, 347
Shared property, 354
 shopping carts, 299
 shout boxes, 490
ShowContent method, 192
ShowLineage methods, 34, 35, 56
ShowMessageBox property, 123
SidebarTemplate, 312
SignOut method, 190, 206
 Silverlight, 448, 515–516
 animations, 535–542
 architecture, 521–522
 dependency properties, 535
 Expression Blend and, 527
 features of, 515–516
 integrating with HTML, 533–534
 layout schemes, 528–533, 551
 WCF services and, 542–551

XAML role in, 522
 Silverlight 1.0, 515
 Silverlight 2 and 3, 515
 Silverlight applications, 517
 compiling, 524
 creating, 517–521
 generating, 551
 HTML Document Object Model in, 551
RootVisual property, 521–522
 visual tree, 522–523
 Silverlight Application template, 551
 Silverlight class members, 527
 Silverlight content adding to Web pages, 524–526
 ASP.NET site for, 517–519
 HTML test page for, 517–518
 integrating with Web site, 542–550
 interactive, 520
 project node for, 518–519
Silverlight control, 525–526
 Silverlight control events, 526–527 handling, 526
 Silverlight controls, 526–527
Silverlight.createObjectEx helper function, 526
 Silverlight events, 526–527
Silverlight.js file, 519
Silverlight.createObjectEx helper function, 526
 Silverlight keywords, 523
SilverlightSiteTestPage file, 519
 singleton software pattern, 387
Site.css file, 457
SiteMap class, 239–240
 events and properties of, 240
SiteMapDataSource
ShowStartingNode property, 246
StartFromCurrentNode property, 246
 site map data sources, 237, 238
SiteMapNode, 239, 240
 methods and properties of, 240
 site map nodes, 239
 custom attributes, adding, 248
 editing, 245
 nesting, 246
SiteMapPath control, 237–238, 241–242, 256
 adding to master pages, 246
 site map providers, 237, 239
 managing, 239
SiteMapResolve event, 240

SiteMapResolve event handler, 247–248, 256
 site maps, 239. *See also* navigation adding, 256
 blank top-level node, 243
 configuring, 242–243
 creating, 243–247
 custom attributes for nodes, 248–250
 default, 239
 updating, 239, 243
Site Map template, 244
Site.master files, 457
 tabs, adding, 465
 site nodes, custom attributes for, 248–250
Skin File template, 160
 skins, 159–161
 applying, 161
 creating, 160
 file storage, 159
Slider extenders, 484
SlideShow extenders, 484
SlidingExpiration method, 190
 sliding expirations, for cached items, 333
 SOAP, 405, 557
 SOA principles, 557
 software design patterns, 451–452
 Solution Explorer (Visual Studio), 54, 55
 MVC folders, 457
 Package/Publish Settings, 583
 References node, 444
 source code
 compiling, 577
 generated by Visual Studio, 74–75
 precompiling, 577–578, 585
 viewing, 55, 75, 410
 Source code mode, switching to, 78
 Source tab (Visual Studio), 54
 Source view (Visual Studio), 55, 69
 Split tab (Visual Studio), 54
SqlCacheDependency class, 336–337
SqlDataAdapter, 220
SqlDataSource control, configuring, 221–222
SqlDependency attribute, 347
SqlPersonalizationProvider class, 259
 SQL Server dependencies, 336–338
Src attribute, 44, 146

StackPanel layout panels, 435, 439–440, 528, 532–533, 551
StartNavigationTemplate, 312
StartsWith method, 504
stateNetworkTimeout, 311
 state service, turning on, 308
 static helper methods, adding to MVC applications, 468
 static pages, 9
 stepping into methods, 376
 stepping over methods, 376
 stepping through code, 376
Storyboard, 539
 SetTarget method, 540
Strategy pattern, 452
Strict attribute, 145
String types, *StartsWith* method, 504
*StripNonAlphanumeric*s method, 103, 112
 strong names, 166
 Structured Query Language (SQL), 217–218
 style definitions, text-based, 155–159
 styles, modifying, 157
 style sheets, building, 156–157
 Styles menu Add Style Rule option, 156
 subdirectories, configuration settings for, 180
Substitution controls, 349
SupportsRedirectWithCookie property, 311
System.Data.DataView class, 502
System.Diagnostics.Debug, 374
System.Diagnostics.Trace calls, tracing, 374
System.Runtime.Serialization namespace, 543
System.ServiceModel.Activation namespace, 544
System.ServiceModel namespace, 543
[System.Web.Script.Services]
 ScriptService] attribute, 504
System.Web.Services.WebService, 46
System.Web.SessionState.
 IRequiresSessionState interface, 416
System.Web.UI.Control class, 79–81
 elements of, 80
System.Web.UI.Page class, 33, 46, 59, 79
 control collection of, 80
 controls, iterating, 80
 object-oriented approach, 63
 properties, methods, and events, 80
System.Web.UI.Page handler, 405
System.Web.UI.UserControl
 user controls, 110
System.Web.UI.WebControls.
 ContentPlaceHolder controls, 145
System.Web.UI.WebControls.
 WebParts.WebPart class, 269
 deriving classes from, 280–281
System.Web.UI.WebControls.
 WebControl class, 83
System.Windows.Browser.
 HtmlDocument class, 533
System.Windows.Browser.
 HtmlPage class, 533
System.Windows.HtmlPage class, 534
 RegisterCreatableType method, 534
 RegisterScriptableObject method, 534

T

tabbed panes, 136–138
 tables
 adding to user controls, 114
 rendering in HTML 3.2 and HTML 4.0, 96
Tables collection, 219
Tabs extenders, 484
TabStrip controls, 136
TagPrefix attribute (*Register* directive), 86
TcpTrace, 5
 testing
 against local version of IIS, 50
 application-specific features, 51
 MVC applications, 454
 text
 editing, 69
 inserting, 69
 text boxes
 TextMode property, 121
 Text property, 91
TextBoxWatermark extenders, 484
Text File template, 448
TextMode property, 121
Text property
 changing, 85
 modifications, 106–107
 setter for, 92, 106–107, 107, 114
 setting, 87–88
TextTextField property, 236
text transfer, 4
TextValueField property, 236
Theme directive, 158
 theme folders
 creating, 156
 skin files in, 159
 themes, 155–159
 applying, 159
 creating and using, 156–159
 predefined, 155
this keyword, 33
Threads window, 377
Tick events, 496
The Timeless Way of Building (Alexander), 451
timeout configuration setting, 311
timeouts, session, 311, 320
Timer controls, 480, 490–497, 512
 creating chat pages with, 491–497
 default interval, 496
 default settings for, 490
 Tick event, 496
TimeSpan, 401
timing modules, 398
 implementing, 396–398
ToggleButton extenders, 484
Toolbox
 adding items to, 84, 91, 100
 opening, 70
 sorting items in, 84
 user controls in, 110
Trace.axd handler, 411
Trace.axd resource, 407–408
TraceFinished event, 373–374
Trace handler, 407–409
traceMode key, 371
Trace objects, adding trace statements to, 368–369
Trace property, 363–370
trace statements, 367–370
Trace.Warn, 368
Trace.Write, 368
tracing, 188, 206
 application tracing, 370–374
 enabling, 383
 enabling programmatically, 373
 page tracing, 363–370
TraceFinished event, 373–374

tracing (*continued*)
 turning on, 64–65, 78
 tracing messages, managing, 374
 tracing output
 for application tracing, 372
 context information, 366–367
 control tree, 365
 managing, 373–374
 trace statements, 367–370
 tracing information, adding, 368–370
 transport channels, 558
 trapping exceptions, 381–382
 tree controls, 132. *See also* TreeView controls
 tree node events, handling, 134–135
 tree nodes
 building, 135
 editing, 133–134
 TreeView controls, 132–135, 209, 237–238, 241, 256
 adding to Web Forms, 133
BorderColor property, 134
BorderStyle property, 134
 data binding support, 135
 formatting, 133
SelectedNodeChanged events, 135
 TreeView Node Editor, 133–134
 TreeView Tasks menu, 133
 triggers
 for partial-page updates, 488
 for *UpdatePanel*, 512
Triggers collection, 489
Triggers property, 512
 troubleshooting, 363. *See also* debugging; exceptions; handling; tracing
type element, 406
 types, scriptable, 551
 type system extensions, 479

U

UIs (user interfaces)
 AJAX support of, 476–477
 consistency in, 143–144. *See also* master pages; skins; themes
 MVC management of, 450
 packaging as components, 62–67
 processing, 405
 progress updates, 497–501

refreshing, 493, 494
 responsiveness, improving, 474.
See also AJAX
 of user controls, 110
 UI programming, Windows-based, 434
 unit testing with MVC framework, 454
UpdatePanelAnimation extenders, 484
UpdatePanel controls, 479, 512
 adding to page, 485, 496–497
Triggers collection, 488, 489, 512
 using, 506
UpdateProgress controls, 480, 497–501
 adding to page, 498
AssociatedUpdatePanelID property, 498
ProgressTemplate, 498
 URL mapping, 251–255, 256
 MVC management of, 453
urlMappings element, 251
 URL Rewrite Module, 255
 URLs
 mangled URLs, 261
 resolving display names to, 237
 tracking session state with, 310, 320
UseCustomControl.aspx, 84
 markup, 86
UseDeviceProfile option, 311
 user access management, 181, 194–199, 206. *See also* authorization
 user controls, 101, 110–117, 528
 adding to pages, 115
 advantage of, 117
ArrayList, 114
 caching, 354–357, 359
 default properties, lack of, 113
 deployment of, 117
 Designer support for, 111, 117
 disadvantage of, 117
 output of, 116–117
 page trace of, 116
System.Web.UI.UserControl
 derivation, 110
Table, adding, 114
Text property, 113
 UI component, 110
 UI functionality, grouping in, 117
 using statement for *System.Collections*, 114
User.Identity.Name key, 261
 user information management, 257–258. *See also* personalization
 user input handling, 10
 multistage data collection, 312–320
 validating, 120–128
 user input controls, 10
 user interfaces. *See* UIs (user interfaces)
 user preferences, storing, 258
 user profiles, 261
 accessing properties of, 266
 defining, 258, 266
 deleting, 260
 grouping and nesting, 261
 profile submission handler, 262
 saving, 260, 264
 using, 259–260
 user roles
 associating with resources, 203
 authorization based on, 203–205
 creating, 196–197
 security trimming and, 251
 users
 adding to Web site, 197
 authenticating, 181, 198
 authorizing, 181, 198–199, 203–206
 WebPart controls, adding, 288
 Web site customization, 267. *See also* Web Parts
 user state, associating with session, 291

V

validate attribute, 406
Validate method, 125, 126
 validation, 119–128
 client-side, 125
 failure of, 126
 server-side, 125–126
 using regular expressions, 126–127
 validation controls, 120–128
ControlToValidate property, 120
 custom logic of, 125
Display property, 128
EnableClientScript property, 125

validation controls (*continued*)
 grouping, 128
 properties of, 128
 tags associated with, 125
ValidationGroup property, 128
 validation expressions, selecting, 126
 validation functions, 128
ValidationGroup property, 128
 validation handlers, 125
 validation script blocks, 128
ValidationSummary controls, 120
 adding to Web Forms, 123
ShowMessageBox property, 123
ValidatorCallout extenders, 484
 variables, displaying values of, 376
var type, 461
VaryByContentEncoding attribute, 347
VaryByCustom attribute, 348, 351
VaryByCustom parameter, 347
VaryByHeader attribute, 348, 351, 359
VaryByParam attribute, 344, 348, 351, 359
 modifying, 351
 verbs, 406
 view code, 459
 View Code button (Visual Studio), 55
 View controls, 136–138
 views
 adding content to, 137
 adding HTML tags to, 472
 adding to *MultiView* controls, 136
 adding to MVC projects, 459, 464, 467, 472
 managing, 136
 navigating between, 137
 view state, 386
 management of, 97–100
 storing, 100
`_VIEWSTATE` field, 65
ViewState property (*Page* class), 93, 97–100
 view templates, 453
 virtual directores
 adding, 27
 Anonymous Authentication for, 183
 creating, 23
 definition of, 13
 file mappings, 17
 file type support, 23
 for source code, 26
 handler mappings, 23
 module mappings, 16–17, 23
 physical path for, 28
 storing Web sites in, 50
 viewing in IIS, 15
 virtual URLs, mapping to ASPX files, 251–255
 Visual Studio
 Administrative mode, 52
 and ASP.NET, 50–58
 ASP.NET code generated by, 69
 asynchronous proxy method generation, 568
 building a page in, 68–75. *See also* pages
 Call Stack window, 377
 code generated by, 54
 controls, adding with, 67–78
 debugging support, 374–377
 deployment support, 578–585
 Design view, 68
 FTP server connectivity, 51
 Hello World application, 52–58
 HTML files, creating, 28
 layout of, 54
 Locals window, 376–377
 MVC wiring, 455
 prebuilt login page, 199
 Properties pane, 71
 running as administrator, 175
 Source view, 69
 Threads window, 377
 Toolbox, adding items to, 91
 Watch window, 377
 WCF contract placeholder files, 562
 Web application configuration files, 167–168
 Web application development in, 67–68
 Web Control Library, default code for, 82
 Web site templates, 576–577
 Visual Studio Web server, 50
 visual trees, constructing, 522–523
 Vlissides, John, 451

W

WarningLevel attribute, 146
 Watch window, 377
 Wayback Machine, 514
 WCF, 555
 ASP.NET and, 560–561
 behaviors, 558–559
 channels, 558
 compatibility mode, 561
 elements of, 557–559
 endpoints, 557–558
 messages, 559
 role of, 556–557
 service contracts, 573
 side-by-side mode, 560
 Web sites enabled for, 573
 WCF clients, building, 567–572, 573
 WCF contracts, 558
 Visual Studio-generated placeholders, 562
 WCF proxies, Silverlight-enabled, 548
 WCF Service Application template, 562
 WCF services
 calling asynchronously, 570–571
 exposing, 563, 573
 service contracts for, 564
 service references, 568
 Silverlight and, 542–550, 551
 writing, 561–567
 WCF Service template, 573
 Web Application projects, 518
 Web applications. *See also* applications
 building, 57
 configuration files for, 167–168
 debugging, 41
 deploying, 578–586
 developing, 3
 evolution of, 514–515
 executable code, adding, 32–33
 global space for, 385–404
 installers for, 585
 loading, 12
 loose XAML files in, 442
 One-Click Publish, 579
 precompiling, 577–578
 publishing, 583, 585
 running, 57
 security of, 31
 storage in temporary directory, 42
 Web-based security, 182–189
WebBrowsable property, 283
 Web browsers. *See* browsers
web.config file, 167–168
 anonymous access setting, 262
 application settings, adding, 170–173, 581

web.config file (*continued*)
application tracing settings, 325
authentication node, 184–185
authorization element, 199
authorization node, 184–185,
 192–193
child files, 167
compiler tracing, 374
creating, 170
custom error attributes, 378
customErrors section, 379
debugger setting, 375
<deny users="*" node, 204
forced authentication settings,
 186
Forms Authentication,
 implementing in, 184–189
handlers in, 406
httpHandlers section, 413, 419
httpModules section, 398
identity configuration element,
 311
location element, 180
login URL specified, 201
managing with Web Site
 Administration Tool, 170–172
outputCacheProfile elements,
 359
outputCacheSettings section,
 353
output caching configuration,
 359
personalization properties,
 defining in, 258–259
<*profile*> element, 261, 266
profile schemas, defining in, 259
sample, 165
securityTrimmingEnabled
 attribute, 251
session state settings, 311–312
site map configuration settings,
 242–243
top-level, 167
tracing, enabling in, 370–371
transform for configuration
 changes, 581
transforming for deployment,
 579
urlMappings element, 251
WCF service contracts, 573
Web Control Library, default code
 for, 82
web.debug.config file, 167, 582
Web development, 3
 issues of, 21–22
WebDisplayName property, 283

Web Forms, 449
adding to Web sites, 53
based on master pages, 148–
 149, 244
vs. MVC framework, 453
sign-in forms, 121–124
user controls in, 110
Web packaging, 579
Web pages. *See also* pages
adding Silverlight content,
 524–526
appearance of and output
 caching, 357
building with Web Parts,
 272–280
cacheable, creating, 344–346
Cache property, 341
lifetime of, 296
partial page updating, 512
SharePoint based, 268
static pages, 9
storing multiple versions of, 359
timed automatic posts from,
 512
visual style definition, 155–159
WebPart controls, enabling for,
 288
WebPageTraceListener type, 374
WebPart controls
 built-in, 271–280
 dynamic additions of, 288
 enabling Web pages for, 288
WebPartManager, 269, 270, 288
Web Parts, 267
 adding to catalog, 284–285
 application development, 269
 architecture, 269–280
 built-in, 271–272
 connecting, 271
 creating, 288
 derivation of, 267
 developing, 280–288
 development scenarios, 269
 display modes, 274
 enabling sites for, 288
 history of, 268
 page development, 269
 Render method, 288
 vs. server-side controls, 267
 server-side controls managed
 by, 288
 uses of, 268–269
 using, 272–280
 zones, 270, 270–271

Web Parts pages, 269
CatalogZone, 288
editing capabilities, 288
EditorZone, 288
switching display modes,
 275–276
WebZone, 288
WebPart Toolbox, 271–272
WebPartZone class, 270
 settings for, 272
web.release.config file, 167, 582
WebRequest class, 6
 GetResponse method, 6–7
Web servers. *See* IIS (Internet
 Information Services); servers
WebService class, 405
Web service idiom, AJAX use of,
 475
Web Service projects, creating,
 543–550
Web Setup projects, 585
Web Site Administration Tool
 (WSAT), 163, 169–172, 180,
 206
Add New Access Rule link, 203
Application tab, 170–171
Create Application Settings link,
 170
editing web.config with, 184
Manage Access Rules link, 203
Provider tab, 170, 195
Security tab, 170, 195, 197, 198
web.sitemap file, 239
site nodes, custom attributes
 for, 248–250
Web site performance. *See also*
 performance
 view state management and,
 100
Web site projects, 50–51
Web sites
 adding items to, 53
 adding WPF-style content to,
 436–437, 442–447
AJAX *AutoComplete* extender
 for, 507
asynchronous background
 processing, 474
control flow with Forms
 Authentication, 185
debugging, 383
dynamic content, 9–18
enabling for AJAX, 512
enabling for WCF, 573

Web Sites (*continued*)
 File System Web sites, 50–51, 58, 577, 585
 FTP Web sites, 51, 58, 576, 585
 hosted by IIS, 174–175
 HTTP Web sites, 52, 58, 576, 585
 local, 50
 look and feel of, 143–144. *See also* master pages; skins; themes
 navigation support, 237–256
 packaging for deployment, 579–586
 personalization support for, 257–266
 portal-type, 268–269
 profile schema, defining, 259
 remote, 51
 security for, 181, 206. *See also* security
 testing application-specific features, 51
 testing locally, 50
 user configuration of, 267. *See also* Web Parts
 Web Parts, enabling for, 288
 XAML files, adding, 448
 Web site templates, 576–577
Web.Staging.config file, 582–583
 Web User Control template, 111
WebZone, 288
Win32 API, 434
Win32 Graphics Device Interface (GDI), 434
 Windows authentication, 189
 IIS support for, 183
 Windows-based user interface programming, 434
 WindowsBase reference, 444
 Windows Communication Foundation. *See WCF*
 Windows configuration, 164
 Windows Forms Controls, 62
 Progress control, 497
 Windows Internet Explorer autocomplete feature, 507
 Windows Live ID, 189
 Windows operating system environment variables, 164
 initialization files (.ini files), 164

Windows Presentation Foundation. *See WPF*
 Windows security, 182
 Windows Workflow Foundation, 555
 wire encoding for WCF-based endpoints, 557
Wizard controls, 138, 312–320
 adding controls to steps, 314
 adding steps to, 313
 auto formatting, 312–313
Page_Load method, 317
SidebarTemplate, 312
StartNavigationTemplate, 312
StepType, 313
WizardStep Collection Editor dialog box, 313
 worker processes, in ASP.NET pipeline, 47–48
 WPF, 434–441, 555
 features available through, 434–435
 layout panels, 435
 Silverlight and, 516
 uses of, 433–434
 WPF applications
 creating, 435
 logical tree, 435
 WPF-based content
 as loose XAML files, 437–438
 presenting, 436–437
 WPF Browser Application template, 438, 448
 WPF content
 deploying with XBAPs, 438–441
 rendering, 448
 serving, 442–447
 WPF layouts, top-level nodes, 442
 WPF namespace, 448
WrapPanel layout panel, 435
 wrapper classes
 creating, 458
 for database tables, 472
writeToDiagnosticsTrace key, 371
writeToDiagnosticsTrace option, 374
WriteXml method, 566

X

XAML, 522–524
 namespaces and, 523–524
 role in Silverlight, 522
 visual trees, constructing, 522–523
 in Web applications, 436
 for WPF layouts, 435
 XAML-based browser applications. *See XBAPs*
 (XAML-based browser applications)
 XAML content
 adding, 442
 HTML content, integrating, 442–447
 .xaml extension, 448
 XAML files
 adding to sites, 448
 loose, 436, 437, 438, 442, 444
Pages, declaring in, 448
 XAML plug-in, 448
 XAP files, 524
 .xbap extension, 441
 XBAPs (XAML-based browser applications)
 creating, 438–441, 448
 drawbacks of, 513
xdt:Locator attribute, 583
xdt:Transform attribute, 583
 XHTML document tags, 145
XhtmlTextWriter class, 95
 XML
 configuration files, 164
 DataSet objects serialized as, 220
XmlDataSource control, 208
XMLHttpRequest objects, 490
XmlSiteMapProvider, 239, 242
 XML site maps, 239. *See also* site maps
 "x" namespace, 523

Z

ZoneTemplate, 270