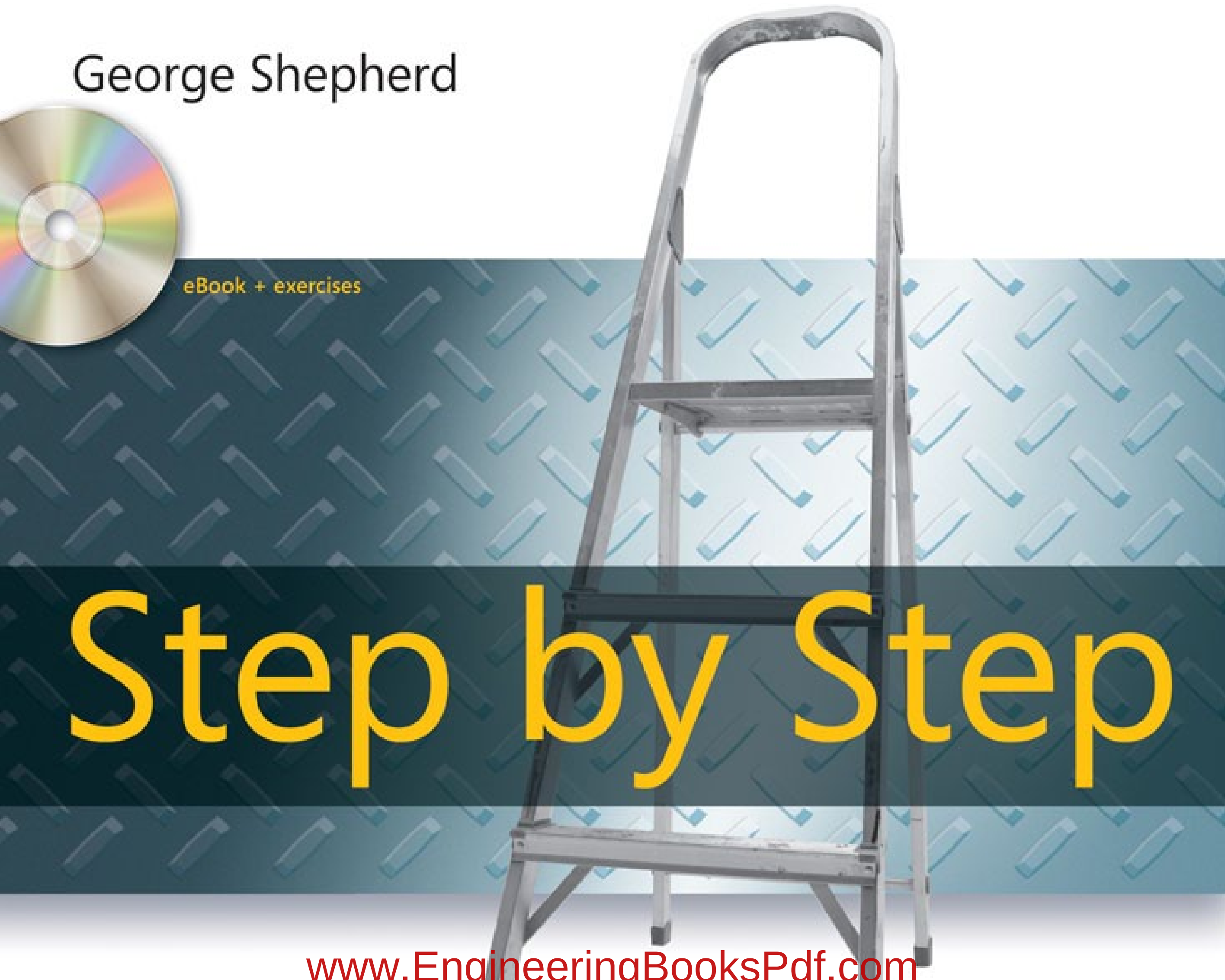
Microsoft

Microsoft®

ASP.NET 4

George Shepherd



Step by Step

How to access your CD files

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

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.

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

Table of Contents

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:

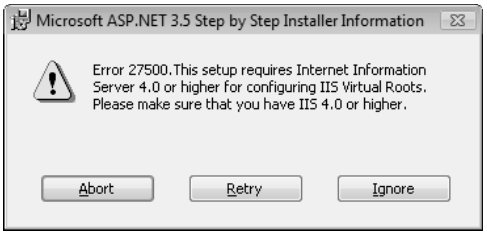
- Visual Studio 2010 is installed on your computer.

  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.



Microsoft ASP.NET 3.5 Step by Step Installer Information

Error 27500.This setup requires Internet Information Server 4.0 or higher for configuring IIS Virtual Roots. Please make sure that you have IIS 4.0 or higher.

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	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	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	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	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.

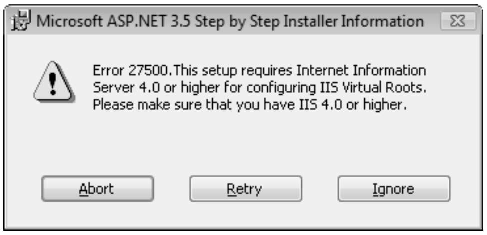
Important Before you begin, make sure that you have

- 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.


Microsoft ASP.NET 3.5 Step by Step Installer Information

Error 27500.This setup requires Internet Information Server 4.0 or higher for configuring IIS Virtual Roots. Please make sure that you have IIS 4.0 or higher.

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 conole, 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 *TreeView*, the *Image*, the *ImageButton*, the *ImageMap*, and the *MultiView/View* 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	Illustrates Forms Authentication and authorization in a Web site
Login.aspx, OptionalLogin.aspx, Web.Config, Web.ConfigForceAuthentication, Web.ConfigForOptionalLogin	Web resources for illustrating Forms Authentication at the very barest level
Chapter 10	
DataBindORama	Illustrates data binding to several different controls, including the *GridView*. Illustrates the *DataSource* 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 *mspinput@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 descendents. 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.")

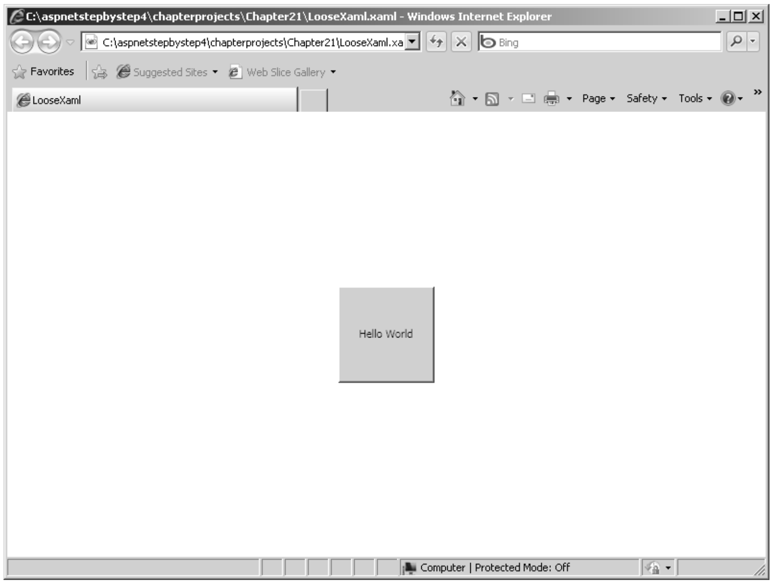

C:\aspnetstepbystep4\chapterprojects\Chapter21\LooseXaml.xaml - Windows Internet Explorer
C:\aspnetstepbystep4\chapterprojects\Chapter21\LooseXaml.xa
Bing
Favorites
Suggested Sites
Web Slice Gallery
LooseXaml
Page
Safety
Tools
Hello World
Computer | Protected Mode: Off

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.

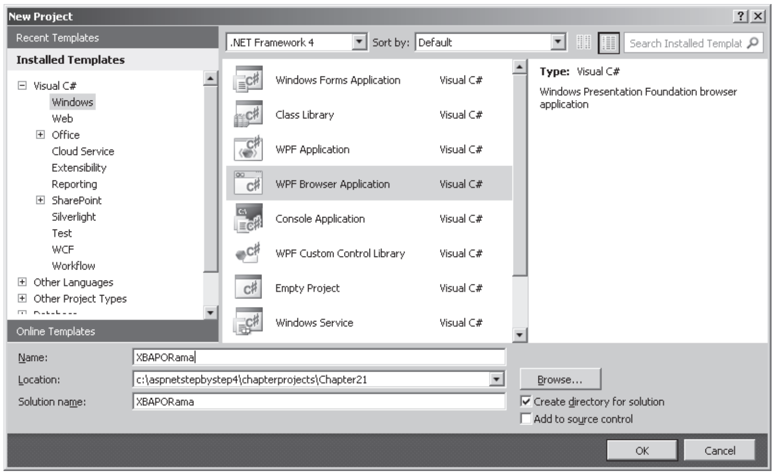
The upside of deploying an application as an XBAP is that it works in very much the same way that a Windows-based desktop application works (though with greatly reduced permissions and tightened security). For example, the application can handle mouse click events and can respond to control events all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



Type: Visual C#
Windows Presentation Foundation browser application

Name: XBAPORama
Location: c:\aspnetstepbystep4\chapterprojects\Chapter21
Solution name: XBAPORama
☑ Create directory for solution
☐ Add to source control

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.

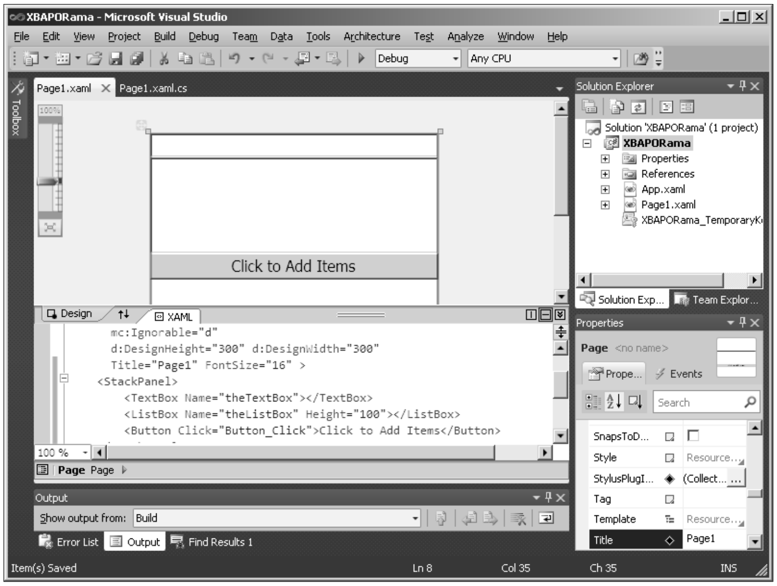
3. Change the layout panel from a *Grid* to a *StackPanel* so that it is simpler to work with. With a *StackPanel*, you can drop in controls and not worry about creating grid columns and rows:

```
<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:



```
      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="Button_Click">Click to Add Items</Button>
```

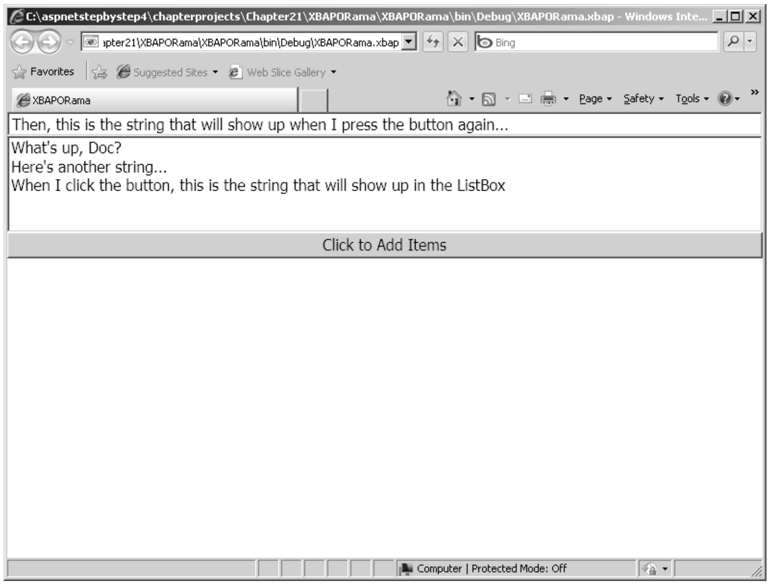
5. Double-click the button to add a handler. Visual Studio creates a handler for the button click. You can find the handler in the code file for the page. Because you didn't name the *Button*, Visual Studio gave the handler a default name of *Button_Click*. The method looks very much like the ASP.NET button click handlers except the second argument is a *RoutedEventArg* instead of the .NET typical *EventArg*.

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.theListBox.Items.Add(this.theTextBox.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):


C:\aspnetstepbystep4\chapterprojects\Chapter21\XBAPORama\XBAPORama\bin\Debug\XBAPORama.xbap - Windows Inte...
apter21\XBAPORama\XBAPORama\bin\Debug\XBAPORama.xbap
Bing
Favorites
Suggested Sites
Web Slice Gallery
XBAPORama
Page
Safety
Tools
Computer | Protected Mode: Off

Then, this is the string that will show up when I press the button again...

What's up, Doc?
Here's another string...
When I click the button, this is the string that will show up in the ListBox

Click to Add Items

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.

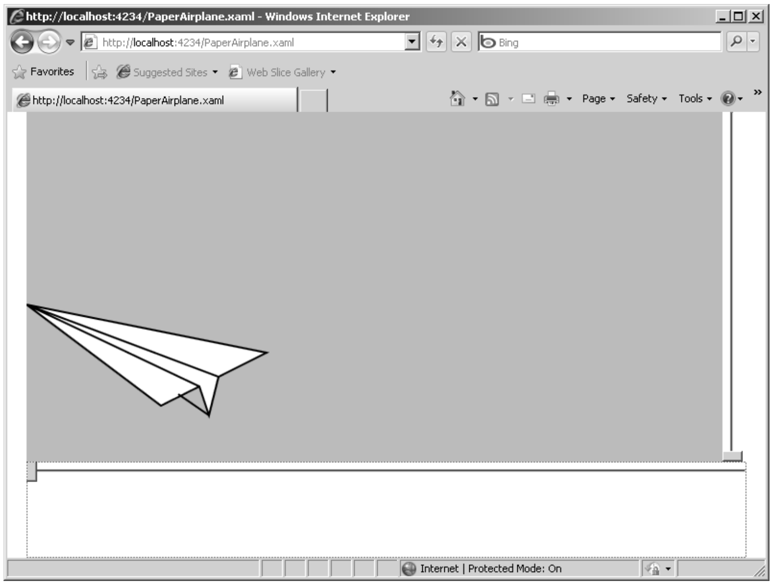
```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="750">
    <Grid
        <!-- Grid column and row definitions are here... -->
        <Canvas Background="SkyBlue" Grid.Row="0"
                Grid.Column="0">
        </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:

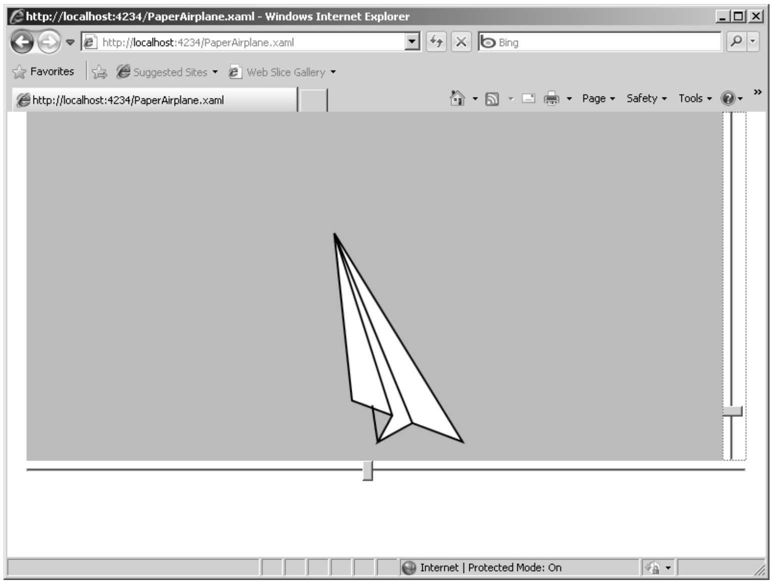
```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="750">

    <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:


http://localhost:4234/PaperAirplane.xaml - Windows Internet Explorer
http://localhost:4234/PaperAirplane.xaml
Bing
Favorites
Suggested Sites
Web Slice Gallery
http://localhost:4234/PaperAirplane.xaml
Page
Safety
Tools
Internet | Protected Mode: On

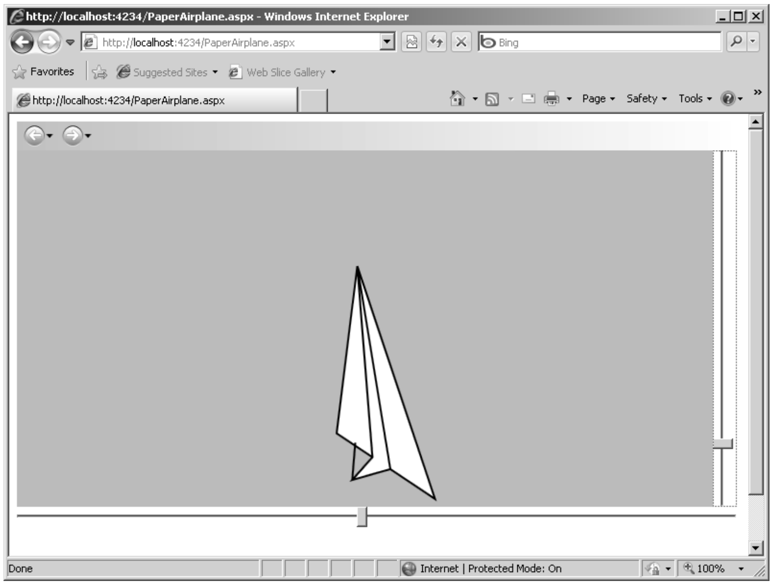
9. Experiment by moving the sliders. Because the vertical slider controls the angle of rotation, moving it up causes the airplane to spin in a clockwise direction. Because the horizontal slider is connected to the *Path*'s *Canvas.Left* property, moving the horizontal slider moves the plane along the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


http://localhost:4234/PaperAirplane.xaml - Windows Internet Explorer
http://localhost:4234/PaperAirplane.xaml
Bing
Favorites
Suggested Sites
Web Slice Gallery
http://localhost:4234/PaperAirplane.xaml
Page
Safety
Tools
Internet | Protected Mode: On

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>
```

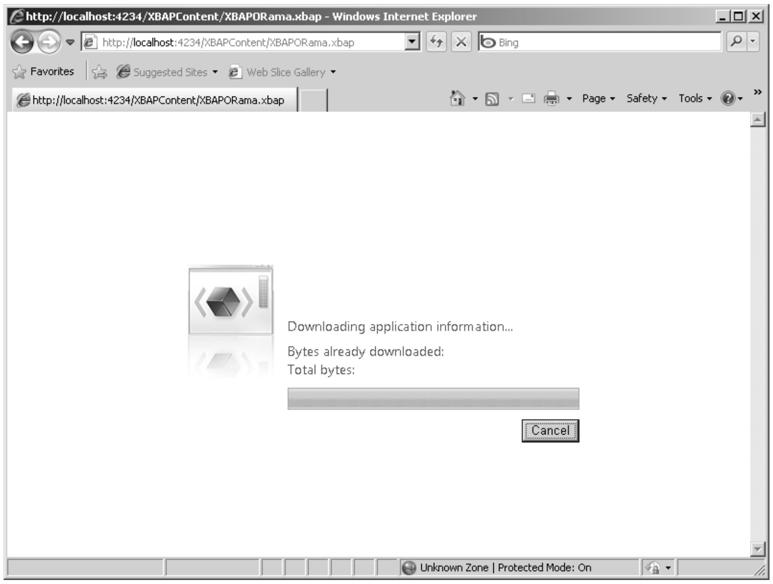
11. Run the page. The PaperAirplane.xaml content appears in a frame in the page. The XAML content has the same functionality in the frame as it did when it was run in the browser:


http://localhost:4234/PaperAirplane.aspx - Windows Internet Explorer
http://localhost:4234/PaperAirplane.aspx
Bing
Favorites
Suggested Sites
Web Slice Gallery
http://localhost:4234/PaperAirplane.aspx
Page
Safety
Tools
Done
Internet | Protected Mode: On
100%

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

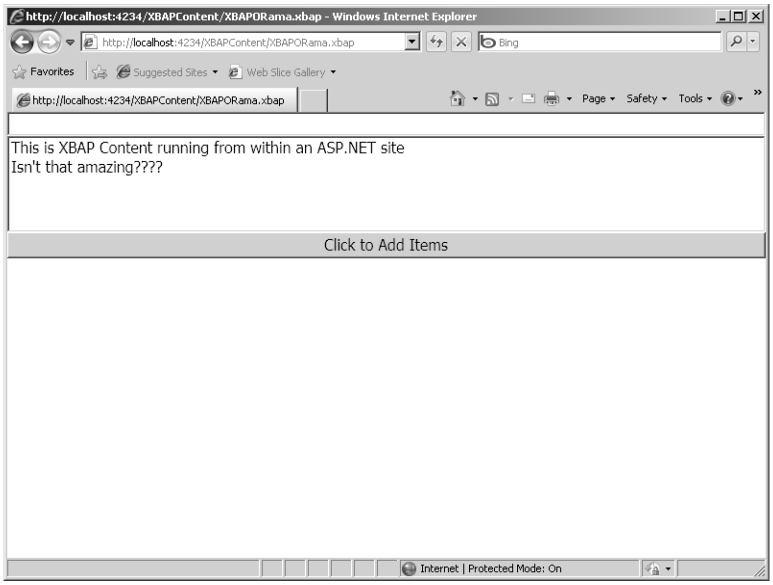
12. Add the XBAP content from the previous example to this site. First, create a new folder under the Project node in Solution Explorer. Name the folder *XBAPContent*. Right-click the new folder and click Add, Exiting Item. Navigate to the previous example's bin directory (on my computer, it is C:\aspnetstepbystep4\chapterprojects\Chapter21\XBAPORama\XBAPORama\bin\Debug). Add XBAPORama.xbap, XBAPORama.exe, and XBAPORama.exe.manifest to this XAMLORama 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.


http://localhost:4234/XBAPContent/XBAPORama.xbap - Windows Internet Explorer
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Cancel
Unknown Zone | Protected Mode: On

Downloading application information...
Bytes already downloaded:
Total bytes:

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


http://localhost:4234/XBAPContent/XBAPORama.xbap - Windows Internet Explorer
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Click to Add Items
Internet | Protected Mode: On

This is XBAP Content running from within an ASP.NET site
Isn't that amazing????

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 <*Page*> tag and an ending </*Page*> tag. Using WPF within XAML requires the standard WPF namespace "http://schemas.microsoft.com/winfx/2006/xaml/presentation" and the keywords namespace "http://schemas.microsoft.com/winfx/2006/xaml" (which is often mapped to "x").
Add a *Canvas* to the *Page*	Use the <*Canvas*> opening tag and the </*Canvas*> closing tag. Nest objects you'd like displayed in the canvas between the opening and closing tags.
Add content to the *Canvas*	Nest objects you'd like to appear on the canvas between the <*Canvas*> opening tag and the </*Canvas*> closing tag. Assign positions within the canvas using the *Canvas.Top* and *Canvas.Right* properties.
Add a *Grid* to a *Page*	Declare a <*Grid*> opening tag and a </*Grid*> closing tag on the page. Use the Grid's *RowDefinitions* and the Grid's *ColumnDefinitions* properties to define the rows and columns.
Add content to the *Grid*	Nest objects you'd like to appear on the canvas between the <*Grid*> opening tag and the </*Grid*> closing tag. Assign positions within the grid using the *Grid.Row* and *Grid.Column* 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 contront 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.

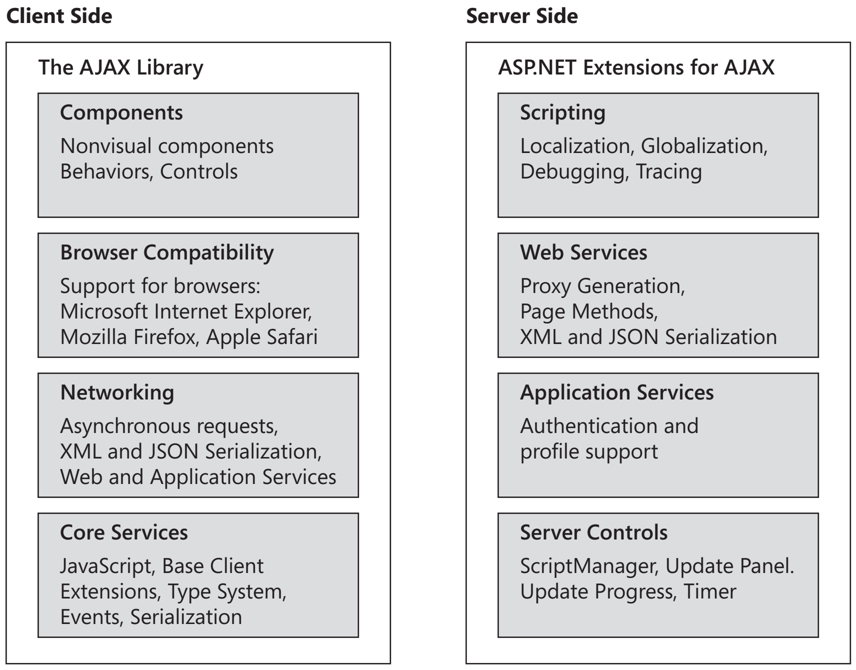


FIGURE 23-1 The conceptual organization of ASP.NET AJAX support layers.

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
Accordion	This extender is useful for displaying a group of panes one pane at a time. It's similar to using several *CollapsiblePanels* constrained to allow only one to be expanded at a time. The *Accordion* is composed of a group of *AccordionPane* controls.
AlwaysVisibleControl	This extender is useful for pinning a control to the page so that its position remains constant while content behind it moves and scrolls.
Animation	This extender provides a clean interface for animating page content.
AsyncFileUpload	This control is for uploading a file asynchronously in the background.
AutoComplete	This extender is designed to communicate with a Web service to list possible text entries based on what's already in the text box.
Calendar	This extender is targeted for the *TextBox* control providing client-side date-picking functionality in a customizable way.
CascadingDropDown	This extender is targeted toward the *DropDownList* control. It functions to populate a set of related *DropDownList* controls automatically.
CollapsiblePanel	This extender is targeted toward the *Panel* control for adding collapsible sections to a Web page.
ConfirmButton	This extender is targeted toward the *Button* control (and types derived from the *Button* 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).
DragPanel	This is an extender targeted toward *Panel* controls for adding the capability for users to drag the *Panel* around the page.
DropDown	This extender implements a Microsoft SharePoint–style drop-down menu.
DropShadow	This extender is targeted toward the *Panel* control that applies a drop shadow to the *Panel*.
DynamicPopulate	This extender uses an HTML string returned by a Web service or page method call.
FilteredTextBox	This extender is used to ensure that an end user enters only valid characters in a text box.
HoverMenu	This extender is targeted for any *WebControl* 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
ListSearch	This extender searches items in a designated *ListBox* or *DropDownList* based on keystrokes as they're typed by the user.
MaskedEdit	This extender is targeted toward *TextBox* controls to constrain the kind of text that the *TextBox* will accept by applying a mask.
ModalPopup	This extender mimics the standard Windows modal dialog box behavior. With the *ModalPopup*, a page can display content of a pop-up window that focuses attention on itself until it is dismissed explicitly by the end user.
MutuallyExclusiveCheckBox	This extender is targeted toward the *CheckBox* control. The extender groups *CheckBox* controls using a key. When a number of *CheckBox* controls all share the same key, the extender ensures that only a single check box will appear selected at a time.
NoBot	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.
NumericUpDown	This extender is targeted toward the *TextBox* 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 *TextBox*.
PagingBulletedList	This extender is targeted toward the *BulletedList* control. The extender enables sorted paging on the client side.
PasswordStrength	This extender is targeted toward the *TextBox* control to help when end users type passwords. Whereas the typical *TextBox* hides only the actual text, the *PasswordStrength* extender also displays the strength of the password using visual cues.
PopupControl	This extender is targeted toward all controls. Its purpose is to open a pop-up window for displaying additional relevant content.
Rating	This control renders a rating system from which end users rate something using images to represent their choice (stars are common).
ReorderList	This ASP.NET AJAX control implements a bulleted, data-bound list with items that can be reordered interactively.
ResizableControl	This extender works with any element on a Web page. Once the *ResizableControl* is associated with an element, the user can resize that control. The *ResizableControl* puts a handle on the lower right corner of the control.
RoundedCorners	The *RoundedCorners* extender can be applied to any Web page element to turn square corners into rounded corners.

Component	Description
Seadragon	The *Seadragon* control renders an image along with buttons for zooming in and out, going to full screen, and panning,
Slider	This extender is targeted to the *TextBox* control. It adds a graphical slider that the end user can use to change the numeric value in the *TextBox*.
SlideShow	This extender controls and adds buttons users can use to move between images individually and to play the slide show automatically.
Tabs	This server-side control manages a set of tabbed panels for managing content on a page.
TextBoxWatermark	*TextBoxWatermark* extends the *TextBox* control to display a message while the *TextBox* is empty. When the *TextBox* contains some text, the *TextBox* appears as a typical *TextBox*.
ToggleButton	This extender extends the *CheckBox* to show custom images reflecting the state of the *CheckBox*.
UpdatePanelAnimation	This extender provides a clean interface for animating content associated with an *UpdatePanel*.
ValidatorCallout	*ValidatorCallout* extends the validator controls (such as *RequiredFieldValidator* and *RangeValidator*). 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-beside 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();
    }
}
```

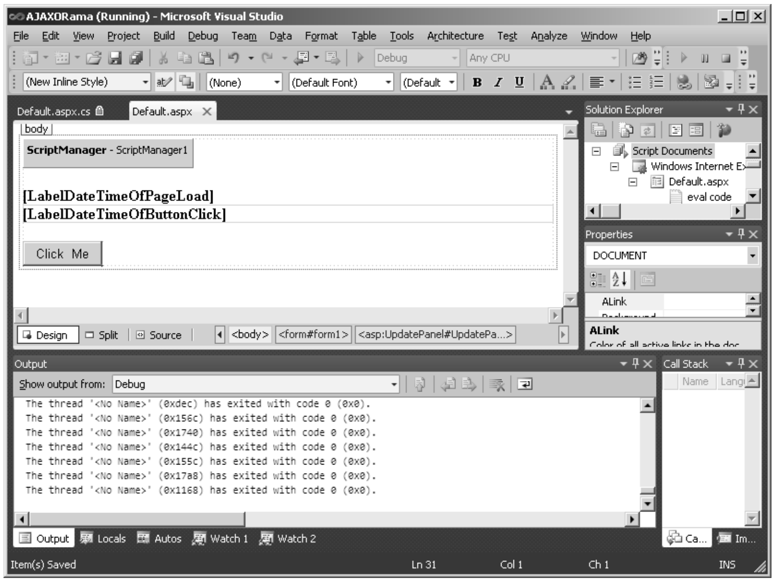
4. Run the page and generate some postbacks by clicking the button a few times. Notice that the label on the page updates with the current date and time each time you click the button.

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();
    }
}
```

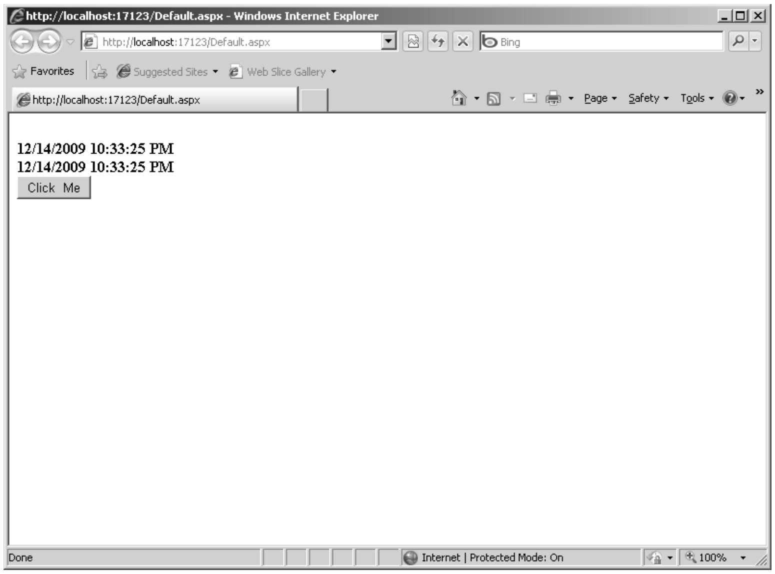
The following graphic shows the *UpdatePanel*, *Button*, and *Labels* as displayed in the Visual Studio Designer (there are some line breaks in between so that the page is readable):



```
The thread '<No Name>' (0xdec) has exited with code 0 (0x0).
The thread '<No Name>' (0x156c) has exited with code 0 (0x0).
The thread '<No Name>' (0x1740) has exited with code 0 (0x0).
The thread '<No Name>' (0x144c) has exited with code 0 (0x0).
The thread '<No Name>' (0x155c) has exited with code 0 (0x0).
The thread '<No Name>' (0x17a8) has exited with code 0 (0x0).
The thread '<No Name>' (0x1168) has exited with code 0 (0x0).
```

7. 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:*


http://localhost:17123/Default.aspx - Windows Internet Explorer
http://localhost:17123/Default.aspx
Bing
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Click Me
Done
Internet | Protected Mode: On
100%

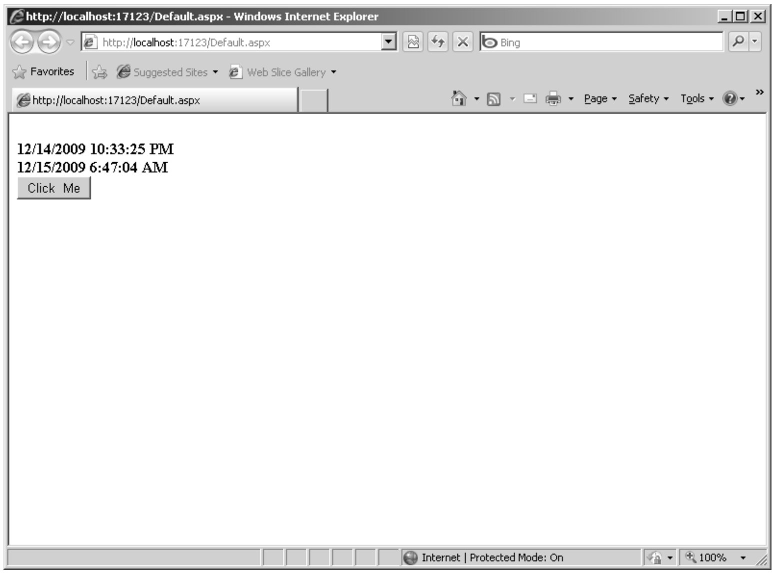
12/14/2009 10:33:25 PM
12/14/2009 10:33:25 PM

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*.

9. 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*:


http://localhost:17123/Default.aspx - Windows Internet Explorer
http://localhost:17123/Default.aspx
Bing
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Click Me
Internet | Protected Mode: On
100%

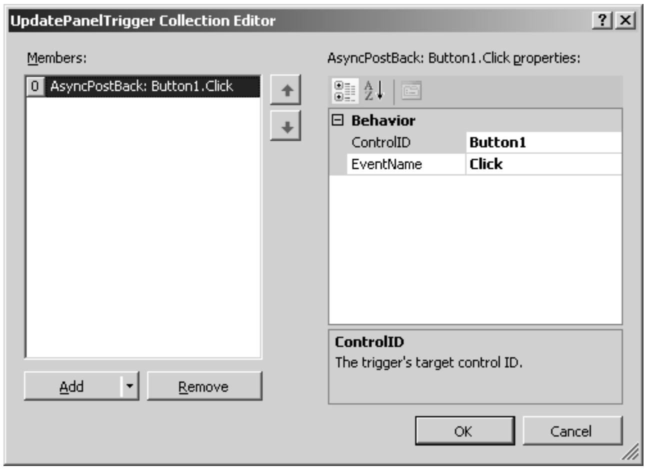
12/14/2009 10:33:25 PM
12/15/2009 6:47:04 AM

10. 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:


UpdatePanelTrigger Collection Editor
Members:
0 AsyncPostBack: Button1.Click
AsyncPostBack: Button1.Click properties:
Add
Remove
OK
Cancel

Behavior
ControlID: Button1
EventName: Click

ControlID
The trigger's target control ID.

(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...
}


}
```

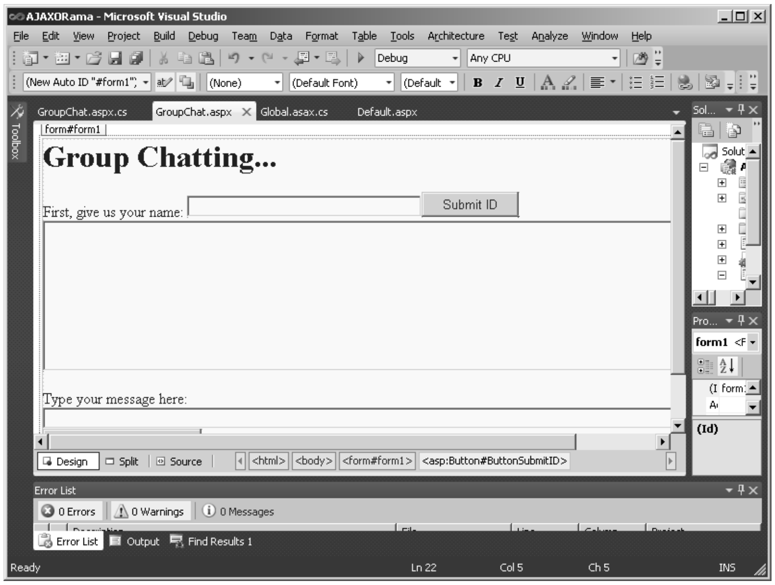
3. Create a chat page by adding a new page to the Web site and calling it *GroupChat.aspx*. This will hold a text box with messages as they accumulate, and it also gives users a means of adding messages.

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();
    }
}
```

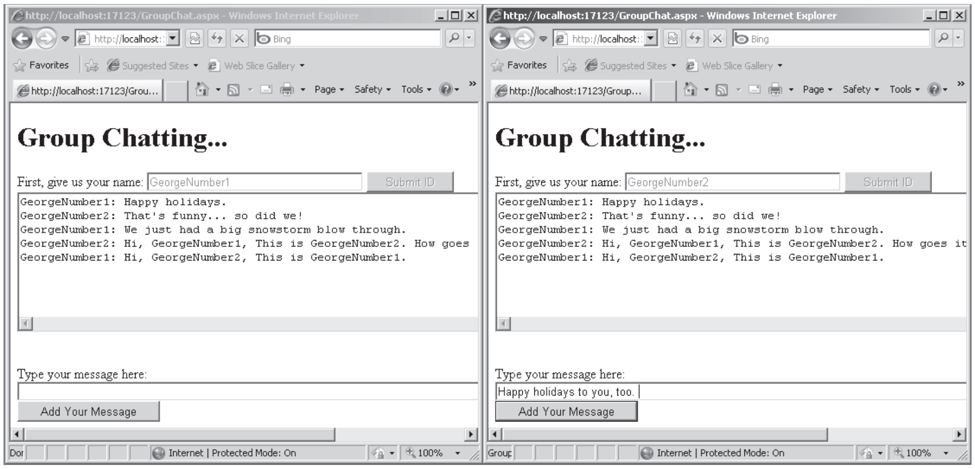
14. Now run the page to see how it works. After you've identified yourself, you can start typing in messages—and you'll see them appear in the conversation *TextBox*. Try browsing the page using two separate browsers. Do you see an issue? The user typing a message gets to see the message appear in the 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*:


http://localhost:17123/GroupChat.aspx - Windows Internet Explorer
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
Internet | Protected Mode: On
100%

Group Chatting...

First, give us your name: GeorgeNumber1 Submit ID

```
GeorgeNumber1: Happy holidays.
GeorgeNumber2: That's funny... so did we!
GeorgeNumber1: We just had a big snowstorm blow through.
GeorgeNumber2: Hi, GeorgeNumber1, This is GeorgeNumber2. How goes
GeorgeNumber1: Hi, GeorgeNumber2, This is GeorgeNumber1.
```

Type your message here:

Add Your Message

Group Chatting...

First, give us your name: GeorgeNumber2 Submit ID

```
GeorgeNumber1: Happy holidays.
GeorgeNumber2: That's funny... so did we!
GeorgeNumber1: We just had a big snowstorm blow through.
GeorgeNumber2: Hi, GeorgeNumber1, This is GeorgeNumber2. How goes it
GeorgeNumber1: Hi, GeorgeNumber2, This is GeorgeNumber1.
```

Type your message here:

Happy holidays to you, too.

Add Your Message

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.

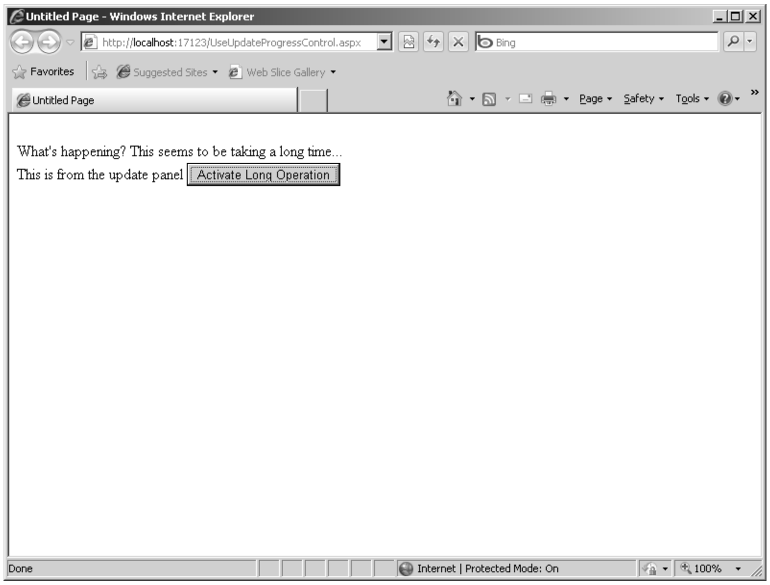
```
public partial class UseUpdateProgressControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }

    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 droplist 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:


Untitled Page - Windows Internet Explorer
http://localhost:17123/UseUpdateProgressControl.aspx
Bing
Favorites
Suggested Sites
Web Slice Gallery
Untitled Page
Page
Safety
Tools
Done
Internet | Protected Mode: On
100%

What's happening? This seems to be taking a long time...
This is from the update panel [Activate Long Operation]

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 cancelation 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*.

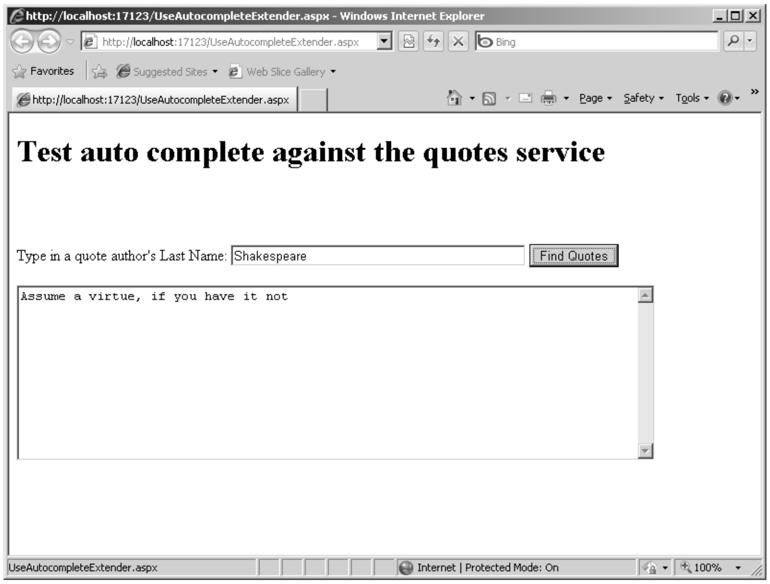
11. Update the *Page_Load* method. It should look up the quotes based on the name in the text box by retrieving the *QuotesCollection* and calling the *QuotesCollection GetQuotesByLastName* method:

```
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:


http://localhost:17123/UseAutocompleteExtender.aspx - Windows Internet Explorer
Favorites
Suggested Sites
Web Slice Gallery
Page
Safety
Tools
UseAutocompleteExtender.aspx
Internet | Protected Mode: On
100%

Test auto complete against the quotes service

Type in a quote author's Last Name: Shakespeare [Find Quotes]

```
Assume a virtue, if you have it not
```

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;&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:

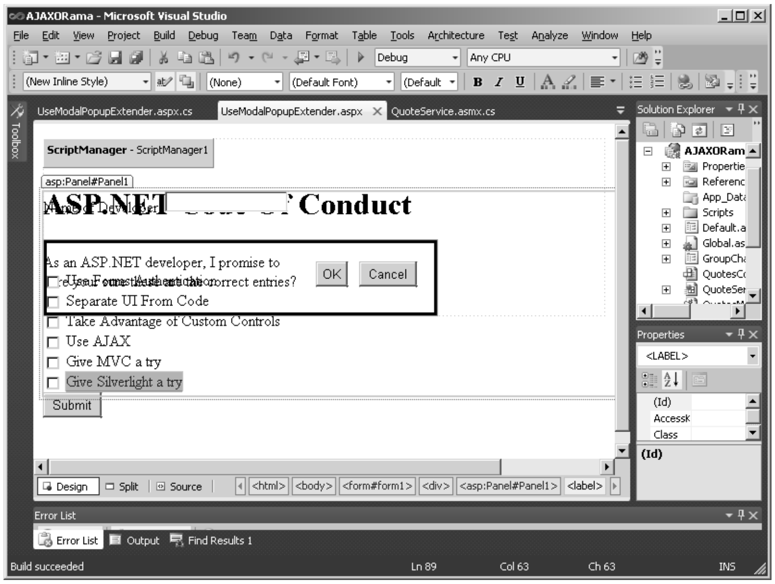
```
<script type="text/javascript">

    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.s 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*:

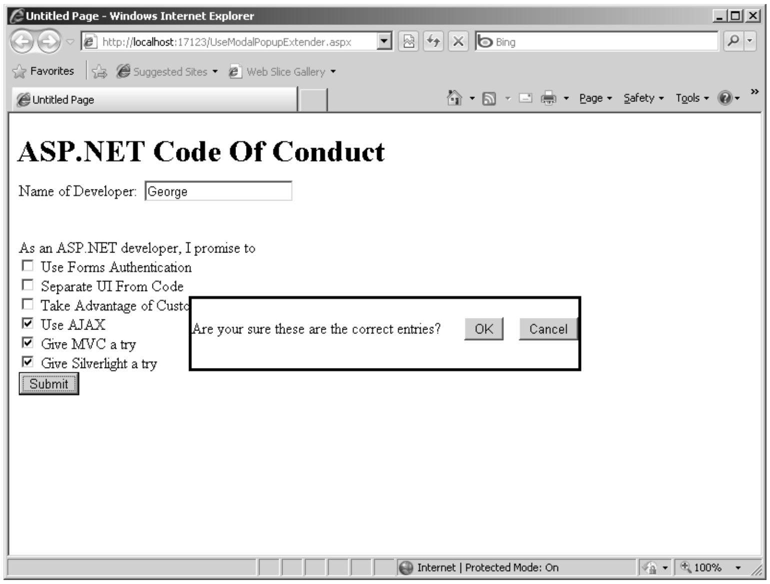
```
<cc1:ModalPopupExtender
    ID="ModalPopupExtender1"
    runat="server"
    OkControlID="ButtonOK"
    CancelControlID="ButtonCancel"
    OnCancelScript="onCancel()"
    OnOkScript="onOk()"
    TargetControlID="ButtonSubmit"
    PopupControlID="PanelModalPopup"
    runat="server"
    DynamicServicePath=""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



Untitled Page - Windows Internet Explorer
http://localhost:17123/UseModalPopupExtender.aspx
Bing
Favorites
Suggested Sites
Web Slice Gallery
Untitled Page
Page
Safety
Tools
Internet | Protected Mode: On
100%

ASP.NET Code Of Conduct

Name of Developer: George

As an ASP.NET developer, I promise to
☐ Use Forms Authentication
☐ Separate UI From Code
☐ Take Advantage of Custo
☑ Use AJAX
☑ Give MVC a try
☑ Give Silverlight a try

Submit

Are your sure these are the correct entries? OK Cancel

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 *ScriptManager* 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 *UpdatePanel* from the toolbox. Controls that you place in the *UpdatePanel* will trigger updates for only that panel, leaving the rest of the page untouched.
Assign arbitrary triggers to an *UpdatePanel* (that is, trigger partial page updates using controls and events not related to the panel)	Modify an *UpdatePanel*'s trigger collection to include the new events and controls. Highlight the *UpdatePanel* from within the Visual Studio designer. Select the *Triggers* property from within the property editor. Assign triggers as appropriate.
Implement regularly timed automatic posts from your page	Use the AJAX *Timer* 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