Professional ASP.NET 4 in C# and VB

Bill Evjen, Scott Hanselman, Devin Rader

PROFESSIONAL ASP.NET 4

PROFESSIONAL

ASP.NET 4

PROFESSIONAL

ASP.NET 4

IN C# AND VB

Bill Evjen
Scott Hanselman
Devin Rader

WILEY
Wiley Publishing, Inc.

Professional ASP.NET 4: In C# and VB

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

ISBN: 978-0-470-50220-4

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2009943645

To Tuija, always.

—Bill Evjen

To Momo and the boys. Toot!

—Scott Hanselman

ABOUT THE AUTHORS

BILL EVJEN is an active proponent of .NET technologies and community-based learning initiatives for .NET. He has been actively involved with .NET since the first bits were released in 2000. In the same year, Bill founded the St. Louis .NET User Group (`www.stlnet.org`), one of the world's first such groups. Bill is also the founder and former executive director of the International .NET Association (`www.ineta.org`), which represents more than 500,000 members worldwide.

Based in St. Louis, Missouri, Bill is an acclaimed author and speaker on ASP.NET and Services. He has authored or coauthored more than 20 books including *Professional C# 2010, Professional VB 2008, ASP.NET Professional Secrets, XML Web Services for ASP.NET, and Web Services Enhancements: Understanding the WSE for Enterprise Applications* (all published by Wiley). In addition to writing, Bill is a speaker at numerous conferences, including DevConnections, VSLive!, and TechEd. Along with these items, Bill works closely with Microsoft as a Microsoft Regional Director and an MVP.

Bill is the Global Head of Platform Architecture for Thomson Reuters, Lipper, the international news and financial services company (`www.thomsonreuters.com`). He graduated from Western Washington University in Bellingham, Washington, with a Russian language degree. When he isn't tinkering on the computer, he can usually be found at his summer house in Toivakka, Finland. You can reach Bill on Twitter at `@billevjen`.

SCOTT HANSELMAN works for Microsoft as a Principal Program Manager Lead in the Server and Tools Online Group, aiming to spread the good word about developing software, most often on the Microsoft stack. Before this, Scott was the Chief Architect at Corillian, an eFinance enabler, for 6+ years, and before Corillian, he was a Principal Consultant at Microsoft Gold Partner for 7 years. He was also involved in a few things like the MVP and RD programs and will speak about computers (and other passions) whenever someone will listen to him. He blogs at `www.hanselman.com`, podcasts at `www.hanselminutes.com`, and runs a team that contributes to `www.asp.net, www.windowsclient.net`, and `www.silverlight.net`. Follow Scott on Twitter `@shanselman`.

DEVIN RADER works at Infragistics where he focuses on delivering great experiences to developers using their controls. He's done work on all of the .NET platforms, but most recently has been focused on Web technologies ASP.NET and Silverlight. As a co-founder of the St. Louis .NET User group and a former INETA board member, and a member of the Central New Jersey .NET user group, he's an active supporter of the .NET developer community. He's also co-author or technical editor of numerous books on .NET, including Wrox's *Silverlight 3 Programmer's Reference.* Follow Devin on Twitter `@devinrader`.

ABOUT THE TECHNICAL EDITORS

CARLOS FIGUEROA has been developing and designing Web solutions for the last 8 years, participating in international projects for the pharmaceutical industry, banking, commercial air transportation, and the government. During these years, Carlos has been deeply involved as an early adopter of Microsoft Web development technologies, such as ASP.NET and Silverlight.

He has been awarded Microsoft Most Valuable Professional for the last 5 years and holds the MCAD certification. Carlos is a Senior Software Developer at Oshyn, Inc. (`www.oshyn.com`), a company specialized on delivering innovative business solutions for the web, mobile devices and emerging technology platforms. At Oshyn, Carlos is dedicated to help some of the most recognizable brands in the world to achieve technology success. You can reach Carlos at `cfigueroa1982@hotmail.com` or follow him on twitter `@carlosfigueroa`.

ANDREW MOORE is a graduate of Purdue University–Calumet in Hammond, Indiana, and has been developing software since 1998 for radar systems, air traffic management, discrete-event simulation, and business communications applications using C, C++, C#, and Java on the Windows, UNIX, and Linux platforms. Andrew is also a contributor to the Wrox Blox article series.

He is currently working as a Senior Software Engineer at Interactive Intelligence, Inc., in Indianapolis, Indiana, developing server-side applications for a multimedia unified business communications platform. Andrew lives in Indiana with his wife Barbara and children Sophia and Andrew.

CREDITS

ACQUISITIONS EDITOR
Paul Reese

SENIOR PROJECT EDITOR
Kevin Kent

TECHNICAL EDITORS
Carlos Figueroa
Andrew Moore

PRODUCTION EDITOR
Daniel Scribner

COPY EDITOR
Paula Lowell

EDITORIAL DIRECTOR
Robyn B. Siesky

EDITORIAL MANAGER
Mary Beth Wakefield

ASSOCIATE DIRECTOR OF MARKETING
David Mayhew

PRODUCTION MANAGER
Tim Tate

VICE PRESIDENT AND EXECUTIVE GROUP PUBLISHER
Richard Swadley

VICE PRESIDENT AND EXECUTIVE PUBLISHER
Barry Pruett

ASSOCIATE PUBLISHER
Jim Minatel

PROJECT COORDINATOR, COVER
Lynsey Stanford

PROOFREADERS
Word One

INDEXER
J & J Indexing

COVER DESIGNER
Michael E. Trent

COVER IMAGE
© Jon Feingersh Photography Inc / Blend Images / Jupiter Images

ACKNOWLEDGMENTS

THANKS TO KEVIN KENT, PAUL REESE, AND JIM MINATEL for the opportunity to work on such a great book. In addition to my co-authors, I would like to thank my family for putting up with all the writing. Thank you Tuija, Sofia, Henri, and Kalle!

—Bill Evjen

CONTENTS

INTRODUCTION

SIMPLY PUT, ASP.NET 4 IS AN AMAZING TECHNOLOGY to use to build your Web solutions! When ASP.NET 1.0 was introduced in 2000, many considered it a revolutionary leap forward in the area of Web application development. ASP.NET 2.0 was just as exciting and revolutionary, and ASP.NET 4 is continuing a forward march in providing the best framework today in building applications for the Web. ASP.NET 4 continues to build on the foundation laid by the release of ASP.NET 1.0/2.0/3.5 by focusing on the area of developer productivity.

This book covers the whole of ASP.NET. It not only introduces new topics, but it also shows you examples of these new technologies in action. So sit back, pull up that keyboard, and enjoy!

A LITTLE BIT OF HISTORY

Before organizations were even thinking about developing applications for the Internet, much of the application development focused on thick desktop applications. These thick-client applications were used for everything from home computing and gaming to office productivity and more. No end was in sight for the popularity of this application model.

During that time, Microsoft developers developed thick-client applications using mainly Visual Basic (VB).

Visual Basic was not only a programming language — it was tied to an IDE that allowed for easy thick-client application development. In the Visual Basic model, developers could drop controls onto a form, set properties for these controls, and provide code behind them to manipulate the events of the control. For example, when an end user clicked a button on one of the Visual Basic forms, the code behind the form handled the event.

Then, in the mid-1990s, the Internet arrived on the scene. Microsoft was unable to move the Visual Basic model to the development of Internet-based applications. The Internet definitely had a lot of power, and right away, the problems facing the thick-client application model were revealed. Internet-based applications created a single instance of the application that everyone could access. Having one instance of an application meant that when the application was upgraded or patched, the changes made to this single instance were immediately available to each and every user visiting the application through a browser.

To participate in the Web application world, Microsoft developed Active Server Pages (ASP). ASP was a quick and easy way to develop Web pages. ASP pages consisted of a single page that contained a mix of markup and languages. The power of ASP was that you could include VBScript or JScript code instructions in the page executed on the Web server before the page was sent to the end user's Web browser. This was an easy way to create dynamic Web pages customized based on instructions dictated by the developer.

ASP used script between brackets and percentage signs `<% %>` to control server-side behaviors. A developer could then build an ASP page by starting with a set of static HTML. Any dynamic element needed by the page was defined using a scripting language (such as VBScript or JScript). When a user requested the page from the server by using a browser, the `asp.dll` (an ISAPI application that provided a bridge between the scripting language and the Web server) would take hold of the page and define all the dynamic aspects of the page on-the-fly based on the programming logic specified in the script. After all the dynamic aspects of the page were defined, the result was an HTML page output to the browser of the requesting client.

As the Web application model developed, more and more languages mixed in with the static HTML to help manipulate the behavior and look of the output page. Over time, such a large number of languages, scripts, and plain text could be placed in a typical ASP page that developers began to refer to pages that used these features as *spaghetti code*. For example, having a page that used HTML, VBScript, JavaScript, Cascading Style Sheets, T-SQL, and more was quite possible. In certain instances, these pages became a manageability nightmare.

ASP evolved and new versions were released. ASP 2.0 and 3.0 were popular because the technology made creating Web pages relatively straightforward and easy. Their popularity was enhanced because they appeared in the late 1990s, just as the dotcom era was born. During this time, a mountain of new Web pages and portals were developed, and ASP was one of the leading technologies individuals and companies used to build them. Even today, you can still find a lot of `.asp` pages on the Internet — including some of Microsoft's own Web pages.

However, even at the time of the final release of Active Server Pages in late 1998, Microsoft employees Marc Anders and Scott Guthrie had other ideas. Their ideas generated what they called XSP (an abbreviation with no meaning) — a new way of creating Web applications in an object-oriented manner instead of in the procedural manner of ASP 3.0. They showed their idea to many different groups within Microsoft, and they were well received. In the summer of 2000, the beta of what was then called ASP+ was released at Microsoft's Professional Developers Conference. The attendees eagerly started working with it. When the technology became available (with the final release of the .NET Framework 1.0), it was renamed ASP.NET — receiving the .NET moniker that most of Microsoft's new products were receiving at that time.

Before the introduction of .NET, the model that classic ASP provided and what developed in Visual Basic were so different that few VB developers also developed Web applications, and few Web application developers also developed the thick-client applications of the VB world. There was a great divide. ASP.NET bridged this gap. ASP.NET brought a Visual Basic–style eventing model to Web application development, providing much-needed state management techniques over stateless HTTP. Its model is much like the earlier Visual Basic model in that a developer can drag and drop a control onto a design surface or form, manipulate the control's properties, and even work with the code behind these controls to act on certain events that occur during their lifecycles. What ASP.NET created is really the best of both models, as you will see throughout this book.

I know you will enjoy working with this latest release of ASP.NET 4. Nothing is better than getting your hands on a new technology and seeing what is possible. The following section discusses the goals of ASP.NET so that you can find out what to expect from this new offering!

THE GOALS OF ASP.NET

ASP.NET 4 is another major release of the product and builds on the previous releases with additional classes and capabilities. This release of the Framework and Visual Studio was code-named *Hawaii* internally at Microsoft. ASP.NET 4 continues on a path to make ASP.NET developers the most productive developers in the Web space. This book also focuses on the new additions to ASP.NET 4 and the .NET Framework 4 with the release of ASP.NET 4.

Ever since the release of ASP.NET 2.0, the Microsoft team has focused its goals on developer productivity, administration, and management, as well as performance and scalability.

Developer Productivity

Much of the focus of ASP.NET 4 is on productivity. Huge productivity gains were made with the release of ASP.NET 1.*x* and 2.0; could it be possible to expand further on those gains?

One goal the development team had for ASP.NET was to eliminate much of the tedious coding that ASP.NET originally required and to make common ASP.NET tasks easier. The developer productivity capabilities are presented throughout this book. Before venturing into these capabilities, this introduction looks at the older ASP.NET 1.0 technology to make a comparison to ASP.NET 4. Listing I-1 provides an example of using ASP.NET 1.0 to build a table in a Web page that includes the capability to perform simple paging of the data provided.

LISTING I-1: Showing data in a DataGrid server control with paging enabled (VB only)

```
<%@ Page Language="VB" AutoEventWireup="True" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<script runat="server">

    Private Sub Page_Load(ByVal sender As System.Object, _
      ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            BindData()
        End If
    End Sub

    Private Sub BindData()
        Dim conn As SqlConnection = New _
           SqlConnection("server='localhost';
           trusted_connection=true; Database='Northwind'")
        Dim cmd As SqlCommand = _
           New SqlCommand("Select * From Customers", conn)
        conn.Open()

        Dim da As SqlDataAdapter = New SqlDataAdapter(cmd)
        Dim ds As New DataSet

        da.Fill(ds, "Customers")

        DataGrid1.DataSource = ds
        DataGrid1.DataBind()
    End Sub

    Private Sub DataGrid1_PageIndexChanged(ByVal source As Object, _
      ByVal e As _
       System.Web.UI.WebControls.DataGridPageChangedEventArgs)
        DataGrid1.CurrentPageIndex = e.NewPageIndex
        BindData()
    End Sub

</script>
<html>
<head>
</head>
<body>
    <form runat="server">
        <asp:DataGrid id="DataGrid1" runat="server"
         AllowPaging="True"
         OnPageIndexChanged="DataGrid1_PageIndexChanged">
        </asp:DataGrid>
    </form>
</body>
</html>
```

Although quite a bit of code is used here, this is a dramatic improvement over the amount of code required to accomplish this task using classic Active Server Pages 3.0. We will not go into the details of this older code; it just demonstrates that to add any additional common functionality (such as paging) for the data shown in a table, the developer had to create custom code.

This is one area where the developer productivity gains are most evident. ASP.NET 4 provides a control called the GridView server control. This control is much like the DataGrid server control, but the GridView server control (besides offering many additional features) contains the built-in capability to apply paging, sorting, and editing of data with relatively little work on your part. Listing I-2 shows an example of the GridView server control. This example builds a table of data from the Customers table in the Northwind database that includes paging.

LISTING I-2: Viewing a paged dataset with the GridView server control

```
<%@ Page Language="VB" %>

<script runat="server">

</script>

<html xmlns=http://www.w3.org/1999/xhtml>
<head runat="server">
    <title>GridView Demo</title>
</head>
<body>
    <form runat="server">
        <asp:GridView ID="GridView1" Runat="server"
         AllowPaging="True"
         DataSourceId="Sqldatasource1" />
        <asp:SqlDataSource ID="SqlDataSource1" Runat="server"
         SelectCommand="Select * From Customers"
         ProviderName="System.Data.OleDb"
         ConnectionString="Provider=SQLOLEDB;Server=localhost;uid=sa;
         pwd=password;database=Northwind" />
    </form>
</body>
</html>
```

That's it! You can apply paging by using a couple of server controls. You turn on this capability using a server control attribute, the `AllowPaging` attribute of the GridView control:

```
<asp:GridView ID="GridView1" Runat="server" AllowPaging="True" DataSourceId="SqlDataSource1" />
```

The other interesting event occurs in the code section of the document:

```
<script runat="server"></script>
```

These two lines of code are not actually needed to run the file. They are included here to make a point — *you don't need to write any server-side code to make this all work!* You need to include only some server controls: one control to get the data and one control to display the data. Then the controls are wired together.

Performance and Scalability

One of the goals for ASP.NET that was set by the Microsoft team was to provide the world's fastest Web application server. This book also addresses a number of performance tactics available in ASP.NET 4.

One of the most exciting performance capabilities is the caching capability aimed at exploiting Microsoft's SQL Server. ASP.NET 4 includes a feature called *SQL cache invalidation*. Before ASP.NET 2.0, caching the results that came from SQL Server and updating the cache based on a time interval was possible — for example, every 15 seconds or so. This meant that the end user might see stale data if the result set changed sometime during that 15-second period.

In some cases, this time interval result set is unacceptable. In an ideal situation, the result set stored in the cache is destroyed if any underlying change occurs in the source from which the result set is retrieved — in this case, SQL Server. With ASP.NET 4, you can make this happen with the use of SQL cache invalidation. This means that when the result set from SQL Server changes, the output cache is triggered to change, and the end user always sees the latest result set. The data presented is never stale.

ASP.NET 4 provides 64-bit support. This means that you can run your ASP.NET applications on 64-bit Intel or AMD processors.

Because ASP.NET 4 is fully backward compatible with ASP.NET 1.0, 1.1, 2.0, and 3.5, you can now take any former ASP.NET application, recompile the application on the .NET Framework 4, and run it on a 64-bit processor.

ADDITIONAL FEATURES OF ASP.NET 4

You just learned some of the main goals of the ASP.NET team that built ASP.NET. To achieve these goals, ASP.NET provides a mountain of features to make your development process easier. A few of these features are described in the following sections.

ASP.NET Developer Infrastructures

An exciting aspect of ASP.NET is that infrastructures are in place for you to use in your applications. The ASP.NET team selected some of the most common programming operations performed with Web applications to be built directly into ASP.NET. This saves you considerable time and coding.

Membership and Role Management

Prior to ASP.NET 2.0, if you were developing a portal that required users to log in to the application to gain privileged access, invariably you had to create it yourself. Creating applications with areas that are accessible only to select individuals can be tricky.

You will find that with ASP.NET 4 this capability is built in. You can validate users as shown in Listing I-3.

LISTING I-3: Validating a user in code

VB
```
If (Membership.ValidateUser (Username.Text, Password.Text)) Then
   ' Allow access code here
End If
```

C#
```
if (Membership.ValidateUser (Username.Text, Password.Text)) {
   // Allow access code here
}
```

A series of APIs, controls, and providers in ASP.NET 4 enable you to control an application's user membership and role management. Using these APIs, you can easily manage users and their complex roles — creating, deleting, and editing them. You get all this capability by using the APIs or a built-in Web tool called the Web Site Administration Tool.

As far as storing users and their roles, ASP.NET 4 uses an `.mdf` file (the file type for the SQL Server Express Edition) for storing all users and roles. You are in no way limited to just this data store, however. You can expand everything offered to you by ASP.NET and build your own providers using whatever you fancy as a data store. For example, if you want to build your user store in LDAP or within an Oracle database, you can do so quite easily.

Personalization

One advanced feature that portals love to offer their membership base is the capability to personalize their offerings so that end users can make the site look and function however they want. The capability to personalize an application and store the personalization settings is completely built into the ASP.NET Framework.

Because personalization usually revolves around a user and possibly a role that this user participates in, the personalization architecture can be closely tied to the membership and role infrastructures. You have a couple of options for storing the created personalization settings. The capability to store these settings in either Microsoft Access or in SQL Server is built into ASP.NET 4. As with the capabilities of the membership and role APIs, you can use the flexible provider model, and then either change how the built-in provider uses the available data store or build your own custom data provider to work with a completely new data store. The personalization API also supports a union of data stores, meaning that you can use more than one data store if you want.

Because creating a site for customization using these APIs is so easy, this feature is quite a value-add for any application you build.

The ASP.NET Portal Framework

During the days of ASP.NET 1.0, developers could go to the ASP.NET team's site (found at `asp.net`) and download some Web application demos such as IBuySpy. These demos are known as Developer Solution Kits and are used as the basis for many of the Web sites on the Internet today. Some were even extended into open source frameworks such as DotNetNuke.

The nice thing about some of these frameworks was that you could use the code they provided as a basis to build either a Web store or a portal. You simply took the base code as a starting point and extended it. For example, you could change the look and feel of the presentation part of the code or introduce advanced functionality into its modular architecture. Developer Solution Kits are quite popular because they make performing these types of operations so easy.

Because of the popularity of frameworks, ASP.NET 4 offers built-in capability for using Web Parts to easily build portals. The possibilities for what you can build using the Portal Framework is astounding. The power of building and using Web Parts is that it easily enables end users to completely customize the portal for their own preferences.

Site Navigation

The ASP.NET team members realize that end users want to navigate through applications with ease. The mechanics to make this work in a logical manner are sometimes hard to code. The team solved the problem in ASP.NET with a series of navigation-based server controls.

For example, you can build a site map for your application in an XML file that specific controls can inherently work from. Listing I-4 shows a sample site map file.

LISTING I-4: An example of a site map file

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

<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
    <siteMapNode title="Home" description="Home Page"
     url="default.aspx">
       <siteMapNode title="News" description="The Latest News"
        url="News.aspx">
          <siteMapNode title="U.S." description="U.S. News"
           url="News.aspx?cat=us" />
```

```
                <siteMapNode title="World" description="World News"
                 url="News.aspx?cat=world" />
                <siteMapNode title="Technology"
                 description="Technology News"
                 url="News.aspx?cat=tech" />
                <siteMapNode title="Sports" description="Sports News"
                 url="News.aspx?cat=sport" />
             </siteMapNode>
             <siteMapNode title="Finance"
              description="The Latest Financial Information"
               url="Finance.aspx">
                <siteMapNode title="Quotes"
                 description="Get the Latest Quotes"
                 url="Quotes.aspx" />
                <siteMapNode title="Markets"
                 description="The Latest Market Information"
                 url="Markets.aspx">
                   <siteMapNode title="U.S. Market Report"
                    description="Looking at the U.S. Market"
                    url="MarketsUS.aspx" />
                   <siteMapNode title="NYSE"
                    description="The New York Stock Exchange"
                    url="NYSE.aspx" />
                </siteMapNode>
                <siteMapNode title="Funds" description="Mutual Funds"
                 url="Funds.aspx" />
             </siteMapNode>
             <siteMapNode title="Weather" description="The Latest Weather"
              url="Weather.aspx" />
          </siteMapNode>
       </siteMap>
```

After you have a site map in place, you can use this file as the data source behind a couple of site navigation server controls, such as the TreeView and the SiteMapPath server controls. The TreeView server control enables you to place an expandable site navigation system in your application. Figure I-1 shows you an example of one of the many looks you can give the TreeView server control.

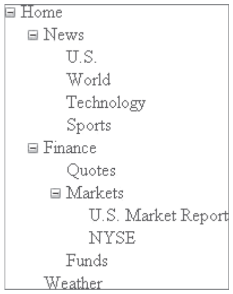

⊟ Home
⊟ News
U.S.
World
Technology
Sports
⊟ Finance
Quotes
⊟ Markets
U.S. Market Report
NYSE
Funds
Weather

FIGURE I-1

SiteMapPath is a control that provides the capability to place what some call *breadcrumb navigation* in your application so that the end user can see the path that he has taken in the application and can easily navigate to higher levels in the tree. Figure I-2 shows you an example of the SiteMapPath server control at work.

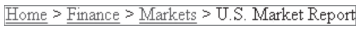

Home > Finance > Markets > U.S. Market Report

FIGURE I-2

These site navigation capabilities provide a great way to get programmatic access to the site layout and even to take into account things like end-user roles to determine which parts of the site to show.

The ADO.NET Entity Framework

Most developers need to work with an underlying database of some kind. Whether that is a Microsoft SQL Server database or an Oracle database, your applications are usually pulling content of some kind to work with. The difficulty in working with an underlying database is that a database and your object-oriented code handle objects in such dramatically different ways.

In the database world, your data structures are represented in tables, and collections within items (such as a `Customer` object with associated `Orders`) are simply represented as two tables with a `Join` statement required between them. In contrast, in your object-oriented code, these objects are represented so that the

`Orders` item is simply a property within the `Customers` object. Bringing these two worlds together and mapping these differences have always been a bit laborious.

ASP.NET 4 includes the ability to work with the ADO.NET Entity Framework, which you will find is somewhat similar to working with LINQ to SQL. The purpose of the ADO.NET Entity Framework is to allow you to create an Entity Data Model (EDM) that will make mapping the object-oriented objects that you create along with how these objects are represented in the database easy.

One advantage of the ADO.NET Entity Framework is that it works with many different types of databases, so you will not be limited to working with a single database as you are with LINQ to SQL. Another advantage is that the ADO.NET Entity Framework is the basis of some other exciting technologies that ASP.NET 4 includes, such as ADO.NET Data Services.

ASP.NET Dynamic Data

Another great ASP.NET feature is called ASP.NET Dynamic Data. This capability enables you to easily create a reporting and data entry application from a database in just a couple of minutes.

Working with ASP.NET Dynamic Data is as simple as pointing to an Entity Data Model that you created in your application and allowing the dynamic data engine to create the Web pages for you that provide you with full create, edit, update, and delete capabilities over the database.

ASP.NET Dynamic Data requires that you have an Entity Data Model in place for it to work. The nice thing is that you are not limited to working with just the ADO.NET Entity Framework — you can also work with any LINQ to SQL models that you have created.

One great feature of the architecture of ASP.NET Dynamic Data is that it is based on working with templates in the dynamic generation of the pages for the site. As a developer working with this system, you are able to use the system "as-is" or even take pieces of it and incorporate its abilities in any of your pre-existing ASP.NET applications.

WCF Data Services

ASP.NET 4 also includes another great feature called WCF Data Services. Formally known as ADO.NET Data Services, WCF Data Services enables you to create a RESTful service interface against your database.

Using WCF Data Services, you can provide the capability to use the URL of the request as a command-driven URI along with HTTP verbs to direct the server on how you want to deal with the underlying data. You can create, read, update, or delete underlying database data using this technology, but as the implementer of the interface, you are also just as able to limit and restrict end user capability and access.

The ASP.NET Compilation System

Compilation in ASP.NET 1.0 was always a tricky scenario. With ASP.NET 1.0, you could build an application's code-behind files using ASP.NET and Visual Studio, deploy it, and then watch as the `.aspx` files were compiled page by page as each page was requested. If you made any changes to the code-behind file in ASP.NET 1.0, it was not reflected in your application until the entire application was rebuilt. That meant that the same page-by-page request had to be done again before the entire application was recompiled.

Everything about how ASP.NET 1.0 worked with classes and compilation is different from how it is in ASP.NET today. The mechanics of the compilation system actually begin with how a page is structured in ASP.NET 4. In ASP.NET 1.0, you constructed your pages either by using the code-behind model or by placing all the server code inline between `<script>` tags on your `.aspx` page. Most pages were constructed using the code-behind model because this was the default when using Visual Studio .NET 2002 or 2003. Creating your page using the inline style in these IDEs was quite difficult. If you did, you were deprived of the use of IntelliSense, which can be quite the lifesaver when working with the tremendously large collection of classes that the .NET Framework offers.

ASP.NET 4 offers a different code-behind model from the 1.0/1.1 days because the .NET Framework 4 has the capability to work with *partial classes* (also called partial types). Upon compilation, the separate files are combined into a single offering. This gives you much cleaner code-behind pages. The code that was part of the `Web Form Designer Generated` section of your classes is separated from the code-behind classes that you create yourself. Contrast this with the ASP.NET 1.0 `.aspx` file's need to derive from its own code-behind file to represent a single logical page.

ASP.NET 4 applications can include a App_Code directory where you place your class's source. Any class placed here is dynamically compiled and reflected in the application. You do not use a separate build process when you make changes as you did with ASP.NET 1.0. This is a *just save and hit* deployment model like the one in classic ASP 3.0. Visual Studio 2010 also automatically provides IntelliSense for any objects that are placed in the App_Code directory, whether you are working with the code-behind model or are coding inline.

ASP.NET 4 also provides you with tools that enable you to pre-compile your ASP.NET applications — both `.aspx` pages and code behind — so that no page within your application has latency when it is retrieved for the first time. Doing this is also a great way to discover any errors in the pages without invoking every page. Precompiling your ASP.NET 2.0 (as well as 3.5 or 4) applications is as simple as using `aspnet_compiler.exe` and employing some of the available flags. As you pre-compile your entire application, you also receive error notifications if any errors are found anywhere within it. Pre-compilation also enables you to deliver only the created assembly to the deployment server, thereby protecting your code from snooping, unwanted changes, and tampering after deployment. You will see examples of these scenarios later in this book.

Health Monitoring for Your ASP.NET Applications

The built-in health monitoring capabilities are rather significant features designed to make managing a deployed ASP.NET application easier. Health monitoring provides what the term implies — the capability to monitor the health and performance of your deployed ASP.NET applications.

Using the health monitoring system enables you to perform event logging for health monitoring events, which are called *Web events*, such as failed logins, application starts and stops, or any unhandled exceptions. The event logging can occur in more than one place; therefore, you can log to the event log or even back to a database. In addition to performing this disk-based logging, you can also use the system to e-mail health monitoring information.

Besides working with specific events in your application, you can also use the health monitoring system to take health snapshots of a running application. As you can with most systems that are built into ASP.NET 4, you can extend the health monitoring system and create your own events for recording application information.

Health monitoring is already enabled by default in the system `.config` files. The default setup for health monitoring logs all errors and failure audits to the event log. For instance, throwing an error in your application results in an error notification in the Application log.

You can change the default event logging behaviors simply by making some minor changes to your application's `web.config` file. For instance, suppose that you want to store this error event information in a SQL Express file contained within the application. You can make this change by adding a `<healthMonitoring>` node to your `web.config` file as presented in Listing I-5.

LISTING I-5: Defining health monitoring in the web.config file

```
  <healthMonitoring enabled="true">
    <providers>
       <clear />
       <add name="SqlWebEventProvider"
        connectionStringName="LocalSqlServer"
        maxEventDetailsLength="1073741823" buffer="false"
        bufferMode="Notification"
        type="System.Web.Management.SqlWebEventProvider,
           System.Web,Version=4.0.0.0,Culture=neutral,
           PublicKeyToken=b03f5f7f11d50a3a"/>
    </providers>
    <rules>
       <clear />
       <add name="All Errors Default" eventName="All Errors"
        provider="SqlWebEventProvider"
        profile="Default" minInstances="1" maxLimit="Infinite"
        minInterval="00:01:00" custom="" />
       <add name="Failure Audits Default" eventName="Failure Audits"
        provider="SqlWebEventProvider" profile="Default"
        minInstances="1"
        maxLimit="Infinite" minInterval="00:01:00" custom="" />
    </rules>
</healthMonitoring>
```

After this change, events are logged in the `ASPNETDB.MDF` file that is automatically created on your behalf if it does not already exist in your project.

Opening this SQL Express file, you will find an `aspnet_WebEvent_Events` table where all this information is stored.

You will learn much more about the health monitoring capabilities provided with ASP.NET 4 in Chapter 34.

Reading and Writing Configuration Settings

Using the `WebConfigurationManager` class, you have the capability to read and write to the server or application configuration files. This means that you can write and read settings in the `machine.config` or the `web.config` files that your application uses.

The capability to read and write to configuration files is not limited to working with the local machine in which your application resides. You can also perform these operations on remote servers and applications.

Of course, a GUI-based way exists in which you can perform these read or change operations on the configuration files at your disposal. The exciting thing, however, is that the built-in GUI tools that provide this functionality (such as the ASP.NET MMC snap-in when using Windows XP or the latest IIS interface if you are using Windows 7) use the `WebConfigurationManager` class, which is also available for building custom administration tools.

Listing I-6 shows an example of reading a connection string from an application's `web.config` file.

LISTING I-6: Reading a connection string from the application's web.config file

VB

```
Protected Sub Page_Load(ByVal sender As Object,
 ByVal e As System.EventArgs)
   Try
      Dim connectionString As String =
         ConfigurationManager.ConnectionStrings("Northwind").
            ConnectionString.ToString()
```

```
        Label1.Text = connectionString
    Catch ex As Exception
        Label1.Text = "No connection string found."
    End Try
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    try
    {
        string connectionString =
            ConfigurationManager.ConnectionStrings["Northwind"].
                ConnectionString.ToString();
        Label1.Text = connectionString;
    }
    catch (Exception)
    {
        Label1.Text = "No connection string found.";
    }
}
```

This little bit of code writes the Northwind connection string found in the `web.config` file to the screen using a Label control. As you can see, grabbing items from the configuration file is rather simple.

Localization

ASP.NET is making localizing applications easier than ever. In addition to using Visual Studio, you can create resource files (`.resx`) that allow you to dynamically change the pages you create based on the culture settings of the requestor.

ASP.NET 4 provides the capability to provide resources application-wide or just to particular pages in your application through the use of two application folders — App_GlobalResources and App_LocalResources.

The items defined in any `.resx` files you create are then accessible directly in the ASP.NET server controls or programmatically using expressions such as

```
<%= Resources.Resource.Question %>
```

This system is straightforward and simple to implement. Chapter 32 covers this topic in greater detail.

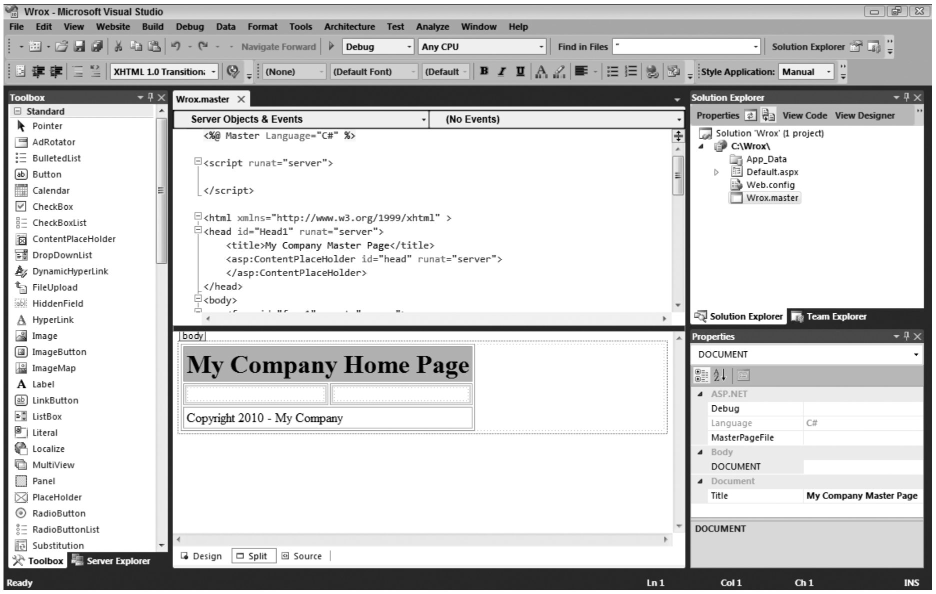
Expanding on the Page Framework

ASP.NET pages can be built based on visual inheritance. This was possible in the Windows Forms world, but it is also possible with ASP.NET. You also gain the capability to easily apply a consistent look and feel to the pages of your application by using themes. Many of the difficulties in working with ADO.NET are made easier through a series of data source controls that take care of accessing and retrieving data from a large collection of data stores.

Master Pages

With the capability of *master pages* in ASP.NET, you can use visual inheritance within your ASP.NET applications. Because many ASP.NET applications have a similar structure throughout their pages, building a page template once and using that same template throughout the application is logical.

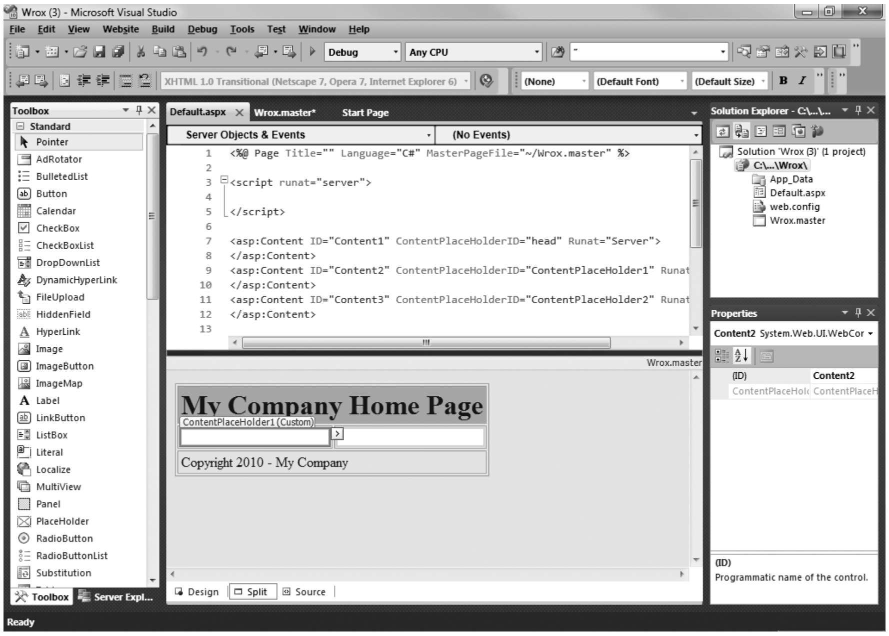
In ASP.NET, you do this by creating a `.master` page, as shown in Figure I-3.



```
<%@ Master Language="C#" %>

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>My Company Master Page</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
```

FIGURE I-3

An example master page might include a header, footer, and any other elements that all the pages can share. Besides these core elements, which you might want on every page that inherits and uses this template, you can place `<asp:ContentPlaceHolder>` server controls within the master page itself for the subpages (or content pages) to use to change specific regions of the master page template. The editing of the subpage is shown in Figure I-4.



```
<%@ Page Title="" Language="C#" MasterPageFile="~/Wrox.master" %>

<script runat="server">

</script>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat
</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolder2" Runat
</asp:Content>
```

FIGURE I-4

When an end user invokes one of the subpages, she is actually looking at a single page compiled from both the subpage and the master page that the particular subpage inherited from. This also means that the server and client code from both pages are enabled on the new single page.

The nice thing about master pages is that you have a single place to make any changes that affect the entire site. This eliminates making changes to each and every page within an application.

Themes

The inclusion of themes in ASP.NET has made providing a consistent look and feel across your entire site quite simple. Themes are simple text files where you define the appearance of server controls that can be applied across the site, to a single page, or to a specific server control. You can also easily incorporate graphics and Cascading Style Sheets (CSS), in addition to server control definitions.

Themes are stored in the App_Theme directory within the application root for use within that particular application. One cool capability of themes is that you can dynamically apply them based on settings that use the personalization service provided by ASP.NET. Each unique user of your portal or application can have her own personalized look and feel that she has chosen from your offerings.

Objects for Accessing Data

One of the more code-intensive tasks in ASP.NET 1.0 was the retrieval of data. In many cases, this meant working with a number of objects. If you have been working with ASP.NET for a while, then you know that it was an involved process to display data from a Microsoft SQL Server table within a DataGrid server control. For instance, you first had to create a number of new objects. They included a `SqlConnection` object followed by a `SqlCommand` object. When those objects were in place, you then created a `SqlDataReader` to populate your DataGrid by binding the result to the DataGrid. In the end, a table appeared containing the contents of the data you were retrieving (such as the Customers table from the Northwind database).

Today, ASP.NET eliminates this intensive procedure with the introduction of a set of objects that work specifically with data access and retrieval. These data controls are so easy to use that you access and retrieve data to populate your ASP.NET server controls without writing any code. You saw an example of this in Listing I-2, where an `<asp:SqlDataSource>` server control retrieved rows of data from the Customers table in the Northwind database from SQL Server. This SqlDataSource server control was then bound to the GridView server control via the use of simple attributes within the GridView control itself. It really could not be any easier!

The great news about this functionality is that it is not limited to just Microsoft's SQL Server. In fact, several data source server controls are at your disposal. You also have the capability to create your own. In addition to the SqlDataSource server control, ASP.NET 4 includes the AccessDataSource, XmlDataSource, ObjectDataSource, SiteMapDataSource, and LinqDataSource server controls. You will use all these data controls later in this book.

WHAT YOU NEED FOR ASP.NET 4

You might find that installing Visual Studio 2010 is best to work through the examples in this book; you can, however, just use Microsoft's Notepad and the command-line compilers that come with the .NET Framework 4. To work through *every* example in this book, you need the following:

- Windows Server 2003, Windows Server 2008, Windows 2000, Windows XP, Windows Vista, or Windows 7
- Visual Studio 2010 (this will install the .NET Framework 4)
- SQL Server 2000, 2005, or 2008
- Microsoft Access or SQL Server Express Edition

The nice thing is that you are not required to have Microsoft Internet Information Services (IIS) to work with ASP.NET 4 because ASP.NET includes a built-in Web server based on the previously released Microsoft Cassini technology. Moreover, if you do not have a full-blown version of SQL Server, don't be alarmed. Many examples that use this database can be altered to work with Microsoft's SQL Server Express Edition, which you will find free on the Internet.

WHO SHOULD READ THIS BOOK?

This book was written to introduce you to the features and capabilities that ASP.NET 4 offers, as well as to give you an explanation of the foundation that ASP.NET provides. We assume you have a general understanding of Web technologies, such as previous versions of ASP.NET, Active Server Pages 2.0/3.0, or JavaServer Pages. If you understand the basics of Web programming, you should not have much trouble following along with this book's content.

If you are brand new to ASP.NET, be sure to check out *Beginning ASP.NET 4: In C# and VB* by Imar Spaanjaars (Wiley Publishing, Inc., 2010) to help you understand the basics.

In addition to working with Web technologies, we also assume that you understand basic programming constructs, such as variables, `For Each` loops, and object-oriented programming.

You may also be wondering whether this book is for the Visual Basic developer or the C# developer. We are happy to say that it is for both! When the code differs substantially, this book provides examples in both VB and C#.

WHAT THIS BOOK COVERS

This book explores the release of ASP.NET 4. It covers each major new feature included in ASP.NET 4 in detail. The following list tells you something about the content of each chapter.

- **Chapter 1, "Application and Page Frameworks."** The first chapter covers the frameworks of ASP.NET applications as well as the structure and frameworks provided for single ASP.NET pages. This chapter shows you how to build ASP.NET applications using IIS or the built-in Web server that comes with Visual Studio 2010. This chapter also shows you the folders and files that are part of ASP.NET. It discusses ways to compile code and shows you how to perform cross-page posting. This chapter ends by showing you easy ways to deal with your classes from within Visual Studio 2010.
- **Chapters 2, 3, and 4.** These three chapters are grouped together because they all deal with server controls. This batch of chapters starts by examining the idea of the server control and its pivotal role in ASP.NET development. In addition to looking at the server control framework, these chapters delve into the plethora of server controls that are at your disposal for ASP.NET development projects. Chapter 2, "ASP.NET Server Controls and Client-Side Scripts," looks at the basics of working with server controls. Chapter 3, "ASP.NET Web Server Controls," covers the controls that have been part of the ASP.NET technology since its initial release and the controls that have been added in each of the ASP.NET releases. Chapter 4, "Validation Server Controls," describes a special group of server controls: those for validation. You can use these controls to create beginning-to-advanced form validations.
- **Chapter 5, "Working with Master Pages."** Master pages are a great capability of ASP.NET. They provide a means of creating templated pages that enable you to work with the entire application, as opposed to single pages. This chapter examines the creation of these templates and how to apply them to your content pages throughout an ASP.NET application.
- **Chapter 6, "Themes and Skins."** The Cascading Style Sheet files you are allowed to use in ASP.NET 1.0/1.1 are simply not adequate in many regards, especially in the area of server controls. When using these early versions, the developer can never be sure of the HTML output these files might generate.

This chapter looks at how to deal with the styles that your applications require and shows you how to create a centrally managed look-and-feel for all the pages of your application by using themes and the skin files that are part of a theme.

- **Chapter 7, "Data Binding."** One of the more important tasks of ASP.NET is presenting data, and this chapter shows you how to do that. ASP.NET provides a number of controls to which you can attach data and present it to the end user. This chapter looks at the underlying capabilities that enable you to work with the data programmatically before issuing the data to a control.
- **Chapter 8, "Data Management with ADO.NET."** This chapter presents the ADO.NET data model provided by ASP.NET, which allows you to handle the retrieval, updating, and deleting of data quickly and logically. This data model enables you to use one or two lines of code to get at data stored in everything from SQL Server to XML files.
- **Chapter 9, "Querying with LINQ."** The .NET Framework 4 includes a nice access model language called LINQ. LINQ is a set of extensions to the .NET Framework that encompass language-integrated query, set, and transform operations. This chapter introduces you to LINQ and how to effectively use this feature in your Web applications today.
- **Chapter 10, "Working with XML and LINQ to XML."** Without a doubt, XML has become one of the leading technologies used for data representation. For this reason, the .NET Framework and ASP.NET 4 have many capabilities built into their frameworks that enable you to easily extract, create, manipulate, and store XML. This chapter takes a close look at the XML technologies built into ASP.NET and the underlying .NET Framework.
- **Chapter 11, "Introduction to the Provider Model."** A number of systems are built into ASP.NET that make the lives of developers so much easier and more productive than ever before. These systems are built on an architecture called a *provider model*, which is rather extensible. This chapter gives an overview of this provider model and how it is used throughout ASP.NET 4.
- **Chapter 12, "Extending the Provider Model."** After an introduction of the provider model, this chapter looks at some of the ways to extend the provider model found in ASP.NET 4. This chapter also reviews a couple of sample extensions to the provider model.
- **Chapter 13, "Site Navigation."** It is quite apparent that many developers do not simply develop single pages — they build applications. Therefore, they need mechanics that deal with functionality throughout the entire application, not just the pages. One of the application capabilities provided by ASP.NET 4 is the site navigation system covered in this chapter. The underlying navigation system enables you to define your application's navigation structure through an XML file, and it introduces a whole series of navigation server controls that work with the data from these XML files.
- **Chapter 14, "Personalization."** Developers are always looking for ways to store information pertinent to the end user. After it is stored, this personalization data has to be persisted for future visits or for grabbing other pages within the same application. The ASP.NET team developed a way to store this information — the ASP.NET personalization system. The great thing about this system is that you configure the entire behavior of the system from the `web.config` file.
- **Chapter 15, "Membership and Role Management."** This chapter covers the membership and role management system developed to simplify adding authentication and authorization to your ASP.NET applications. These two systems are extensive; they make some of the more complicated authentication and authorization implementations of the past a distant memory. This chapter focuses on using the `web.config` file for controlling how these systems are applied, as well as on the server controls that work with the underlying systems.
- **Chapter 16, "Portal Frameworks and Web Parts."** This chapter explains Web Parts — a way of encapsulating pages into smaller and more manageable objects. The great thing about Web Parts is that they can be made of a larger Portal Framework, which can then enable end users to completely modify how the Web Parts are constructed on the page — including their appearance and layout.

- **Chapter 17, "HTML and CSS Design with ASP.NET."** Visual Studio 2010 places a lot of focus on building a CSS-based Web. This chapter takes a close look at how you can effectively work with HTML and CSS design for your ASP.NET applications.
- **Chapter 18, "ASP.NET AJAX."** AJAX is a hot buzzword in the Web application world these days. AJAX is an acronym for *Asynchronous JavaScript and XML*. In Web application development, it signifies the capability to build applications that make use of the `XMLHttpRequest` object. Visual Studio 2010 contains the ability to build AJAX-enabled ASP.NET applications from the default install of the IDE. This chapter takes a look at this way to build your applications.
- **Chapter 19, "ASP.NET AJAX Control Toolkit."** Along with the capabilities to build ASP.NET applications that make use of the AJAX technology, a series of controls is available to make the task rather simple. This chapter takes a good look at the ASP.NET AJAX Control Toolkit and how to use this toolkit with your applications today.
- **Chapter 20, "Security."** This chapter discusses security beyond the membership and role management features provided by ASP.NET 4. This chapter provides an in-depth look at the authentication and authorization mechanics inherent in the ASP.NET technology, as well as HTTP access types and impersonations.
- **Chapter 21, "State Management."** Because ASP.NET is a request-response–based technology, state management and the performance of requests and responses take on significant importance. This chapter introduces these two separate but important areas of ASP.NET development.
- **Chapter 22, "Caching."** Because of the request-response nature of ASP.NET, caching (storing previously generated results, images, and pages) on the server becomes rather important to the performance of your ASP.NET applications. This chapter looks at some of the advanced caching capabilities provided by ASP.NET, including the SQL cache invalidation feature which is part of ASP.NET 4. This chapter also takes a look at object caching and object caching extensibility.
- **Chapter 23, "Debugging and Error Handling."** Being able to handle unanticipated errors in your ASP.NET applications is vital for any application that you build. This chapter tells you how to properly structure error handling within your applications. It also shows you how to use various debugging techniques to find errors that your applications might contain.
- **Chapter 24, "File I/O and Streams."** More often than not, you want your ASP.NET applications to work with items that are outside the base application. Examples include files and streams. This chapter takes a close look at working with various file types and streams that might come into your ASP.NET applications.
- **Chapter 25, "User and Server Controls."** Not only can you use the plethora of server controls that come with ASP.NET, but you can also use the same framework these controls use and build your own. This chapter describes building your own server controls and how to use them within your applications.
- **Chapter 26, "Modules and Handlers."** Sometimes, just creating dynamic Web pages with the latest languages and databases does not give you, the developer, enough control over an application. At times, you need to be able to dig deeper and create applications that can interact with the Web server itself. You want to be able to interact with the low-level processes, such as how the Web server processes incoming and outgoing HTTP requests. This chapter looks at two methods of manipulating the way ASP.NET processes HTTP requests: HttpModule and HttpHandler. Each method provides a unique level of access to the underlying processing of ASP.NET, and each can be a powerful tool for creating Web applications.
- **Chapter 27, "ASP.NET MVC."** ASP.NET MVC is the latest major addition to ASP.NET and has generated a lot of excitement from the development community. ASP.NET MVC supplies you with the means to create ASP.NET using the Model-View-Controller models that many developers expect. ASP.NET MVC provides developers with the testability, flexibility, and maintainability in the applications they build. It is important to remember that ASP.NET MVC is not meant to be

a replacement to the ASP.NET everyone knows and loves, but instead is simply a different way to construct your applications.

- **Chapter 28, "Using Business Objects."** Invariably, you are going to have components created with previous technologies that you do not want to rebuild but that you do want to integrate into new ASP.NET applications. If this is the case, the .NET Framework makes incorporating your previous COM components into your applications fairly simple and straightforward. This chapter also shows you how to build .NET components instead of turning to the previous COM component architecture.
- **Chapter 29, "ADO.NET Entity Framework."** Mapping objects from the database to the objects within your code is always a laborious and sometimes difficult process. The inclusion of the ADO.NET Entity Framework in ASP.NET makes this task significantly simpler. Using Visual Studio 2010, you are able to visually design your entity data models and then very easily access these models from code allowing the ADO.NET Entity Framework to handle the connections and transactions to the underlying database.
- **Chapter 30, "ASP.NET Dynamic Data."** This feature in ASP.NET 4 allows you to quickly and easily put together a reporting and data entry application from your database. You are also able to take these same capabilities and incorporate them into a pre-existing application.
- **Chapter 31, "Working with Services."** XML Web services have monopolized all the hype for the past few years, and a major aspect of the Web services model within .NET is part of ASP.NET. This chapter reveals the ease not only of building XML Web services, but consuming them in an ASP.NET application. This chapter then ventures further by describing how to build XML Web services that utilize SOAP headers and how to consume this particular type of service. Another feature in ASP.NET, WCF Data Services, allows you to create a RESTful service layer using an Entity Data Model. Using this capability, you can quickly set up a service layer that allows you to expose your content as AtomPub or JSON, which will allow the consumer to completely interact with the underlying database.
- **Chapter 32, "Building Global Applications."** Developers usually build Web applications in the English language and then, as the audience for the application expands, they realize the need to globalize the application. Of course, building the Web application to handle an international audience right from the start is ideal, but, in many cases, this may not be possible because of the extra work it requires. ASP.NET provides an outstanding way to address the internationalization of Web applications. Changes to the API, the addition of capabilities to the server controls, and even Visual Studio itself equip you to do the extra work required to more easily bring your application to an international audience. This chapter looks at some of the important items to consider when building your Web applications for the world.
- **Chapter 33, "Configuration."** Configuration in ASP.NET can be a big topic because the ASP.NET team is not into building black boxes; instead, it is building the underlying capabilities of ASP.NET in a fashion that can easily be expanded on later. This chapter teaches you to modify the capabilities and behaviors of ASP.NET using the various configuration files at your disposal.
- **Chapter 34, "Instrumentation."** ASP.NET gives you greater capability to apply instrumentation techniques to your applications. The ASP.NET Framework includes performance counters, the capability to work with the Windows Event Tracing system, possibilities for application tracing (covered in Chapter 23 of this book), and the most exciting part of this discussion — a health monitoring system that allows you to log a number of different events over an application's lifetime. This chapter takes an in-depth look at this health monitoring system.
- **Chapter 35, "Administration and Management."** Besides making it easier for the developer to be more productive in building ASP.NET applications, the ASP.NET team also put considerable effort into making the managing of applications easier. In the past, using ASP.NET 1.0/1.1, you managed ASP.NET applications by changing values in an XML configuration file. This chapter provides an overview of the GUI tools that come with ASP.NET today that enable you to manage your Web applications easily and effectively.

- **Chapter 36, "Packaging and Deploying ASP.NET Applications."** So you have built an ASP.NET application — now what? This chapter takes the building process one step further and shows you how to package your ASP.NET applications for easy deployment. Many options are available for working with the installers and compilation model to change what you are actually giving your customers.
- **Appendix A, "Migrating Older ASP.NET Projects."** In some cases, you build your ASP.NET 4 applications from scratch, starting everything new. In many instances, however, this is not an option. You need to take an ASP.NET application that was previously built on the 1.0, 1.1, 2.0, or 3.5 versions of the .NET Framework and migrate the application so that it can run on the .NET Framework 4. This appendix focuses on migrating ASP.NET 1.*x*, 2.0, or 3.5 applications to the 4 Framework.
- **Appendix B, "ASP.NET Ultimate Tools."** This appendix takes a look at the tools available to you as an ASP.NET developer. Many of the tools here will help you to expedite your development process and, in many cases, make you a better developer.
- **Appendix C, "Silverlight 3 and ASP.NET."** Silverlight is a means to build fluid applications using XAML. This technology enables developers with really rich vector-based applications.
- **Appendix D, "Dynamic Types and Languages."** As of the release of ASP.NET 4, you can now build your Web applications using IronRuby and IronPython. This appendix takes a quick look at using dynamic languages in building your Web applications.
- **Appendix E, "ASP.NET Online Resources."** This small appendix points you to some of the more valuable online resources for enhancing your understanding of ASP.NET.

CONVENTIONS

This book uses a number of different styles of text and layout to help differentiate among various types of information. Here are examples of the styles used and an explanation of what they mean:

- New words being defined are shown in *italics*.
- Keys that you press on the keyboard, such as Ctrl and Enter, are shown in initial caps and spelled as they appear on the keyboard.
- File names, file extensions, URLs, and code that appears in regular paragraph text are shown in a `monospaced` typeface.

A block of code that you can type as a program and run is shown on separate lines, like this:

```
 public static void Main()
{
   AFunc(1,2,"abc");
}
```

or like this:

```
public static void Main()    {       AFunc(1,2,"abc");    }
```

Sometimes you see code in a mixture of styles, like this:

```
 // If we haven't reached the end, return true, otherwise
// set the position to invalid, and return false.
pos++;
if (pos < 4)
   return true;
else {
   pos = -1;
   return false;
}
```

When mixed code is shown like this, the bold code is what you should focus on in the current example.

We demonstrate the syntactical usage of methods, properties, and so on using the following format:

```
SqlDependency="database:table"
```

Here, the italicized parts indicate *placeholder text*: object references, variables, or parameter values that you need to insert.

Most of the code examples throughout the book are presented as numbered listings that have descriptive titles, like this:

LISTING I-7: Targeting WML devices in your ASP.NET pages

Each listing is numbered (for example, *Listing 1-3*) where the first number represents the chapter number and the number following the hyphen represents a sequential number that indicates where that listing falls within the chapter. Downloadable code from the Wrox Web site (`www.wrox.com`) also uses this numbering system (for the most part) so that you can easily locate the examples you are looking for.

All code is shown in both VB and C#, when warranted. The exception is for code in which the only difference is, for example, the value given to the `Language` attribute in the `Page` directive. In such situations, we don't repeat the code for the C# version; the code is shown only once, as in the following example:

```
<%@ Page Language="VB"%>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DataSetDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
       <asp:DropDownList ID="Dropdownlist1" Runat="server" DataTextField="name"
        DataSourceID="XmlDataSource1">
       </asp:DropDownList>

       <asp:XmlDataSource ID="XmlDataSource1" Runat="server"
        DataFile="</Painters.xml">
       </asp:DataSetDataSource>
   </form>
</body>
</html>
```

SOURCE CODE

As you work through the examples in this book, you may choose either to type all the code manually or to use the source code files that accompany the book. All the source code used in this book is available for download at `www.wrox.com`. When you get to the site, simply locate the book's title (either by using the Search box or one of the topic lists) and click the Download Code link. You can then choose to download all the code from the book in one large Zip file or download just the code you need for a particular chapter.

Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-0-470-50220-4.

After you download the code, just decompress it with your favorite compression tool. Alternatively, you can go to the main Wrox code download page at `www.wrox.com/dynamic/books/download.aspx` to see the code available for this book and all other Wrox books. Remember that you can easily find the code you are looking for by referencing the listing number of the code example from the book, such as "Listing 1-1."

We used these listing numbers when naming most of the downloadable code files. Those few listings that are not named by their listing number are accompanied by the file name so you can easily find them in the downloadable code files.

ERRATA

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

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

If you do not spot "your" error already on the Book Errata page, go to `www.wrox.com/contact/techsupport.shtml` and complete the form there to send us the error you have found. We will check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

P2P.WROX.COM

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

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

1. Go to `p2p.wrox.com` and click the Register link.
2. Read the terms of use and click Agree.
3. Supply the information required to join, as well as any optional information you want to provide, and click Submit.

You will receive an e-mail with information describing how to verify your account and complete the joining process.

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

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

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

1

Application and Page Frameworks

WHAT'S IN THIS CHAPTER?

- ➤ Choosing application location and page structure options
- ➤ Working with page directives, page events, and application folders
- ➤ Choosing compilation options

The evolution of ASP.NET continues! The progression from Active Server Pages 3.0 to ASP.NET 1.0 was revolutionary, to say the least. And now the revolution continues with the latest release of ASP. NET — version 4. The original introduction of ASP.NET 1.0 fundamentally changed the Web programming model. ASP.NET 4 is just as revolutionary in the way it will increase your productivity. As of late, the primary goal of ASP.NET is to enable you to build powerful, secure, dynamic applications using the least possible amount of code. Although this book covers the new features provided by ASP.NET 4, it also covers all the offerings of ASP.NET technology.

If you are new to ASP.NET and building your first set of applications in ASP.NET 4, you may be amazed by the vast amount of wonderful server controls it provides. You may marvel at how it enables you to work with data more effectively using a series of data providers. You may be impressed at how easily you can build in security and personalization.

The outstanding capabilities of ASP.NET 4 do not end there, however. This chapter looks at many exciting options that facilitate working with ASP.NET pages and applications. One of the first steps you, the developer, should take when starting a project is to become familiar with the foundation you are building on and the options available for customizing that foundation.

APPLICATION LOCATION OPTIONS

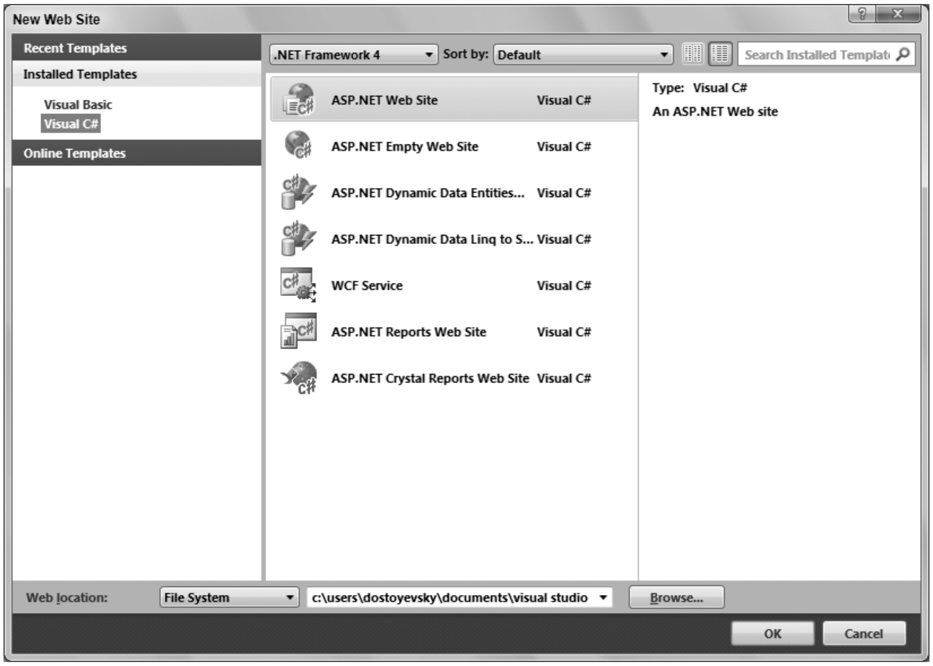
With ASP.NET 4, you have the option — using Visual Studio 2010 — to create an application with a virtual directory mapped to IIS or a standalone application outside the confines of IIS. Whereas, the early Visual Studio .NET 2002/2003 IDEs forced developers to use IIS for all Web applications, Visual Studio 2008/2010 (and Visual Web Developer 2008/2010 Express Edition, for that matter) includes a built-in Web server that you can use for development, much like the one used in the past with the ASP.NET Web Matrix.

This built-in Web server was previously presented to developers as a code sample called Cassini. In fact, the code for this mini Web server is freely downloadable from the ASP.NET team Web site found at `www.asp.net`.

The following section shows you how to use the built-in Web server that comes with Visual Studio 2010.

Built-in Web Server

By default, Visual Studio 2010 builds applications without the use of IIS. You can see this when you select File ➪ New ➪ Web Site in the IDE. By default, the location provided for your application is in `C:\Users\BillEvjen\Documents\Visual Studio 10\WebSites` if you are using Windows 7 (shown in Figure 1-1). It is not `C:\Inetpub\wwwroot\` as it would have been in Visual Studio .NET 2002/2003. By default, any site that you build and host inside `C:\Users\BillEvjen\Documents\Visual Studio 10\WebSites` (or any other folder you create) uses the built-in Web server that is part of Visual Studio 2010. If you use the built-in Web server from Visual Studio 2010, you are not locked into the WebSites folder; you can create any folder you want in your system.



ASP.NET Web Site	Visual C#
ASP.NET Empty Web Site	Visual C#
ASP.NET Dynamic Data Entities...	Visual C#
ASP.NET Dynamic Data Linq to S...	Visual C#
WCF Service	Visual C#
ASP.NET Reports Web Site	Visual C#
ASP.NET Crystal Reports Web Site	Visual C#

Type: Visual C#
An ASP.NET Web site

Web location: File System c:\users\dostoyevsky\documents\visual studio

FIGURE 1-1

To change from this default, you have a handful of options. Click the Browse button in the New Web Site dialog. The Choose Location dialog opens, shown in Figure 1-2.

If you continue to use the built-in Web server that Visual Studio 2010 provides, you can choose a new location for your Web application from this dialog. To choose a new location, select a new folder and save your `.aspx` pages and any other associated files to this directory. When using Visual Studio 2010, you can run your application completely from this location. This way of working with the ASP.NET pages you create is ideal if you do not have access to a Web server because it enables you to build applications that do not reside on a machine with IIS. This means that you can even develop ASP.NET applications on operating systems such as Windows 7 Home Edition.

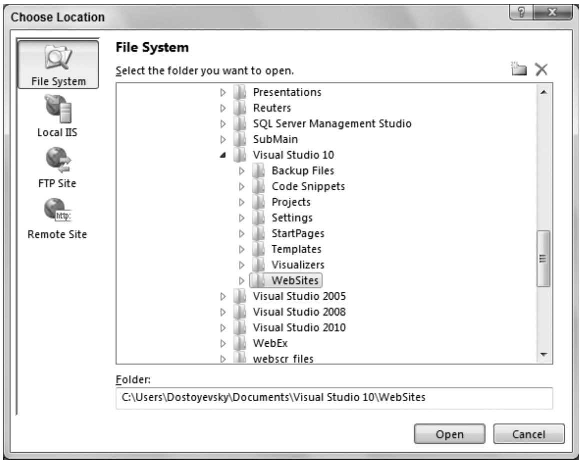


FIGURE 1-2

IIS

From the Choose Location dialog, you can also change where your application is saved and which type of Web server your application employs. To use IIS (as you probably did when you used Visual Studio .NET 2002/2003), select the Local IIS button in the dialog. This changes the results in the text area to show you a list of all the virtual application roots on your machine. You are required to run Visual Studio as an administrator user if you want to see your local IIS instance.

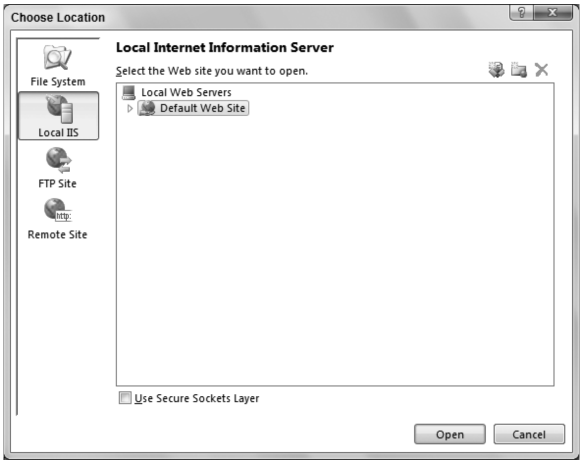


FIGURE 1-3

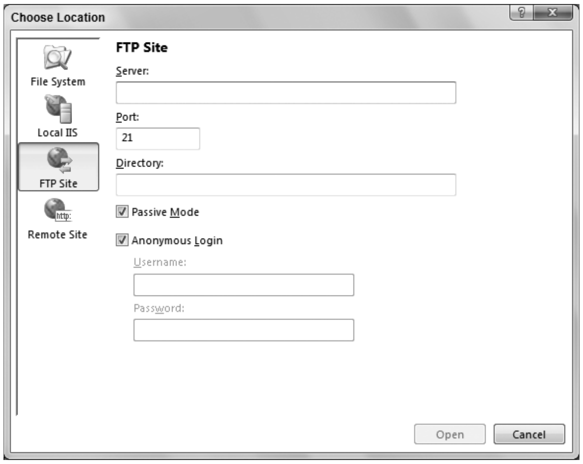
To create a new virtual root for your application, highlight Default Web Site. Two accessible buttons appear at the top of the dialog box (see Figure 1-3). When you look from left to right, the first button in the upper-right corner of the dialog box is for creating a new Web application — or a virtual root. This button is shown as a globe inside a box. The second button enables you to create virtual directories for any of the virtual roots you created. The third button is a Delete button, which allows you to delete any selected virtual directories or virtual roots on the server.

After you have created the virtual directory you want, click the Open button. Visual Studio 2010 then goes through the standard process to create your application. Now, however, instead of depending on the built-in Web server from ASP.NET 4, your application will use IIS. When you invoke your application, the URL now consists of something like `http://localhost/MyWeb/Default.aspx`, which means it is using IIS.

FTP

Not only can you decide on the type of Web server for your Web application when you create it using the Choose Location dialog, but you can also decide where your application is going to be located. With the previous options, you built applications that resided on your local server. The FTP option enables you to actually store and even code your applications while they reside on a server somewhere else in your enterprise — or on the other side of the planet. You can also use the FTP capabilities to work on different locations within the same server. Using this capability provides a wide range of possible options. You can see this in Figure 1-4.

To create your application on a remote server using FTP, simply provide the server name, the port to use, and the directory — as well as any required credentials. If the correct information is provided, Visual Studio 2010 reaches out to the remote server and creates the appropriate files for the start of your application, just as if it were doing the job locally. From this point on, you can open your project and connect to the remote server using FTP.



FTP Site

Server:

Port: 21

Directory:

☑ Passive Mode

☑ Anonymous Login

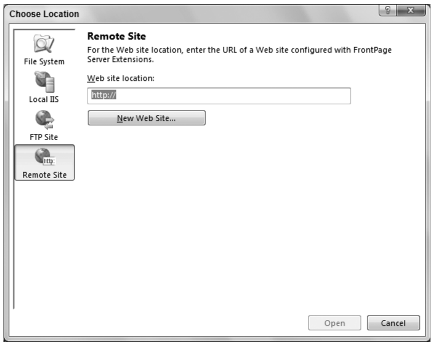
Username:

Password:

FIGURE 1-4

Web Site Requiring FrontPage Extensions

The last option in the Choose Location dialog is the Remote Site option. Clicking this button provides a dialog that enables you to connect to a remote or local server that utilizes FrontPage Extensions. This option is displayed in Figure 1-5.



Remote Site

For the Web site location, enter the URL of a Web site configured with FrontPage Server Extensions.

Web site location: http://

FIGURE 1-5

THE ASP.NET PAGE STRUCTURE OPTIONS

ASP.NET 4 provides two paths for structuring the code of your ASP.NET pages. The first path utilizes the code-inline model. This model should be familiar to classic ASP 2.0/3.0 developers because all the code is contained within a single `.aspx` page. The second path uses ASP.NET's code-behind model, which allows for code separation of the page's business logic from its presentation logic. In this model, the presentation logic for the page is stored in an `.aspx` page, whereas the logic piece is stored in a separate class file: `.aspx.vb` or `.aspx.cs`. Using the code-behind model is considered the best practice because it provides a clean model in separation of pure UI elements from code that manipulates these elements. It is also seen as a better means in maintaining code.

One of the major complaints about Visual Studio .NET 2002 and 2003 is that it forced you to use the code-behind model when developing your ASP.NET pages because it did not understand the code-inline model. The code-behind model in ASP.NET was introduced as a new way to separate the presentation code and business logic. Listing 1-1 shows a typical `.aspx` page generated using Visual Studio .NET 2002 or 2003.

LISTING 1-1: A typical .aspx page from ASP.NET 1.0/1.1

Available for download on Wrox.com

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
    Inherits="WebApplication.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
 <HEAD>
    <title>WebForm1</title>
    <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.1">
    <meta name="CODE_LANGUAGE" content="Visual Basic .NET 7.1">
```

```
        <meta name="vs_defaultClientScript" content="JavaScript">
        <meta name="vs_targetSchema"
          content="http://schemas.microsoft.com/intellisense/ie5">
     </HEAD>
     <body>
        <form id="Form1" method="post" runat="server">
           <P>What is your name?<br>
           <asp:TextBox id="TextBox1" runat="server"></asp:TextBox><BR>
           <asp:Button id="Button1" runat="server" Text="Submit"></asp:Button></P>
           <P><asp:Label id="Label1" runat="server"></asp:Label></P>
        </form>
     </body>
    </HTML>
```

The code-behind file created within Visual Studio .NET 2002/2003 for the `.aspx` page is shown in Listing 1-2.

LISTING 1-2: A typical .aspx.vb/.aspx.cs page from ASP.NET 1.0/1.1

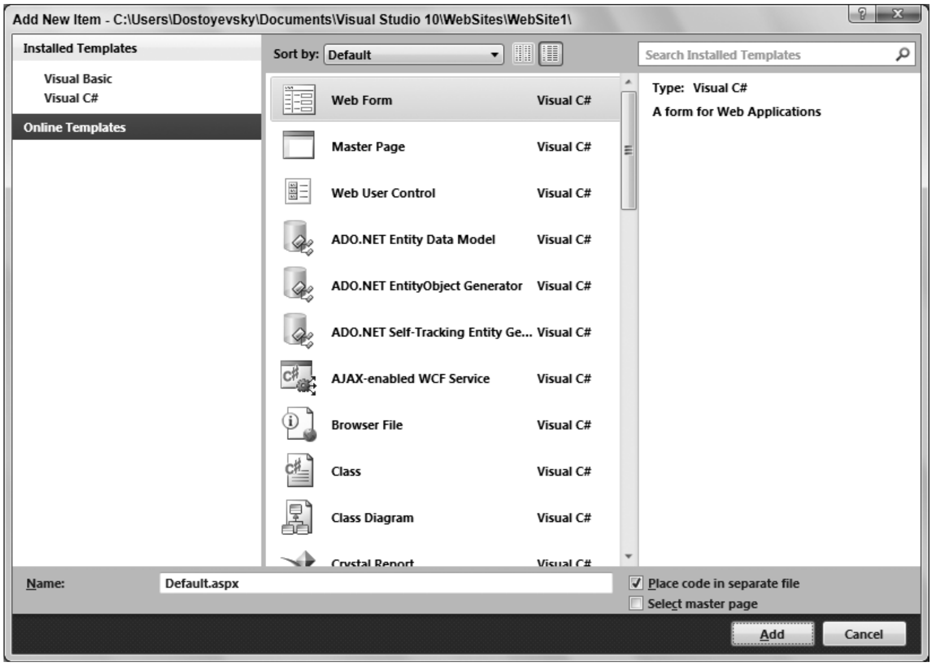
Available for download on Wrox.com

```
Public Class WebForm1
    Inherits System.Web.UI.Page

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
     <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()

    End Sub
    Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label

   'NOTE: The following placeholder declaration is required by the Web Form
        Designer.
   'Do not delete or move it.
    Private designerPlaceholderDeclaration As System.Object

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles MyBase.Init
       'CODEGEN: This method call is required by the Web Form Designer
       'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles MyBase.Load
       'Put user code to initialize the page here
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles Button1.Click
        Label1.Text = "Hello " & TextBox1.Text
    End Sub
End Class
```

In this code-behind page from ASP.NET 1.0/1.1, you can see that a lot of the code that developers never have to deal with is hidden in the `#Region` section of the page. Because ASP.NET 4 is built on top of .NET 4, it can take advantage of the .NET Framework capability of partial classes. Partial classes enable you to separate your classes into multiple class files, which are then combined into a single class when the application is compiled. Because ASP.NET 4 combines all this page code for you behind the scenes when the application is compiled, the code-behind files you work with in ASP.NET 4 are simpler in appearance and the model is easier to use. You are presented with only the pieces of the class that you need. Next, this chapter presents a look at both the inline and code-behind models from ASP.NET 4.

Inline Coding

With the .NET Framework 1.0/1.1, developers went out of their way (and outside Visual Studio .NET) to build their ASP.NET pages inline and avoid the code-behind model that was so heavily promoted by Microsoft and others. Visual Studio 2010 (as well as Visual Web Developer 2010 Express Edition) allows you to build your pages easily using this coding style. To build an ASP.NET page inline instead of using the code-behind model, you simply select the page type from the Add New Item dialog and make sure that the Place Code in Separate File check box is not selected. You can get at this dialog (see Figure 1-6) by right-clicking the project or the solution in the Solution Explorer and selecting Add New Item.


Add New Item - C:\Users\Dostoyevsky\Documents\Visual Studio 10\WebSites\WebSite1\

Sort by: Default

Type: Visual C#

A form for Web Applications

Name: Default.aspx

☑ Place code in separate file

☐ Select master page

Installed Templates
Visual Basic
Visual C#
Online Templates
Search Installed Templates
Web Form Visual C#
Master Page Visual C#
Web User Control Visual C#
ADO.NET Entity Data Model Visual C#
ADO.NET EntityObject Generator Visual C#
ADO.NET Self-Tracking Entity Ge... Visual C#
AJAX-enabled WCF Service Visual C#
Browser File Visual C#
Class Visual C#
Class Diagram Visual C#
Crystal Report Visual C#
Add
Cancel

FIGURE 1-6

From here, you can see the check box you need to unselect if you want to build your ASP.NET pages inline. In fact, many page types have options for both inline and code-behind styles. Table 1-1 shows your inline options when selecting files from this dialog.

TABLE 1-1

FILE OPTIONS USING INLINE CODING	FILE CREATED
Web Form	`.aspx` file
AJAX Web Form	`.aspx` file
Master Page	`.master` file
AJAX Master Page	`.master` file
Web User Control	`.ascx` file
Web Service	`.asmx` file

By using the Web Form option with a few controls, you get a page that encapsulates not only the presentation logic, but the business logic as well. This is illustrated in Listing 1-3.

LISTING 1-3: A simple page that uses the inline coding model

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object,
       ByVal e As System.EventArgs)

        Label1.Text = "Hello " & Textbox1.Text
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Simple Page</title>
</head>
<body>
    <form id="form1" runat="server">
        What is your name?<br />
        <asp:Textbox ID="Textbox1" Runat="server"></asp:Textbox><br />
        <asp:Button ID="Button1" Runat="server" Text="Submit"
         OnClick="Button1_Click" />
        <p><asp:Label ID="Label1" Runat="server"></asp:Label></p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    protected void Button1_Click(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + Textbox1.Text;
    }
</script>
```

From this example, you can see that all the business logic is encapsulated in between `<script>` tags. The nice feature of the inline model is that the business logic and the presentation logic are contained within the same file. Some developers find that having everything in a single viewable instance makes working with the ASP.NET page easier. Another great thing is that Visual Studio 2010 provides IntelliSense when working with the inline coding model and ASP.NET 4. Before Visual Studio 2005, this capability did not exist. Visual Studio .NET 2002/2003 forced you to use the code-behind model and, even if you rigged it so your pages were using the inline model, you lost all IntelliSense capabilities.

Code-Behind Model

The other option for constructing your ASP.NET 4 pages is to build your files using the code-behind model.

It is important to note that the more preferred method is the code-behind model rather than the inline model. This method employs the proper segmentation between presentation and business logic in many cases. You will find that many of the examples in this book use an inline coding model because it works well in showing an example in one listing. Even though the example is using an inline coding style, it is my recommendation that you move the code to employ the code-behind model.

To create a new page in your ASP.NET solution that uses the code-behind model, select the page type you want from the New File dialog. To build a page that uses the code-behind model, you first select the page in the Add New Item dialog and make sure the Place Code in Separate File check box is selected. Table 1-2 shows you the options for pages that use the code-behind model.

TABLE 1-2

FILE OPTIONS USING CODE-BEHIND	FILE CREATED
Web Form	`.aspx` file; `.aspx.vb` or `.aspx.cs` file
AJAX Web Form	`.aspx` file; `.aspx.vb` or `.aspx.cs` file
Master Page	`.master` file; `.master.vb` or `.master.cs` file
AJAX Master Page	`.master.vb` or `.master.cs` file
Web User Control	`.ascx` file; `.ascx.vb` or `.ascx.cs` file
Web Service	`.asmx` file; `.vb` or `.cs` file

The idea of using the code-behind model is to separate the business logic and presentation logic into separate files. Doing this makes working with your pages easier, especially if you are working in a team environment where visual designers work on the UI of the page and coders work on the business logic that sits behind the presentation pieces. Earlier in Listings 1-1 and 1-2, you saw how pages using the code-behind model in ASP.NET 1.0/1.1 were constructed. To see the difference in ASP.NET 4, look at how its code-behind pages are constructed. These differences are illustrated in Listing 1-4 for the presentation piece and Listing 1-5 for the code-behind piece.

LISTING 1-4: An .aspx page that uses the ASP.NET 4 code-behind model

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    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>Simple Page</title>
</head>
<body>
    <form id="form1" runat="server">
        What is your name?<br />
        <asp:Textbox ID="Textbox1" Runat="server"></asp:Textbox><br />
        <asp:Button ID="Button1" Runat="server" Text="Submit"
         OnClick="Button1_Click" />
        <p><asp:Label ID="Label1" Runat="server"></asp:Label></p>
    </form>
</body>
</html>
```

C#

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

LISTING 1-5: A code-behind page

Available for download on Wrox.com

VB

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs) Handles Button1.Click

        Label1.Text = "Hello " & TextBox1.Text
    End Sub
End Class
```

C#

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

public partial class _Default : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
       Label1.Text = "Hello " + Textbox1.Text;
    }
}
```

The `.aspx` page using this ASP.NET 4 code-behind model has some attributes in the `Page` directive that you should pay attention to when working in this mode. The first is the `CodeFile` attribute. This attribute in the `Page` directive is meant to point to the code-behind page that is used with this presentation page. In this case, the value assigned is `Default.aspx.vb` or `Default.aspx.cs`. The second attribute needed is the `Inherits` attribute. This attribute was available in previous versions of ASP.NET, but was little used before ASP.NET 2.0. This attribute specifies the name of the class that is bound to the page when the page is compiled. The directives are simple enough in ASP.NET 4. Look at the code-behind page from Listing 1-5.

The code-behind page is rather simple in appearance because of the partial class capabilities that .NET 4 provides. You can see that the class created in the code-behind file uses partial classes, employing the `Partial` keyword in Visual Basic 2010 and the `partial` keyword from C# 2010. This enables you to simply place the methods that you need in your page class. In this case, you have a button-click event and nothing else.

Later in this chapter, you look at the compilation process for both of these models.

ASP.NET 4 PAGE DIRECTIVES

ASP.NET directives are something that is a part of every ASP.NET page. You can control the behavior of your ASP.NET pages by using these directives. Here is an example of the `Page` directive:

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>
```

Eleven directives are at your disposal in your ASP.NET pages or user controls. You use these directives in your applications whether the page uses the code-behind model or the inline coding model.

Basically, these directives are commands that the compiler uses when the page is compiled. Directives are simple to incorporate into your pages. A directive is written in the following format:

```
<%@ [Directive] [Attribute=Value] %>
```

From this, you can see that a directive is opened with a `<%@` and closed with a `%>`. Putting these directives at the top of your pages or controls is best because this is traditionally where developers expect to see them (although the page still compiles if the directives are located at a different place). Of course, you can also add more than a single attribute to your directive statements, as shown in the following:

```
<%@ [Directive] [Attribute=Value] [Attribute=Value] %>
```

Table 1-3 describes the directives at your disposal in ASP.NET 4.

TABLE 1-3

DIRECTIVE	DESCRIPTION
`Assembly`	Links an assembly to the page or user control for which it is associated.
`Control`	Page directive meant for use with user controls (`.ascx`).
`Implements`	Implements a specified .NET Framework interface.
`Import`	Imports specified namespaces into the page or user control.
`Master`	Enables you to specify master page–specific attributes and values to use when the page parses or compiles. This directive can be used only with master pages (`.master`).
`MasterType`	Associates a class name to a page to get at strongly typed references or members contained within the specified master page.
`OutputCache`	Controls the output caching policies of a page or user control.
`Page`	Enables you to specify page-specific attributes and values to use when the page parses or compiles. This directive can be used only with ASP.NET pages (`.aspx`).
`PreviousPageType`	Enables an ASP.NET page to work with a postback from another page in the application.
`Reference`	Links a page or user control to the current page or user control.
`Register`	Associates aliases with namespaces and class names for notation in custom server control syntax.

The following sections provide a quick review of each of these directives.

@Page

The `@Page` directive enables you to specify attributes and values for an ASP.NET page (`.aspx`) to be used when the page is parsed or compiled. This is the most frequently used directive of the bunch. Because the ASP.NET page is such an important part of ASP.NET, you have quite a few attributes at your disposal. Table 1-4 summarizes the attributes available through the `@Page` directive.

TABLE 1-4

ATTRIBUTE	DESCRIPTION
`AspCompat`	Permits the page to be executed on a single-threaded apartment thread when given a value of `True`. The default setting for this attribute is `False`.
`Async`	Specifies whether the ASP.NET page is processed synchronously or asynchronously.
`AsyncTimeout`	Specifies the amount of time in seconds to wait for the asynchronous task to complete. The default setting is 45 seconds.
`AutoEventWireup`	Specifies whether the page events are autowired when set to `True`. The default setting for this attribute is `True`.
`Buffer`	Enables HTTP response buffering when set to `True`. The default setting for this attribute is `True`.
`ClassName`	Specifies the name of the class that is bound to the page when the page is compiled.

ATTRIBUTE	DESCRIPTION
`ClientIDMode`	Specifies the algorithm that the page should use when generating ClientID values for server controls that are on the page. The default value is AutoID (the mode that was used for ASP.NET pages prior to ASP.NET 4). This is a new attribute of ASP.NET 4.
`ClientTarget`	Specifies the target user agent a control should render content for. This attribute needs to be tied to an alias defined in the `<clientTarget>` section of the `web.config` file.
`CodeFile`	References the code-behind file with which the page is associated.
`CodeFileBaseClass`	Specifies the type name of the base class to use with the code-behind class, which is used by the `CodeFile` attribute.
`CodePage`	Indicates the code page value for the response.
`CompilationMode`	Specifies whether ASP.NET should compile the page or not. The available options include `Always` (the default), `Auto`, or `Never`. A setting of `Auto` means that if possible, ASP.NET will not compile the page.
`CompilerOptions`	Compiler string that indicates compilation options for the page.
`CompileWith`	Takes a `String` value that points to the code-behind file used.
`ContentType`	Defines the HTTP content type of the response as a standard MIME type.
`Culture`	Specifies the culture setting of the page. ASP.NET 3.5 and 4 include the capability to give the `Culture` attribute a value of `Auto` to enable automatic detection of the culture required.
`Debug`	Compiles the page with debug symbols in place when set to `True`.
`Description`	Provides a text description of the page. The ASP.NET parser ignores this attribute and its assigned value.
`EnableEventValidation`	Specifies whether to enable validation of events in postback and callback scenarios. The default setting of `True` means that events will be validated.
`EnableSessionState`	Session state for the page is enabled when set to `True`. The default setting is `True`.
`EnableTheming`	Page is enabled to use theming when set to `True`. The default setting for this attribute is `True`.
`EnableViewState`	View state is maintained across the page when set to `True`. The default value is `True`.
`EnableViewStateMac`	Page runs a machine-authentication check on the page's view state when the page is posted back from the user when set to `True`. The default value is `False`.
`ErrorPage`	Specifies a URL to post to for all unhandled page exceptions.
`Explicit`	Visual Basic `Explicit` option is enabled when set to `True`. The default setting is `False`.
`Language`	Defines the language being used for any inline rendering and script blocks.
`LCID`	Defines the locale identifier for the Web Form's page.
`LinePragmas`	`Boolean` value that specifies whether line pragmas are used with the resulting assembly.
`MasterPageFile`	Takes a `String` value that points to the location of the master page used with the page. This attribute is used with content pages.
`MaintainScrollPositionOnPostback`	Takes a `Boolean` value, which indicates whether the page should be positioned exactly in the same scroll position or whether the page should be regenerated in the uppermost position for when the page is posted back to itself.

continues

TABLE 1-4 *(continued)*

ATTRIBUTE	DESCRIPTION
`MetaDescription`	Allows you to specify a page's description in a meta tag for SEO purposes. This is a new attribute in ASP.NET 4.
`MetaKeywords`	Allows you to specify a page's keywords in a meta tag for SEO purposes. This is a new attribute in ASP.NET 4.
`ResponseEncoding`	Specifies the response encoding of the page content.
`SmartNavigation`	Specifies whether to activate the ASP.NET Smart Navigation feature for richer browsers. This returns the postback to the current position on the page. The default value is `False`. Since ASP.NET 2.0, `SmartNavigation` has been deprecated. Use the `SetFocus()` method and the `MaintainScrollPositionOnPostback` property instead.
`Src`	Points to the source file of the class used for the code behind of the page being rendered.
`Strict`	Compiles the page using the Visual Basic `Strict` mode when set to `True`. The default setting is `False`.
`StylesheetTheme`	Applies the specified theme to the page using the ASP.NET themes feature. The difference between the `StylesheetTheme` and `Theme` attributes is that `StylesheetTheme` will not override preexisting style settings in the controls, whereas `Theme` will remove these settings.
`Theme`	Applies the specified theme to the page using the ASP.NET themes feature.
`Title`	Applies a page's title. This is an attribute mainly meant for content pages that must apply a page title other than what is specified in the master page.
`Trace`	Page tracing is enabled when set to `True`. The default setting is `False`.
`TraceMode`	Specifies how the trace messages are displayed when tracing is enabled. The settings for this attribute include `SortByTime` or `SortByCategory`. The default setting is `SortByTime`.
`Transaction`	Specifies whether transactions are supported on the page. The settings for this attribute are `Disabled`, `NotSupported`, `Supported`, `Required`, and `RequiresNew`. The default setting is `Disabled`.
`UICulture`	The value of the `UICulture` attribute specifies what UI Culture to use for the ASP.NET page. ASP.NET 3.5 and 4 include the capability to give the `UICulture` attribute a value of `Auto` to enable automatic detection of the `UICulture`.
`ValidateRequest`	When this attribute is set to `True`, the form input values are checked against a list of potentially dangerous values. This helps protect your Web application from harmful attacks such as JavaScript attacks. The default value is `True`.
`ViewStateEncryptionMode`	Specifies how the ViewState is encrypted on the page. The options include `Auto`, `Always`, and `Never`. The default is `Auto`.
`WarningLevel`	Specifies the compiler warning level at which to stop compilation of the page. Possible values are `0` through `4`.

Here is an example of how to use the `@Page` directive:

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" %>
```

@Master

The @Master directive is quite similar to the @Page directive except that the @Master directive is meant for master pages (.master). In using the @Master directive, you specify properties of the templated page that you will be using in conjunction with any number of content pages on your site. Any content pages (built using the @Page directive) can then inherit from the master page all the master content (defined in the master page using the @Master directive). Although they are similar, the @Master directive has fewer attributes available to it than does the @Page directive. The available attributes for the @Master directive are shown in Table 1-5.

TABLE 1-5

ATTRIBUTE	DESCRIPTION
AutoEventWireup	Specifies whether the master page's events are autowired when set to True. Default setting is True.
ClassName	Specifies the name of the class that is bound to the master page when compiled.
CodeFile	References the code-behind file with which the page is associated.
CompilationMode	Specifies whether ASP.NET should compile the page. The available options include Always (the default), Auto, or Never. A setting of Auto means that if possible, ASP.NET will not compile the page.
CompilerOptions	Compiler string that indicates compilation options for the master page.
CompileWith	Takes a String value that points to the code-behind file used for the master page.
Debug	Compiles the master page with debug symbols in place when set to True.
Description	Provides a text description of the master page. The ASP.NET parser ignores this attribute and its assigned value.
EnableTheming	Indicates the master page is enabled to use theming when set to True. The default setting for this attribute is True.
EnableViewState	Maintains view state for the master page when set to True. The default value is True.
Explicit	Indicates that the Visual Basic Explicit option is enabled when set to True. The default setting is False.
Inherits	Specifies the CodeBehind class for the master page to inherit.
Language	Defines the language that is being used for any inline rendering and script blocks.
LinePragmas	Boolean value that specifies whether line pragmas are used with the resulting assembly.
MasterPageFile	Takes a String value that points to the location of the master page used with the master page. It is possible to have a master page use another master page, which creates a nested master page.
Src	Points to the source file of the class used for the code behind of the master page being rendered.
Strict	Compiles the master page using the Visual Basic Strict mode when set to True. The default setting is False.
WarningLevel	Specifies the compiler warning level at which you want to abort compilation of the page. Possible values are from 0 to 4.

Here is an example of how to use the @Master directive:

```
<%@ Master Language="VB" CodeFile="MasterPage1.master.vb"
    AutoEventWireup="false" Inherits="MasterPage" %>
```

@Control

The `@Control` directive is similar to the `@Page` directive except that `@Control` is used when you build an ASP.NET user control. The `@Control` directive allows you to define the properties to be inherited by the user control. These values are assigned to the user control as the page is parsed and compiled. The available attributes are fewer than those of the `@Page` directive, but quite a few of them allow for the modifications you need when building user controls. Table 1-6 details the available attributes.

TABLE 1-6

ATTRIBUTE	DESCRIPTION
`AutoEventWireup`	Specifies whether the user control's events are autowired when set to `True`. Default setting is `True`.
`ClassName`	Specifies the name of the class that is bound to the user control when the page is compiled.
`ClientIDMode`	Specifies the algorithm that the page should use when generating ClientID values for server controls that are on the page. The default value is `AutoID` (the mode that was used for ASP.NET pages prior to ASP.NET 4). This is a new attribute of ASP.NET 4.
`CodeFileBaseClass`	Specifies the type name of the base class to use with the code-behind class, which is used by the `CodeFile` attribute.
`CodeFile`	References the code-behind file with which the user control is associated.
`CompilerOptions`	Compiler string that indicates compilation options for the user control.
`CompileWith`	Takes a `String` value that points to the code-behind file used for the user control.
`Debug`	Compiles the user control with debug symbols in place when set to `True`.
`Description`	Provides a text description of the user control. The ASP.NET parser ignores this attribute and its assigned value.
`EnableTheming`	User control is enabled to use theming when set to `True`. The default setting for this attribute is `True`.
`EnableViewState`	View state is maintained for the user control when set to `True`. The default value is `True`.
`Explicit`	Visual Basic `Explicit` option is enabled when set to `True`. The default setting is `False`.
`Inherits`	Specifies the `CodeBehind` class for the user control to inherit.
`Language`	Defines the language used for any inline rendering and script blocks.
`LinePragmas`	`Boolean` value that specifies whether line pragmas are used with the resulting assembly.
`Src`	Points to the source file of the class used for the code behind of the user control being rendered.
`Strict`	Compiles the user control using the Visual Basic `Strict` mode when set to `True`. The default setting is `False`.
`WarningLevel`	Specifies the compiler warning level at which to stop compilation of the user control. Possible values are `0` through `4`.

The `@Control` directive is meant to be used with an ASP.NET user control. The following is an example of how to use the directive:

```
<%@ Control Language="VB" Explicit="True"
    CodeFile="WebUserControl.ascx.vb" Inherits="WebUserControl"
    Description="This is the registration user control." %>
```

@Import

The `@Import` directive allows you to specify a namespace to be imported into the ASP.NET page or user control. By importing, all the classes and interfaces of the namespace are made available to the page or user control. This directive supports only a single attribute: `Namespace`.

The `Namespace` attribute takes a `String` value that specifies the namespace to be imported. The `@Import` directive cannot contain more than one attribute/value pair. Because of this, you must place multiple namespace imports in multiple lines as shown in the following example:

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
```

Several assemblies are already being referenced by your application. You can find a list of these imported namespaces by looking in the root `web.config` file found at `C:\Windows\Microsoft.NET\Framework\v4.0.xxxxx\Config`. You can find this list of assemblies being referenced from the `<assemblies>` child element of the `<compilation>` element. The settings in the root `web.config` file are as follows:

```
<assemblies>
   <add assembly="mscorlib" />
   <add assembly="Microsoft.CSharp, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
   <add assembly="System, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.Configuration, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
   <add assembly="System.Web, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
   <add assembly="System.Data, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.Web.Services, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
   <add assembly="System.Xml, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.Drawing, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
   <add assembly="System.EnterpriseServices, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
   <add assembly="System.Web.Mobile, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b03f5f7f11d50a3a" />
   <add assembly="System.IdentityModel, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.Runtime.Serialization, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.Xaml, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.ServiceModel, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.ServiceModel.Activation, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35"/>
   <add assembly="System.ServiceModel.Channels, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35"/>
   <add assembly="System.ServiceModel.Web, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35"/>
   <add assembly="System.Activities, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35"/>
   <add assembly="System.ServiceModel.Activities, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35"/>
   <add assembly="System.WorkflowServices, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35"/>
   <add assembly="System.Xaml.Hosting, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35"/>
   <add assembly="System.Core, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.Web.Extensions, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35" />
   <add assembly="System.Data.DataSetExtensions, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.Xml.Linq, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.ComponentModel.DataAnnotations, Version=4.0.0.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
   <add assembly="System.Web.DynamicData, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35"/>
   <add assembly="System.Data.Entity, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.Web.Entity, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089"/>
   <add assembly="System.Data.Linq, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.Data.Entity.Design, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" />
   <add assembly="System.Web.ApplicationServices, Version=4.0.0.0, Culture=neutral,
    PublicKeyToken=31bf3856ad364e35" />
   <add assembly="*" />
</assemblies>
```

Because of this reference in the root `web.config` file, these assemblies need not be referenced in a References folder, as you would have done in ASP.NET 1.0/1.1. You can actually add or delete assemblies that are referenced from this list. For example, if you have a custom assembly referenced continuously by each and every application on the server, you can simply add a similar reference to your custom assembly next to these others. Note that you can perform this same task through the application-specific `web.config` file of your application as well.

Even though assemblies might be referenced, you must still import the namespaces of these assemblies into your pages. The same root `web.config` file contains a list of namespaces automatically imported into each and every page of your application. This is specified through the `<namespaces>` child element of the `<pages>` element.

```
    <namespaces>
        <add namespace="System" />
        <add namespace="System.Collections" />
        <add namespace="System.Collections.Generic" />
        <add namespace="System.Collections.Specialized" />
        <add namespace="System.ComponentModel" />
        <add namespace="System.ComponentModel.DataAnnotations" />
        <add namespace="System.Configuration" />
        <add namespace="System.Data.Entity.Design" />
        <add namespace="System.Data.Linq" />
        <add namespace="System.Linq" />
        <add namespace="System.Text" />
        <add namespace="System.Text.RegularExpressions" />
        <add namespace="System.Web" />
        <add namespace="System.Web.Caching" />
        <add namespace="System.DynamicData" />
        <add namespace="System.Web.SessionState" />
        <add namespace="System.Web.Security" />
        <add namespace="System.Web.Profile" />
        <add namespace="System.Web.UI" />
        <add namespace="System.Web.UI.WebControls" />
        <add namespace="System.Web.UI.WebControls.WebParts" />
        <add namespace="System.Web.UI.HtmlControls" />
        <add namespace="System.Xml.Linq" />
    </namespaces>
```

From this XML list, you can see that quite a number of namespaces are imported into each and every one of your ASP.NET pages. Again, you can feel free to modify this selection in the root `web.config` file or even make a similar selection of namespaces from within your application's `web.config` file.

For instance, you can import your own namespace in the `web.config` file of your application to make the namespace available on every page where it is utilized.

```
<?xml version="1.0"?>
<configuration>
   <system.web>
      <pages>
         <namespaces>
            <add namespace="MyCompany.Utilities" />
         </namespaces>
      </pages>
   </system.web>
</configuration>
```

Remember that importing a namespace into your ASP.NET page or user control gives you the opportunity to use the classes without fully identifying the class name. For example, by importing the namespace `System.Data.OleDb` into the ASP.NET page, you can refer to classes within this namespace by using the singular class name (`OleDbConnection` instead of `System.Data.OleDb.OleDbConnection`).

@Implements

The `@Implements` directive gets the ASP.NET page to implement a specified .NET Framework interface. This directive supports only a single attribute: `Interface`.

The `Interface` attribute directly specifies the .NET Framework interface. When the ASP.NET page or user control implements an interface, it has direct access to all its events, methods, and properties.

Here is an example of the `@Implements` directive:

```
<%@ Implements Interface="System.Web.UI.IValidator" %>
```

@Register

The `@Register` directive associates aliases with namespaces and class names for notation in custom server control syntax. You can see the use of the `@Register` directive when you drag and drop a user control onto any of your `.aspx` pages. Dragging a user control onto the `.aspx` page causes Visual Studio 2010 to create a `@Register` directive at the top of the page. This registers your user control on the page so that the control can then be accessed on the `.aspx` page by a specific name.

The `@Register` directive supports five attributes, as described in Table 1-7.

TABLE 1-7

ATTRIBUTE	DESCRIPTION
`Assembly`	The assembly you are associating with the `TagPrefix`.
`Namespace`	The namespace to relate with `TagPrefix`.
`Src`	The location of the user control.
`TagName`	The alias to relate to the class name.
`TagPrefix`	The alias to relate to the namespace.

Here is an example of how to use the `@Register` directive to import a user control to an ASP.NET page:

```
<%@ Register TagPrefix="MyTag" Namespace="MyName.MyNamespace"
    Assembly="MyAssembly" %>
```

@Assembly

The `@Assembly` directive attaches assemblies, the building blocks of .NET applications, to an ASP.NET page or user control as it compiles, thereby making all the assembly's classes and interfaces available to the page. This directive supports two attributes: `Name` and `Src`.

- `Name`: Enables you to specify the name of an assembly used to attach to the page files. The name of the assembly should include the filename only, not the file's extension. For instance, if the file is `MyAssembly.vb`, the value of the name attribute should be `MyAssembly`.
- `Src`: Enables you to specify the source of the assembly file to use in compilation.

The following provides some examples of how to use the `@Assembly` directive:

```
<%@ Assembly Name="MyAssembly" %>
<%@ Assembly Src="MyAssembly.vb" %>
```

@PreviousPageType

This directive is used to specify the page from which any cross-page postings originate. Cross-page posting between ASP.NET pages is explained later in the section "Cross-Page Posting."

The `@PreviousPageType` directive is a directive that works with the cross-page posting capability that ASP.NET 4 provides. This simple directive contains only two possible attributes: `TypeName` and `VirtualPath`:

- `TypeName`: Sets the name of the derived class from which the postback will occur.
- `VirtualPath`: Sets the location of the posting page from which the postback will occur.

@MasterType

The `@MasterType` directive associates a class name to an ASP.NET page to get at strongly typed references or members contained within the specified master page. This directive supports two attributes:

- `TypeName`: Sets the name of the derived class from which to get strongly typed references or members.
- `VirtualPath`: Sets the location of the page from which these strongly typed references and members will be retrieved.

Details of how to use the `@MasterType` directive are shown in Chapter 5. Here is an example of its use:

```
<%@ MasterType VirtualPath="~/Wrox.master" %>
```

@OutputCache

The `@OutputCache` directive controls the output caching policies of an ASP.NET page or user control. This directive supports the ten attributes described in Table 1-8.

TABLE 1-8

ATTRIBUTE	DESCRIPTION
`CacheProfile`	Allows for a central way to manage an application's cache profile. Use the `CacheProfile` attribute to specify the name of the cache profile detailed in the `web.config` file.
`Duration`	The duration of time in seconds that the ASP.NET page or user control is cached.
`Location`	Location enumeration value. The default is `Any`. This is valid for `.aspx` pages only and does not work with user controls (`.ascx`). Other possible values include `Client`, `Downstream`, `None`, `Server`, and `ServerAndClient`.
`NoStore`	Specifies whether to send a no-store header with the page.
`Shared`	Specifies whether a user control's output can be shared across multiple pages. This attribute takes a `Boolean` value and the default setting is `false`.

ATTRIBUTE	DESCRIPTION
`SqlDependency`	Enables a particular page to use SQL Server cache invalidation.
`VaryByControl`	Semicolon-separated list of strings used to vary the output cache of a user control.
`VaryByCustom`	String specifying the custom output caching requirements.
`VaryByHeader`	Semicolon-separated list of HTTP headers used to vary the output cache.
`VaryByParam`	Semicolon-separated list of strings used to vary the output cache.

Here is an example of how to use the `@OutputCache` directive:

```
<%@ OutputCache Duration="180" VaryByParam="None" %>
```

Remember that the `Duration` attribute specifies the amount of time in *seconds* during which this page is to be stored in the system cache.

@Reference

The `@Reference` directive declares that another ASP.NET page or user control should be compiled along with the active page or control. This directive supports just a single attribute:

➤ `VirtualPath`: Sets the location of the page or user control from which the active page will be referenced.

Here is an example of how to use the `@Reference` directive:

```
<%@ Reference VirtualPath="~/MyControl.ascx" %>
```

ASP.NET PAGE EVENTS

ASP.NET developers consistently work with various events in their server-side code. Many of the events that they work with pertain to specific server controls. For instance, if you want to initiate some action when the end user clicks a button on your Web page, you create a button-click event in your server-side code, as shown in Listing 1-6.

LISTING 1-6: A sample button-click event shown in VB

```
Protected Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
   Label1.Text = TextBox1.Text
End Sub
```

In addition to the server controls, developers also want to initiate actions at specific moments when the ASP.NET page is being either created or destroyed. The ASP.NET page itself has always had a number of events for these instances. The following list shows you all the page events you could use in ASP.NET 1.0/1.1:

➤ `AbortTransaction`

➤ `CommitTransaction`

➤ `DataBinding`

➤ `Disposed`

➤ `Error`

➤ `Init`

➤ `Load`

➤ `PreRender`

➤ `Unload`

One of the more popular page events from this list is the `Load` event, which is used in VB as shown in Listing 1-7.

LISTING 1-7: Using the Page_Load event

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Handles Me.Load

   Response.Write("This is the Page_Load event")
End Sub
```

Besides the page events just shown, ASP.NET 4 has the following events:

- `InitComplete`: Indicates the initialization of the page is completed.
- `LoadComplete`: Indicates the page has been completely loaded into memory.
- `PreInit`: Indicates the moment immediately before a page is initialized.
- `PreLoad`: Indicates the moment before a page has been loaded into memory.
- `PreRenderComplete`: Indicates the moment directly before a page has been rendered in the browser.

An example of using any of these events, such as the `PreInit` event, is shown in Listing 1-8.

LISTING 1-8: Using page events

VB

```
<script runat="server" language="vb">
    Protected Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
       Page.Theme = Request.QueryString("ThemeChange")
    End Sub
</script>
```

C#

```
<script runat="server">
    protected void Page_PreInit(object sender, System.EventArgs e)
    {
       Page.Theme = Request.QueryString["ThemeChange"];
    }
</script>
```

If you create an ASP.NET 4 page and turn on tracing, you can see the order in which the main page events are initiated. They are fired in the following order:

1. `PreInit`
2. `Init`
3. `InitComplete`
4. `PreLoad`
5. `Load`
6. `LoadComplete`
7. `PreRender`
8. `PreRenderComplete`
9. `Unload`

With the addition of these choices, you can now work with the page and the controls on the page at many different points in the page-compilation process. You see these useful page events in code examples throughout the book.

DEALING WITH POSTBACKS

When you are working with ASP.NET pages, be sure you understand the page events just listed. They are important because you place a lot of your page behavior inside these events at specific points in a page lifecycle.

In Active Server Pages 3.0, developers had their pages post to other pages within the application. ASP.NET pages typically post back to themselves to process events (such as a button-click event).

For this reason, you must differentiate between posts for the first time a page is loaded by the end user and *postbacks*. A postback is just that — a posting back to the same page. The postback contains all the form information collected on the initial page for processing if required.

Because of all the postbacks that can occur with an ASP.NET page, you want to know whether a request is the first instance for a particular page or is a postback from the same page. You can make this check by using the `IsPostBack` property of the `Page` class, as shown in the following example:

VB
```
If Page.IsPostBack = True Then
  ' Do processing
End If
```

C#
```
if (Page.IsPostBack == true) {
   // Do processing
}
```

In addition to checking against a `True` or `False` value, you can also find out whether the request is not a postback in the following manner:

VB
```
If Not Page.IsPostBack Then
  ' Do processing
End If
```

C#
```
if (!Page.IsPostBack) {
   // Do processing
}
```

CROSS-PAGE POSTING

One common feature in ASP 3.0 that is difficult to achieve in ASP.NET 1.0/1.1 is the capability to do cross-page posting. Cross-page posting enables you to submit a form (say, `Page1.aspx`) and have this form and all the control values post themselves to another page (`Page2.aspx`).

Traditionally, any page created in ASP.NET 1.0/1.1 simply posted to itself, and you handled the control values within this page instance. You could differentiate between the page's first request and any postbacks by using the `Page.IsPostBack` property, as shown here:

```
If Page.IsPostBack Then
  ' deal with control values
End If
```

Even with this capability, many developers still wanted to be able to post to another page and deal with the first page's control values on that page. This is something that is possible in ASP.NET today, and it is quite a simple process.

For an example, create a page called `Page1.aspx` that contains a simple form. Listing 1-9 shows this page.

LISTING 1-9: Page1.aspx

Available for download on Wrox.com

VB
```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object,
       ByVal e As System.EventArgs)

        Label1.Text = "Hello " & TextBox1.Text & "<br />" &
```

continues

LISTING 1-9 *(continued)*

```
            "Date Selected: " & Calendar1.SelectedDate.ToShortDateString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>First Page</title>
</head>
<body>
    <form id="form1" runat="server">
        Enter your name:<br />
        <asp:Textbox ID="TextBox1" Runat="server">
        </asp:Textbox>
        <p>
        When do you want to fly?<br />
        <asp:Calendar ID="Calendar1" Runat="server"></asp:Calendar></p>
        <br />
        <asp:Button ID="Button1" Runat="server" Text="Submit page to itself"
         OnClick="Button1_Click" />
        <asp:Button ID="Button2" Runat="server" Text="Submit page to Page2.aspx"
         PostBackUrl="</Page2.aspx" />
        <p>
        <asp:Label ID="Label1" Runat="server"></asp:Label></p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click (object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text + "<br />" +
            "Date Selected: " + Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```

The code from `Page1.aspx`, as shown in Listing 1-9, is quite interesting. Two buttons are shown on the page. Both buttons submit the form, but each submits the form to a different location. The first button submits the form to itself. This is the behavior that has been the default for ASP.NET 1.0/1.1. In fact, nothing is different about `Button1`. It submits to `Page1.aspx` as a postback because of the use of the `OnClick` property in the button control. A `Button1_Click` method on `Page1.aspx` handles the values that are contained within the server controls on the page.

The second button, `Button2`, works quite differently. This button does not contain an `OnClick` method as the first button did. Instead, it uses the `PostBackUrl` property. This property takes a string value that points to the location of the file to which this page should post. In this case, it is `Page2.aspx`. This means that `Page2.aspx` now receives the postback and all the values contained in the `Page1.aspx` controls. Look at the code for `Page2.aspx`, shown in Listing 1-10.

LISTING 1-10: Page2.aspx

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
```

```
        Dim pp_Textbox1 As TextBox
        Dim pp_Calendar1 As Calendar

        pp_Textbox1 = CType(PreviousPage.FindControl("Textbox1"), TextBox)
        pp_Calendar1 = CType(PreviousPage.FindControl("Calendar1"), Calendar)

        Label1.Text = "Hello " & pp_Textbox1.Text & "<br />" &
            "Date Selected: " & pp_Calendar1.SelectedDate.ToShortDateString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Second Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" Runat="server"></asp:Label>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        TextBox pp_Textbox1;
        Calendar pp_Calendar1;

        pp_Textbox1 = (TextBox)PreviousPage.FindControl("Textbox1");
        pp_Calendar1 = (Calendar)PreviousPage.FindControl("Calendar1");

        Label1.Text = "Hello " + pp_Textbox1.Text + "<br />" + "Date Selected: " +
           pp_Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```

You have a couple of ways of getting at the values of the controls that are exposed from `Page1.aspx` from the second page. The first option is displayed in Listing 1-10. To get at a particular control's value that is carried over from the previous page, you simply create an instance of that control type and populate this instance using the `FindControl()` method from the `PreviousPage` property. The `String` value assigned to the `FindControl()` method is the `Id` value, which is used for the server control from the previous page. After this is assigned, you can work with the server control and its carried-over values just as if it had originally resided on the current page. You can see from the example that you can extract the `Text` and `SelectedDate` properties from the controls without any problem.

Another way of exposing the control values from the first page (`Page1.aspx`) is to create a `Property` for the control, as shown in Listing 1-11.

LISTING 1-11: Exposing the values of the control from a property

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
```

continues

LISTING 1-11 *(continued)*

```
    Public ReadOnly Property pp_TextBox1() As TextBox
        Get
            Return TextBox1
        End Get
    End Property

    Public ReadOnly Property pp_Calendar1() As Calendar
        Get
            Return Calendar1
        End Get
    End Property

    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & TextBox1.Text & "<br />" &
            "Date Selected: " & Calendar1.SelectedDate.ToShortDateString()
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    public TextBox pp_TextBox1
    {
        get
        {
            return TextBox1;
        }
    }

    public Calendar pp_Calendar1
    {
        get
        {
            return Calendar1;
        }
    }

    protected void Button1_Click (object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text + "<br />" +
            "Date Selected: " + Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```

Filename Page1b.aspx

Now that these properties are exposed on the posting page, the second page (`Page2.aspx`) can more easily work with the server control properties that are exposed from the first page. Listing 1-12 shows you how `Page2.aspx` works with these exposed properties.

LISTING 1-12: Consuming the exposed properties from the first page

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
```

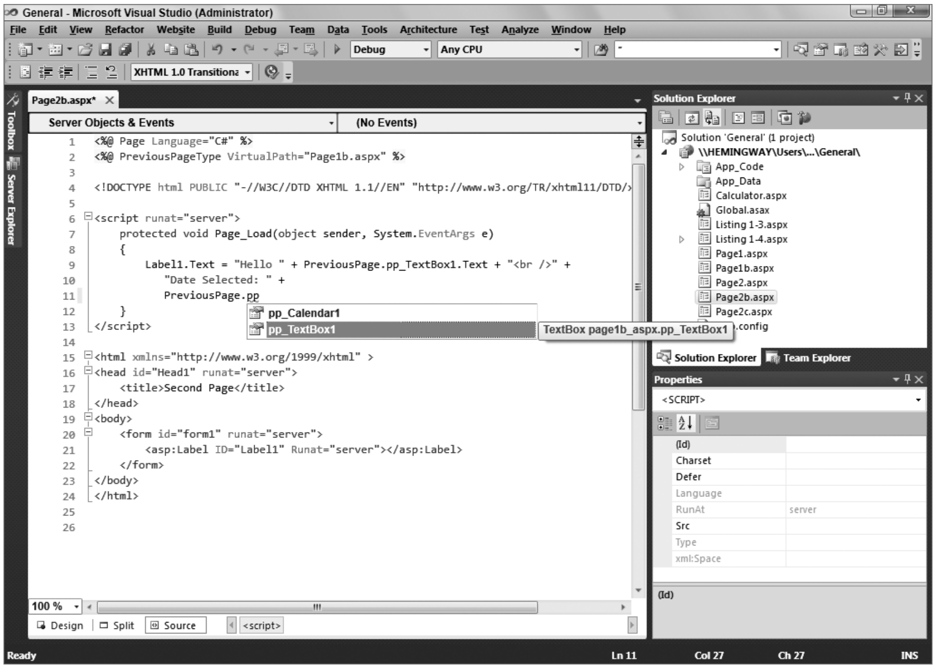
```
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & PreviousPage.pp_Textbox1.Text & "<br />" &
            "Date Selected: " &
            PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString()
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + PreviousPage.pp_TextBox1.Text + "<br />" +
           "Date Selected: " +
           PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```

Filename Page2b.aspx

To be able to work with the properties that `Page1.aspx` exposes, you have to strongly type the `PreviousPage` property to `Page1.aspx`. To do this, you use the `PreviousPageType` directive. This directive allows you to specifically point to `Page1.aspx` with the use of the `VirtualPath` attribute. When that is in place, notice that you can see the properties that `Page1.aspx` exposes through IntelliSense from the `PreviousPage` property. This is illustrated in Figure 1-7.



```
<%@ Page Language="C#" %>
<%@ PreviousPageType VirtualPath="Page1b.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/>

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + PreviousPage.pp_TextBox1.Text + "<br />" +
           "Date Selected: " +
           PreviousPage.pp
    }              pp_Calendar1
</script>          pp_TextBox1     TextBox page1b_aspx.pp_TextBox1

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Second Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" Runat="server"></asp:Label>
    </form>
</body>
</html>
```

FIGURE 1-7

As you can see, working with cross-page posting is straightforward. Notice that when you are cross posting from one page to another, you are not restricted to working only with the postback on the second page. In fact, you can still create methods on `Page1.aspx` that work with the postback before moving onto `Page2.aspx`. To do this, you simply add an `OnClick` event for the button in `Page1.aspx` and a method. You also assign a value for the `PostBackUrl` property. You can then work with the postback on `Page1.aspx` and then again on `Page2.aspx`.

What happens if someone requests `Page2.aspx` before she works her way through `Page1.aspx`? Determining whether the request is coming from `Page1.aspx` or whether someone just hit `Page2.aspx` directly is actually quite easy. You can work with the request through the use of the `IsCrossPagePostBack` property that is quite similar to the `IsPostBack` property from ASP.NET 1.0/1.1. The `IsCrossPagePostBack` property enables you to check whether the request is from `Page1.aspx`. Listing 1-13 shows an example of this.

LISTING 1-13: Using the IsCrossPagePostBack property

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
      If Not PreviousPage Is Nothing AndAlso PreviousPage.IsCrossPagePostBack Then
        Label1.Text = "Hello " & PreviousPage.pp_Textbox1.Text & "<br />" &
            "Date Selected: " &
            PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString()
      Else
        Response.Redirect("Page1.aspx")
      End If
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
      if (PreviousPage != null && PreviousPage.IsCrossPagePostBack) {
        Label1.Text = "Hello " + PreviousPage.pp_TextBox1.Text + "<br />" +
           "Date Selected: " +
           PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString();
      }
      else
      {
         Response.Redirect("Page1.aspx");
      }
    }
</script>
```

Filename Page2c.aspx

ASP.NET APPLICATION FOLDERS

When you create ASP.NET applications, notice that ASP.NET 4 uses a file-based approach. When working with ASP.NET, you can add as many files and folders as you want within your application without recompiling each and every time a new file is added to the overall solution. ASP.NET 4 includes the capability to automatically precompile your ASP.NET applications dynamically.

ASP.NET 1.0/1.1 compiled everything in your solution into a DLL. This is no longer necessary because ASP.NET applications now have a defined folder structure. By using the ASP.NET-defined folders, you can have your code automatically compiled for you, your application themes accessible throughout your application, and your globalization resources available whenever you need them. Look at each of these defined folders to see how they work. The first folder reviewed is the App_Code folder.

App_Code Folder

The App_Code folder is meant to store your classes, `.wsdl` files, and typed datasets. Any of these items stored in this folder are then automatically available to all the pages within your solution. The nice thing about the App_Code folder is that when you place something inside this folder, Visual Studio 2010 automatically detects this and compiles it if it is a class (`.vb` or `.cs`), automatically creates your XML Web service proxy class (from the `.wsdl` file), or automatically creates a typed dataset for you from your `.xsd` files. After the files are automatically compiled, these items are then instantaneously available to any of your ASP.NET pages that are in the same solution. Look at how to employ a simple class in your solution using the App_Code folder.

The first step is to create an App_Code folder. To do this, simply right-click the solution and choose Add ASP.NET Folder ➪ App_Code. Right away, you will notice that Visual Studio 2010 treats this folder differently than the other folders in your solution. The App_Code folder is shown in a different color (gray) with a document pictured next to the folder icon. See Figure 1-8.

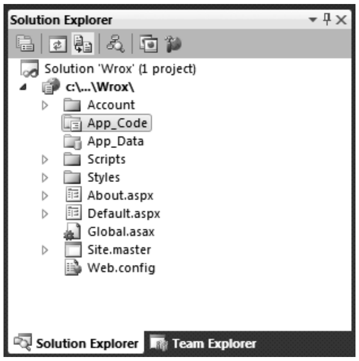


FIGURE 1-8

After the App_Code folder is in place, right-click the folder and select Add New Item. The Add New Item dialog that appears gives you a few options for the types of files that you can place within this folder. The available options include an AJAX-enabled WCF Service, a Class file, a LINQ to SQL Class, an ADO.NET Entity Data Model, an ADO.NET EntityObject Generator, a Sequence Diagram, a Text Template, a Text file, a DataSet, a Report, and a Class Diagram if you are using Visual Studio 2010. Visual Web Developer 2010 Express Edition offers only a subset of these files. For the first example, select the file of type Class and name the class `Calculator.vb` or `Calculator.cs`. Listing 1-14 shows how the `Calculator` class should appear.

LISTING 1-14: The Calculator class

Available for download on Wrox.com

VB

```
Imports Microsoft.VisualBasic

Public Class Calculator
    Public Function Add(ByVal a As Integer, ByVal b As Integer) As Integer
        Return (a + b)
    End Function
End Class
```

C#

```
using System;

public class Calculator
{
    public int Add(int a, int b)
        {
            return (a + b);
    }
}
```

Filenames Calculator.vb and Calculator.cs

What's next? Just save this file, and it is now available to use in any pages that are in your solution. To see this in action, create a simple `.aspx` page that has just a single Label server control. Listing 1-15 shows you the code to place within the `Page_Load` event to make this new class available to the page.

LISTING 1-15: An .aspx page that uses the Calculator class

Available for download on Wrox.com

VB

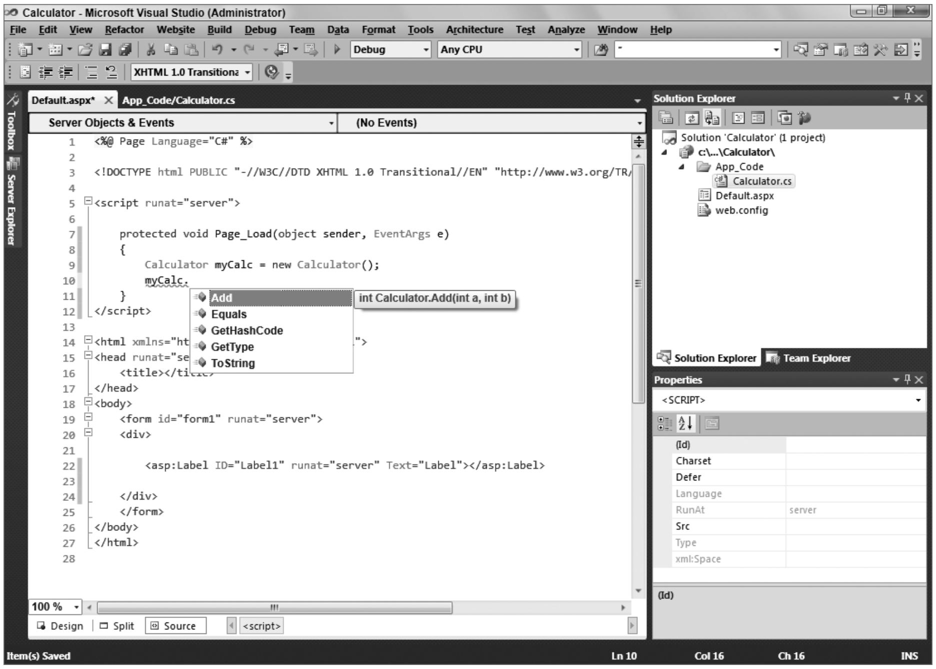
```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myCalc As New Calculator
        Label1.Text = myCalc.Add(12, 12)
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    protected void Page_Load(object sender, System.EventArgs e)
    {
        Calculator myCalc = new Calculator();
        Label1.Text = myCalc.Add(12, 12).ToString();
    }
</script>
```

Filename Calculator.aspx

When you run this .aspx page, notice that it utilizes the Calculator class without any problem, with no need to compile the class before use. In fact, right after saving the Calculator class in your solution or moving the class to the App_Code folder, you also instantaneously receive IntelliSense capability on the methods that the class exposes (as illustrated in Figure 1-9).



```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        Calculator myCalc = new Calculator();
        myCalc.
    }
</script>

<html xmlns="ht                                  ">
<head runat="se
    <title></ti
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

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

FIGURE 1-9

To see how Visual Studio 2010 works with the App_Code folder, open the `Calculator` class again in the IDE and add a `Subtract` method. Your class should now appear as shown in Listing 1-16.

LISTING 1-16: Adding a Subtract method to the Calculator class

Available for download on Wrox.com

VB

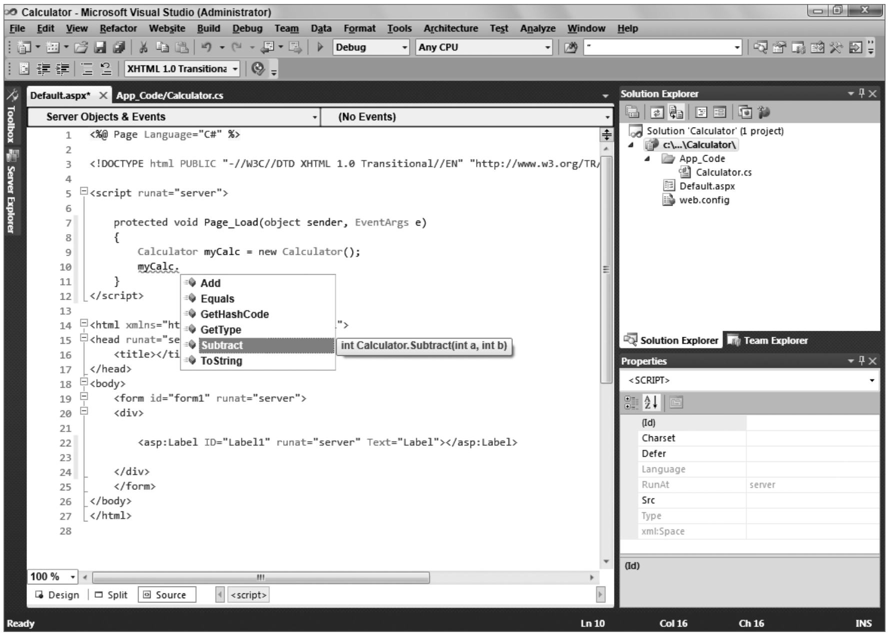
```
Imports Microsoft.VisualBasic

Public Class Calculator
    Public Function Add(ByVal a As Integer, ByVal b As Integer) As Integer
        Return (a + b)
    End Function

    Public Function Subtract(ByVal a As Integer, ByVal b As Integer) As Integer
        Return (a - b)
    End Function
End Class
```

C#

```
using System;

public class Calculator
{
    public int Add(int a, int b)
    {
            return (a + b);
    }

    public int Subtract(int a, int b)
    {
            return (a - b);
    }
}
```

Filenames Calculator.vb and Calculator.cs

After you have added the `Subtract` method to the `Calculator` class, save the file and go back to your `.aspx` page. Notice that the class has been recompiled by the IDE, and the new method is now available to your page. You see this directly in IntelliSense. Figure 1-10 shows this in action.



```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        Calculator myCalc = new Calculator();
        myCalc.
    }
</script>

<html xmlns="ht                                  ">
<head runat="se
    <title></ti
</head>
<body>
    <form id="form1" runat="server">
    <div>

        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

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

FIGURE 1-10

Everything placed in the App_Code folder is compiled into a single assembly. The class files placed within the App_Code folder are not required to use a specific language. This means that even if all the pages of the solution are written in Visual Basic 2010, the `Calculator` class in the App_Code folder of the solution can be built in C# (`Calculator.cs`).

Because all the classes contained in this folder are built into a single assembly, you *cannot* have classes of different languages sitting in the root App_Code folder, as in the following example:

```
\App_Code
   Calculator.cs
   AdvancedMath.vb
```

Having two classes made up of different languages in the App_Code folder (as shown here) causes an error to be thrown. It is impossible for the assigned compiler to work with two different languages. Therefore, to be able to work with multiple languages in your App_Code folder, you must make some changes to the folder structure and to the `web.config` file.

The first step is to add two new subfolders to the App_Code folder — a VB folder and a CS folder. This gives you the following folder structure:

```
\App_Code
   \VB
      Add.vb
   \CS
      Subtract.cs
```

This still will not correctly compile these class files into separate assemblies, at least not until you make some additions to the `web.config` file. Most likely, you do not have a `web.config` file in your solution at this moment, so add one through the Solution Explorer. After it is added, change the `<compilation>` node so that it is structured as shown in Listing 1-17.

LISTING 1-17: Structuring the web.config file so that classes in the App_Code folder can use different languages

```
<compilation>
   <codeSubDirectories>
      <add directoryName="VB"></add>
      <add directoryName="CS"></add>
   </codeSubDirectories>
</compilation>
```

Now that this is in place in your `web.config` file, you can work with each of the classes in your ASP.NET pages. In addition, any C# class placed in the CS folder is now automatically compiled just like any of the classes placed in the VB folder. Because you can add these directories in the `web.config` file, you are not required to name them VB and CS as we did; you can use whatever name tickles your fancy.

App_Data Folder

The App_Data folder holds the data stores utilized by the application. It is a good spot to centrally store all the data stores your application might use. The App_Data folder can contain Microsoft SQL Express files (`.mdf` files), Microsoft Access files (`.mdb` files), XML files, and more.

The user account utilized by your application will have read and write access to any of the files contained within the App_Data folder. By default, this is the ASPNET account. Another reason for storing all your data files in this folder is that much of the ASP.NET system — from the membership and role management systems to the GUI tools, such as the ASP.NET MMC snap-in and ASP.NET Web Site Administration Tool — is built to work with the App_Data folder.

App_Themes Folder

Themes are a way of providing a common look-and-feel to your site across every page. You implement a theme by using a `.skin` file, CSS files, and images used by the server controls of your site. All these elements can make a *theme*, which is then stored in the App_Themes folder of your solution. By storing these elements within the App_Themes folder, you ensure that all the pages within the solution can take advantage of the theme and easily apply its elements to the controls and markup of the page. Themes are discussed in great detail in Chapter 6 of this book.

App_GlobalResources Folder

Resource files are string tables that can serve as data dictionaries for your applications when these applications require changes to content based on things such as changes in culture. You can add Assembly Resource Files (`.resx`) to the App_GlobalResources folder, and they are dynamically compiled and made part of the solution for use by all your `.aspx` pages in the application. When using ASP.NET 1.0/1.1, you had to use the `resgen.exe` tool and had to compile your resource files to a `.dll` or `.exe` for use within your solution. Dealing with resource files in ASP.NET 4 is considerably easier. Simply placing your application-wide resources in this folder makes them instantly accessible. Localization is covered in detail in Chapter 32.

App_LocalResources Folder

Even if you are not interested in constructing application-wide resources using the App_GlobalResources folder, you may want resources that can be used for a single `.aspx` page. You can do this very simply by using the App_LocalResources folder.

You can add resource files that are page-specific to the App_LocalResources folder by constructing the name of the `.resx` file in the following manner:

- `Default.aspx.resx`
- `Default.aspx.fi.resx`
- `Default.aspx.ja.resx`
- `Default.aspx.en-gb.resx`

Now, the resource declarations used on the `Default.aspx` page are retrieved from the appropriate file in the App_LocalResources folder. By default, the `Default.aspx.resx` resource file is used if another match is not found. If the client is using a culture specification of `fi-FI` (Finnish), however, the `Default.aspx.fi.resx` file is used instead. Localization of local resources is covered in detail in Chapter 32.

App_WebReferences Folder

The App_WebReferences folder is a new name for the previous Web References folder that was used in versions of ASP.NET prior to ASP.NET 3.5. Now you can use the App_WebReferences folder and have automatic access to the remote Web services referenced from your application. Chapter 31 covers Web services in ASP.NET.

App_Browsers Folder

The App_Browsers folder holds `.browser` files, which are XML files used to identity the browsers making requests to the application and understanding the capabilities these browsers have. You can find a list of globally accessible `.browser` files at `C:\Windows\Microsoft.NET\Framework\v4.0.xxxxx\Config\Browsers`. In addition, if you want to change any part of these default browser definition files, just copy the appropriate `.browser` file from the Browsers folder to your application's App_Browsers folder and change the definition.

COMPILATION

You already saw how Visual Studio 2010 compiles pieces of your application as you work with them (for instance, by placing a class in the App_Code folder). The other parts of the application, such as the `.aspx` pages, can be compiled just as they were in earlier versions of ASP.NET by referencing the pages in the browser.

When an ASP.NET page is referenced in the browser for the first time, the request is passed to the ASP.NET parser that creates the class file in the language of the page. It is passed to the ASP.NET parser based on the file's extension (`.aspx`) because ASP.NET realizes that this file extension type is meant for its handling and processing. After the class file has been created, the class file is compiled into a DLL and then written to the disk of the Web server. At this point, the DLL is instantiated and processed, and an output is generated for the initial requester of the ASP.NET page. This is detailed in Figure 1-11.

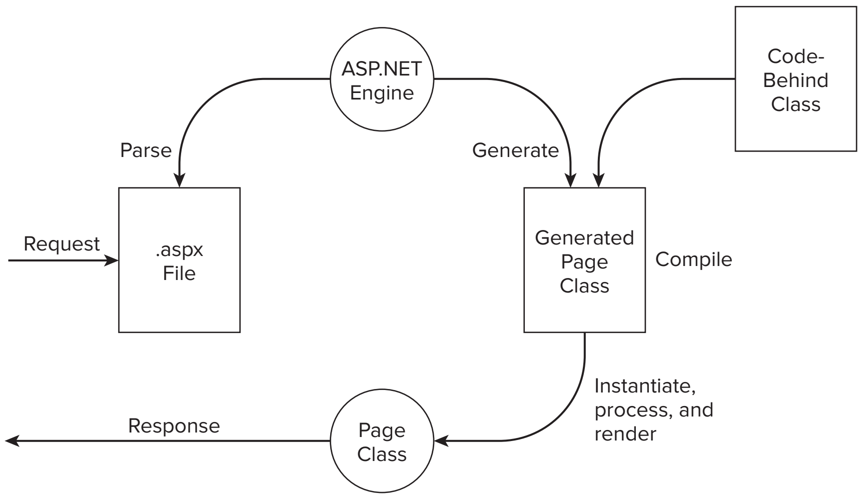


FIGURE 1-11

On the next request, great things happen. Instead of going through the entire process again for the second and respective requests, the request simply causes an instantiation of the already-created DLL, which sends out a response to the requester. This is illustrated in Figure 1-12.

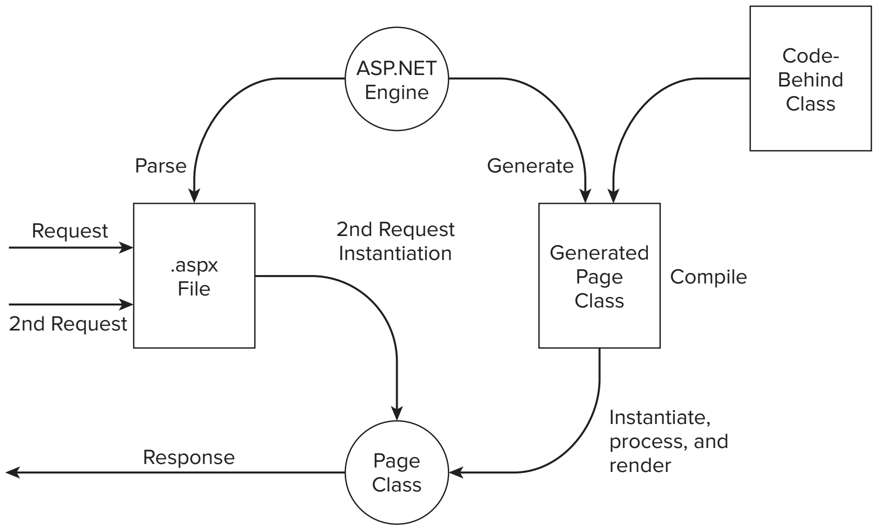


FIGURE 1-12

Because of the mechanics of this process, if you made changes to your `.aspx` code-behind pages, you found it necessary to recompile your application. This was quite a pain if you had a larger site and did not want your end users to experience the extreme lag that occurs when an `.aspx` page is referenced for the first time after compilation. Many developers, consequently, began to develop their own tools that automatically go out and hit every single page within their application to remove this first-time lag hit from the end user's browsing experience.

ASP.NET provides a few ways to precompile your entire application with a single command that you can issue through a command line. One type of compilation is referred to as *in-place precompilation*. To precompile your entire ASP.NET application, you must use the `aspnet_compiler.exe` tool that comes with ASP.NET. You navigate to the tool using the Command window. Open the Command window and navigate to `C:\Windows\Microsoft.NET\Framework\v4.0.xxxxx\`. When you are there, you can work with the `aspnet_compiler` tool. You can also get to this tool directly from the Visual Studio 2010 Command Prompt. Choose Start ➪ All Programs ➪ Microsoft Visual Studio 2010 ➪ Visual Studio Tools ➪ Visual Studio Command Prompt (2010).

After you get the command prompt, you use the `aspnet_compiler.exe` tool to perform an in-place precompilation using the following command:

```
aspnet_compiler -p "C:\Inetpub\wwwroot\WROX" -v none
```

You then get a message stating that the precompilation is successful. The other great thing about this precompilation capability is that you can also use it to find errors on any of the ASP.NET pages in your application. Because it hits each and every page, if one of the pages contains an error that won't be triggered until runtime, you get notification of the error immediately as you employ this precompilation method.

The next precompilation option is commonly referred to as *precompilation for deployment*. This outstanding capability of ASP.NET enables you to compile your application down to some DLLs, which can then be deployed to customers, partners, or elsewhere for your own use. Not only are minimal steps required to do this, but also after your application is compiled, you simply have to move around the DLL and some placeholder files for the site to work. This means that your Web site code is completely removed and placed in the DLL when deployed.

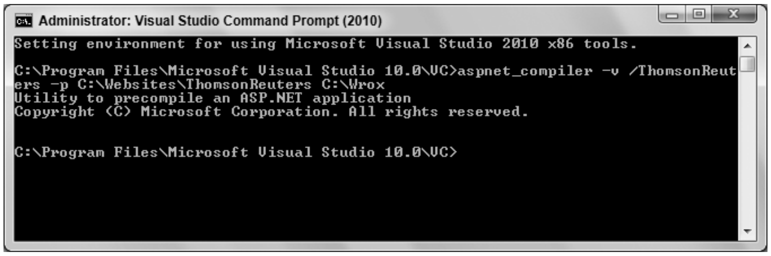
However, before you take these precompilation steps, create a folder in your root drive called, for example, Wrox. This folder is the one to which you will direct the compiler output. When it is in place, you can return to the compiler tool and give the following command:

```
aspnet_compiler -v [Application Name] -p [Physical Location] [Target]
```

Therefore, if you have an application called `ThomsonReuters` located at `C:\Websites\ThomsonReuters`, you use the following commands:

```
aspnet_compiler -v /ThomsonReuters -p C:\Websites\ThomsonReuters C:\Wrox
```

Press the Enter key, and the compiler either tells you that it has a problem with one of the command parameters or that it was successful (shown in Figure 1-13). If it was successful, you can see the output placed in the target directory.



```
Setting environment for using Microsoft Visual Studio 2010 x86 tools.

C:\Program Files\Microsoft Visual Studio 10.0\VC>aspnet_compiler -v /ThomsonReut
ers -p C:\Websites\ThomsonReuters C:\Wrox
Utility to precompile an ASP.NET application
Copyright (C) Microsoft Corporation. All rights reserved.


C:\Program Files\Microsoft Visual Studio 10.0\VC>
```

FIGURE 1-13

In the example just shown, `-v` is a command for the virtual path of the application, which is provided by using `/ThomsonReuters`. The next command is `-p`, which is pointing to the physical path of the application.

In this case, it is `C:\Websites\ThomsonReuters`. Finally, the last bit, `C:\Wrox`, is the location of the compiler output. Table 1-9 describes some of the possible commands for the `aspnet_compiler.exe` tool.

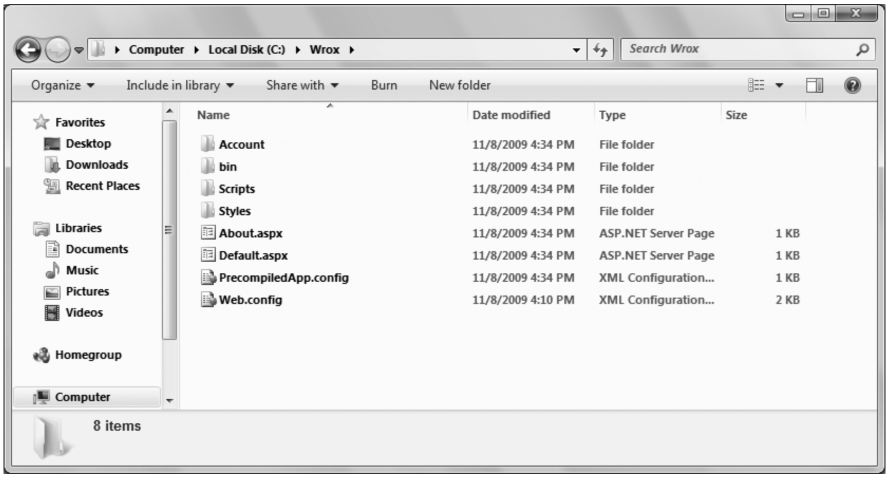
TABLE 1-9

COMMAND	DESCRIPTION
`-m`	Specifies the full IIS metabase path of the application. If you use the `-m` command, you cannot use the `-v` or `-p` command.
`-v`	Specifies the virtual path of the application to be compiled. If you also use the `-p` command, the physical path is used to find the location of the application.
`-p`	Specifies the physical path of the application to be compiled. If this is not specified, the IIS metabase is used to find the application.
`-u`	If this command is utilized, it specifies that the application is updatable.
`-f`	Specifies to overwrite the target directory if it already exists.
`-d`	Specifies that the debug information should be excluded from the compilation process.
`[targetDir]`	Specifies the target directory where the compiled files should be placed. If this is not specified, the output files are placed in the application directory.

After compiling the application, you can go to `C:\Wrox` to see the output. Here you see all the files and the file structures that were in the original application. However, if you look at the content of one of the files, notice that the file is simply a placeholder. In the actual file, you find the following comment:

```
This is a marker file generated by the precompilation tool
and should not be deleted!
```

In fact, you find a `Code.dll` file in the bin folder where all the page code is located. Because it is in a DLL file, it provides great code obfuscation as well. From here on, all you do is move these files to another server using FTP or Windows Explorer, and you can run the entire Web application from these files. When you have an update to the application, you simply provide a new set of compiled files. Figure 1-14 shows a sample output.



Name	Date modified	Type	Size
Account	11/8/2009 4:34 PM	File folder	
bin	11/8/2009 4:34 PM	File folder	
Scripts	11/8/2009 4:34 PM	File folder	
Styles	11/8/2009 4:34 PM	File folder	
About.aspx	11/8/2009 4:34 PM	ASP.NET Server Page	1 KB
Default.aspx	11/8/2009 4:34 PM	ASP.NET Server Page	1 KB
PrecompiledApp.config	11/8/2009 4:34 PM	XML Configuration...	1 KB
Web.config	11/8/2009 4:10 PM	XML Configuration...	2 KB

FIGURE 1-14

Note that this compilation process does not compile *every* type of Web file. In fact, it compiles only the ASP.NET-specific file types and leaves out of the compilation process the following types of files:

➤ HTML files
➤ XML files
➤ XSD files

- ➤ `web.config` files
- ➤ Text files

You cannot do much to get around this, except in the case of the HTML files and the text files. For these file types, just change the file extensions of these file types to `.aspx`; they are then compiled into the `Code.dll` like all the other ASP.NET files.

BUILD PROVIDERS

As you review the various ASP.NET folders, note that one of the more interesting folders is the App_Code folder. You can simply drop code files, XSD files, and even WSDL files directly into the folder for automatic compilation. When you drop a class file into the App_Code folder, the class can automatically be utilized by a running application. In the early days of ASP.NET, if you wanted to deploy a custom component, you had to precompile the component before being able to utilize it within your application. Now ASP.NET simply takes care of all the work that you once had to do. You do not need to perform any compilation routine.

Which file types are compiled in the App_Code folder? As with most things in ASP.NET, this is determined through settings applied in a configuration file. Listing 1-18 shows a snippet of configuration code taken from the master `web.config` file found in ASP.NET 4.

LISTING 1-18: Reviewing the list of build providers

```
<compilation>
   <buildProviders>
      <add extension=".aspx" type="System.Web.Compilation.PageBuildProvider" />
      <add extension=".ascx"
       type="System.Web.Compilation.UserControlBuildProvider" />
      <add extension=".master"
       type="System.Web.Compilation.MasterPageBuildProvider" />
      <add extension=".asmx"
       type="System.Web.Compilation.WebServiceBuildProvider" />
      <add extension=".ashx"
       type="System.Web.Compilation.WebHandlerBuildProvider" />
      <add extension=".soap"
       type="System.Web.Compilation.WebServiceBuildProvider" />
      <add extension=".resx" type="System.Web.Compilation.ResXBuildProvider" />
      <add extension=".resources"
       type="System.Web.Compilation.ResourcesBuildProvider" />
      <add extension=".wsdl" type="System.Web.Compilation.WsdlBuildProvider" />
      <add extension=".xsd" type="System.Web.Compilation.XsdBuildProvider" />
      <add extension=".js" type="System.Web.Compilation.ForceCopyBuildProvider" />
      <add extension=".lic"
       type="System.Web.Compilation.IgnoreFileBuildProvider" />
      <add extension=".licx"
       type="System.Web.Compilation.IgnoreFileBuildProvider" />
      <add extension=".exclude"
       type="System.Web.Compilation.IgnoreFileBuildProvider" />
      <add extension=".refresh"
       type="System.Web.Compilation.IgnoreFileBuildProvider" />
      <add extension=".edmx"
       type="System.Data.Entity.Design.AspNet.
          EntityDesignerBuildProvider" />
      <add extension=".xoml" type="System.ServiceModel.Activation.
          WorkflowServiceBuildProvider, System.WorkflowServices,
          Version=4.0.0.0, Culture=neutral,
          PublicKeyToken=31bf3856ad364e35" />
      <add extension=".svc"
       type="System.ServiceModel.Activation.ServiceBuildProvider,
          System.ServiceModel.Activation, Version=4.0.0.0,
          Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
```

continues

LISTING 1-18 *(continued)*

```
      <add extension=".xamlx"
       type="System.Xaml.Hosting.XamlBuildProvider,
          System.Xaml.Hosting, Version=4.0.0.0, Culture=neutral,
          PublicKeyToken=31bf3856ad364e35" />
   </buildProviders>
</compilation>
```

This section contains a list of build providers that can be used by two entities in your development cycle. The build provider is first used is during development when you are building your solution in Visual Studio 2010. For instance, placing a `.wsdl` file in the App_Code folder during development in Visual Studio causes the IDE to give you automatic access to the dynamically compiled proxy class that comes from this `.wsdl` file. The other entity that uses the build providers is ASP.NET itself. As stated, simply dragging and dropping a `.wsdl` file in the App_Code folder of a deployed application automatically gives the ASP.NET application access to the created proxy class.

A build provider is simply a class that inherits from `System.Web.Compilation.BuildProvider`. The `<buildProviders>` section in the `web.config` file allows you to list the build provider classes that will be utilized. The capability to dynamically compile any WSDL file is defined by the following line in the configuration file.

```
<add extension=".wsdl" type="System.Web.Compilation.WsdlBuildProvider" />
```

This means that any file utilizing the `.wsdl` file extension is compiled using the `WsdlBuildProvider`, a class that inherits from `BuildProvider`. Microsoft provides a set number of build providers out of the box for you to use. As you can see from the set in Listing 1-18, a number of providers are available in addition to the `WsdlBuildProvider`, including providers such as the `XsdBuildProvider`, `PageBuildProvider`, `UserControlBuildProvider`, `MasterPageBuildProvider`, and more. Just by looking at the names of some of these providers you can pretty much understand what they are about. The next section, however, reviews some other providers whose names might not ring a bell right away.

Using the Built-in Build Providers

Two of the providers that this section covers are the `ForceCopyBuildProvider` and the `IgnoreFileBuildProvider`, both of which are included in the default list of providers.

The `ForceCopyBuildProvider` is basically a provider that copies only those files for deployment that use the defined extension. (These files are not included in the compilation process.) An extension that utilizes the `ForceCopyBuildProvider` is shown in the predefined list in Listing 1-18. This is the `.js` file type (a JavaScript file extension). Any `.js` files are simply copied and not included in the compilation process (which makes sense for JavaScript files). You can add other file types that you want to be a part of this copy process with the command shown here:

```
<add extension=".chm" type="System.Web.Compilation.ForceCopyBuildProvider" />
```

In addition to the `ForceCopyBuildProvider`, you should also be aware of the `IgnoreFileBuildProvider` class. This provider causes the defined file type to be ignored in the deployment or compilation process. This means that any file type defined with `IgnoreFileBuildProvider` is simply ignored. Visual Studio will not copy, compile, or deploy any file of that type. So, if you are including Visio diagrams in your project, you can simply add the following `<add>` element to the `web.config` file to have this file type ignored. An example is presented here:

```
<add extension=".vsd" type="System.Web.Compilation.IgnoreFileBuildProvider" />
```

With this in place, all `.vsd` files are ignored.

Using Your Own Build Providers

In addition to using the predefined build providers out of the box, you can also take this build provider stuff one step further and construct your own custom build providers to use within your applications.

For example, suppose you wanted to construct a `Car` class dynamically based upon settings applied in a custom `.car` file that you have defined. You might do this because you are using this `.car` definition file in multiple projects or many times within the same project. Using a build provider makes defining these multiple instances of the `Car` class simpler.

Listing 1-19 presents an example of the `.car` file type.

LISTING 1-19: An example of a .car file

Available for download on Wrox.com

```
<?xml version="1.0" encoding="utf-8" ?>
<car name="EvjenCar">
  <color>Blue</color>
  <door>4</door>
  <speed>150</speed>
</car>
```

Filename Evjen.car

In the end, this XML declaration specifies the name of the class to compile as well as some values for various properties and a method. These elements make up the class. Now that you understand the structure of the `.car` file type, the next step is to construct the build provider. To accomplish this task, create a new Class Library project in the language of your choice within Visual Studio. Name the project `CarBuildProvider`. The `CarBuildProvider` contains a single class — `Car.vb` or `Car.cs`. This class inherits from the base class `BuildProvider` and overrides the `GenerateCode()` method of the `BuildProvider` class. Listing 1-20 presents this class.

LISTING 1-20: The CarBuildProvider

Available for download on Wrox.com

VB

```
Imports System.IO
Imports System.Web.Compilation
Imports System.Xml
Imports System.CodeDom

Public Class Car
    Inherits BuildProvider

    Public Overrides Sub GenerateCode(ByVal myAb As AssemblyBuilder)
        Dim carXmlDoc As XmlDocument = New XmlDocument()

        Using passedFile As Stream = Me.OpenStream()
            carXmlDoc.Load(passedFile)
        End Using

        Dim mainNode As XmlNode = carXmlDoc.SelectSingleNode("/car")
        Dim selectionMainNode As String = mainNode.Attributes("name").Value

        Dim colorNode As XmlNode = carXmlDoc.SelectSingleNode("/car/color")
        Dim selectionColorNode As String = colorNode.InnerText

        Dim doorNode As XmlNode = carXmlDoc.SelectSingleNode("/car/door")
        Dim selectionDoorNode As String = doorNode.InnerText

        Dim speedNode As XmlNode = carXmlDoc.SelectSingleNode("/car/speed")
        Dim selectionSpeedNode As String = speedNode.InnerText

        Dim ccu As CodeCompileUnit = New CodeCompileUnit()
        Dim cn As CodeNamespace = New CodeNamespace()
        Dim cmp1 As CodeMemberProperty = New CodeMemberProperty()
        Dim cmp2 As CodeMemberProperty = New CodeMemberProperty()
        Dim cmm1 As CodeMemberMethod = New CodeMemberMethod()
```

continues

LISTING 1-20 *(continued)*

```
        cn.Imports.Add(New CodeNamespaceImport("System"))

        cmp1.Name = "Color"
        cmp1.Type = New CodeTypeReference(GetType(System.String))
        cmp1.Attributes = MemberAttributes.Public
        cmp1.GetStatements.Add(New CodeSnippetExpression("return """ &
           selectionColorNode & """"))

        cmp2.Name = "Doors"
        cmp2.Type = New CodeTypeReference(GetType(System.Int32))
        cmp2.Attributes = MemberAttributes.Public
        cmp2.GetStatements.Add(New CodeSnippetExpression("return " &
           selectionDoorNode))

        cmm1.Name = "Go"
        cmm1.ReturnType = New CodeTypeReference(GetType(System.Int32))
        cmm1.Attributes = MemberAttributes.Public
        cmm1.Statements.Add(New CodeSnippetExpression("return " &
           selectionSpeedNode))

        Dim ctd As CodeTypeDeclaration = New CodeTypeDeclaration(selectionMainNode)
        ctd.Members.Add(cmp1)
        ctd.Members.Add(cmp2)
        ctd.Members.Add(cmm1)

        cn.Types.Add(ctd)
        ccu.Namespaces.Add(cn)

        myAb.AddCodeCompileUnit(Me, ccu)
    End Sub

End Class
```

C#

```
using System.IO;
using System.Web.Compilation;
using System.Xml;
using System.CodeDom;

namespace CarBuildProvider
{
    class Car : BuildProvider
    {
        public override void GenerateCode(AssemblyBuilder myAb)
        {
            XmlDocument carXmlDoc = new XmlDocument();

            using (Stream passedFile = OpenStream())
            {
                carXmlDoc.Load(passedFile);
            }
            XmlNode mainNode = carXmlDoc.SelectSingleNode("/car");
            string selectionMainNode = mainNode.Attributes["name"].Value;

            XmlNode colorNode = carXmlDoc.SelectSingleNode("/car/color");
            string selectionColorNode = colorNode.InnerText;

            XmlNode doorNode = carXmlDoc.SelectSingleNode("/car/door");
            string selectionDoorNode = doorNode.InnerText;

            XmlNode speedNode = carXmlDoc.SelectSingleNode("/car/speed");
            string selectionSpeedNode = speedNode.InnerText;
```

```
                CodeCompileUnit ccu = new CodeCompileUnit();
                CodeNamespace cn = new CodeNamespace();
                CodeMemberProperty cmp1 = new CodeMemberProperty();
                CodeMemberProperty cmp2 = new CodeMemberProperty();
                CodeMemberMethod cmm1 = new CodeMemberMethod();

                cn.Imports.Add(new CodeNamespaceImport("System"));

                cmp1.Name = "Color";
                cmp1.Type = new CodeTypeReference(typeof(string));
                cmp1.Attributes = MemberAttributes.Public;
                cmp1.GetStatements.Add(new CodeSnippetExpression("return \"" +
                   selectionColorNode + "\""));

                cmp2.Name = "Doors";
                cmp2.Type = new CodeTypeReference(typeof(int));
                cmp2.Attributes = MemberAttributes.Public;
                cmp2.GetStatements.Add(new CodeSnippetExpression("return " +
                   selectionDoorNode));

                cmm1.Name = "Go";
                cmm1.ReturnType = new CodeTypeReference(typeof(int));
                cmm1.Attributes = MemberAttributes.Public;
                cmm1.Statements.Add(new CodeSnippetExpression("return " +
                   selectionSpeedNode));

                CodeTypeDeclaration ctd = new CodeTypeDeclaration(selectionMainNode);
                ctd.Members.Add(cmp1);
                ctd.Members.Add(cmp2);
                ctd.Members.Add(cmm1);

                cn.Types.Add(ctd);
                ccu.Namespaces.Add(cn);

                myAb.AddCodeCompileUnit(this, ccu);
            }
        }
    }
```

Filenames Car.vb and Car.cs

As you look over the `GenerateCode()` method, you can see that it takes an instance of `AssemblyBuilder`. This `AssemblyBuilder` object is from the `System.Web.Compilation` namespace and, because of this, your Class Library project must have a reference to the `System.Web` assembly. With all the various objects used in this `Car` class, you also have to import in the following namespaces:

```
Imports System.IO
Imports System.Web.Compilation
Imports System.Xml
Imports System.CodeDom
```

When you have done this, one of the tasks remaining in the `GenerateCode()` method is loading the `.car` file. Because the `.car` file is using XML for its form, you are able to load the document easily using the `XmlDocument` object. From there, by using the CodeDom, you can create a class that contains two properties and a single method dynamically. The class that is generated is an abstract representation of what is defined in the provided `.car` file. On top of that, the name of the class is also dynamically driven from the value provided via the name attribute used in the main `<Car>` node of the `.car` file.

The `AssemblyBuilder` instance that is used as the input object then compiles the generated code along with everything else into an assembly.

What does it mean that your ASP.NET project has a reference to the `CarBuildProvider` assembly in its project? It means that you can create a `.car` file of your own definition and drop this file into the

App_Code folder. The second you drop the file into the App_Code folder, you have instant programmatic access to the definition specified in the file.

To see this in action, you need a reference to the build provider in either the server's `machine.config` or your application's `web.config` file. A reference is shown in Listing 1-21.

LISTING 1-21: Making a reference to the build provider in the web.config file

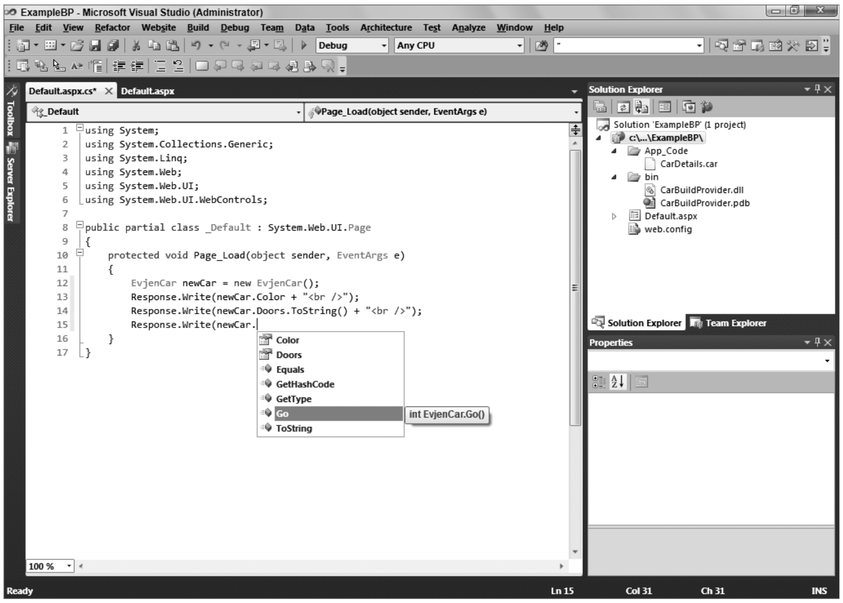
Available for download on Wrox.com

```
<configuration>
   <system.web>
      <compilation debug="false">
         <buildProviders>
            <add extension=".car" type="CarBuildProvider.Car"/>
         </buildProviders>
      </compilation>
   </system.web>
</configuration>
```

The `<buildProviders>` element is a child element of the `<compilation>` element. The `<buildProviders>` element takes a couple of child elements to add or remove providers. In this case, because you want to add a reference to the custom `CarBuildProvider` object, you use the `<add>` element. The `<add>` element can take two possible attributes — `extension` and `type`. You must use both of these attributes. In the `extension` attribute, you define the file extension that this build provider will be associated with. In this case, you use the `.car` file extension. This means that any file using this file extension is associated with the class defined in the `type` attribute. The `type` attribute then takes a reference to the `CarBuildProvider` class that you built — `CarBuildProvider.Car`.

With this reference in place, you can create the `.car` file that was shown earlier in Listing 1-19. Place the created `.car` file in the App_Code folder. You instantly have access to a dynamically generated class that comes from the definition provided via the file. For example, because I used `EvjenCar` as the value of the name attribute in the `<Car>` element, this will be the name of the class generated, and I will find this exact name in IntelliSense as I type in Visual Studio.

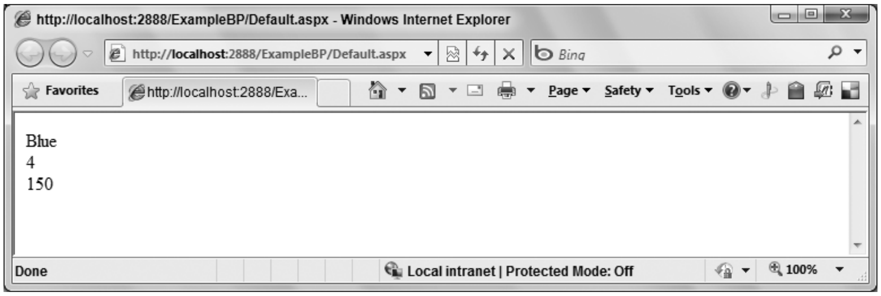
If you create an instance of the `EvjenCar` class, you also find that you have access to the properties and the method that this class exposes. This is shown in Figure 1-15.



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

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        EvjenCar newCar = new EvjenCar();
        Response.Write(newCar.Color + "<br />");
        Response.Write(newCar.Doors.ToString() + "<br />");
        Response.Write(newCar.
    }
}
```

FIGURE 1-15

In addition to getting access to the properties and methods of the class, you also gain access to the values that are defined in the `.car` file. This is shown in Figure 1-16. The simple code example shown in Figure 1-15 is used for this browser output.


http://localhost:2888/ExampleBP/Default.aspx - Windows Internet Explorer
http://localhost:2888/ExampleBP/Default.aspx
Bing
Favorites
http://localhost:2888/Exa...
Page
Safety
Tools
Done
Local intranet | Protected Mode: Off
100%

```
Blue
4
150
```

FIGURE 1-16

Although a `Car` class is not the most useful thing in the world, this example shows you how to take the build provider mechanics into your own hands to extend your application's capabilities.

GLOBAL.ASAX

If you add a new item to your ASP.NET application, you get the Add New Item dialog. From here, you can see that you can add a Global Application Class to your applications. This adds a `Global.asax` file. This file is used by the application to hold application-level events, objects, and variables — all of which are accessible application-wide. Active Server Pages developers had something similar with the `Global.asa` file.

Your ASP.NET applications can have only a single `Global.asax` file. This file supports a number of items. When it is created, you are given the following template:

```
<%@ Application Language="VB" %>

<script runat="server">

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
       ' Code that runs on application startup
    End Sub

    Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
       ' Code that runs on application shutdown
    End Sub

    Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
       ' Code that runs when an unhandled error occurs
    End Sub

    Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
       ' Code that runs when a new session is started
    End Sub

    Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
       ' Code that runs when a session ends.
       ' Note: The Session_End event is raised only when the sessionstate mode
       ' is set to InProc in the Web.config file. If session mode is
       ' set to StateServer
       ' or SQLServer, the event is not raised.
    End Sub

</script>
```

Just as you can work with page-level events in your `.aspx` pages, you can work with overall application events from the `Global.asax` file. In addition to the events listed in this code example, the following list details some of the events you can structure inside this file:

- `Application_Start`: Called when the application receives its very first request. It is an ideal spot in your application to assign any application-level variables or state that must be maintained across all users.
- `Session_Start`: Similar to the `Application_Start` event except that this event is fired when an individual user accesses the application for the first time. For instance, the `Application_Start` event fires once when the first request comes in, which gets the application going, but the `Session_Start` is invoked for each end user who requests something from the application for the first time.
- `Application_BeginRequest`: Although it is not listed in the preceding template provided by Visual Studio 2010, the `Application_BeginRequest` event is triggered before each and every request that comes its way. This means that when a request comes into the server, before this request is processed, the `Application_BeginRequest` is triggered and dealt with before any processing of the request occurs.
- `Application_AuthenticateRequest`: Triggered for each request and enables you to set up custom authentications for a request.
- `Application_Error`: Triggered when an error is thrown anywhere in the application by any user of the application. This is an ideal spot to provide application-wide error handling or an event recording the errors to the server's event logs.
- `Session_End`: When running in `InProc` mode, this event is triggered when an end user leaves the application.
- `Application_End`: Triggered when the application comes to an end. This is an event that most ASP.NET developers won't use that often because ASP.NET does such a good job of closing and cleaning up any objects that are left around.

In addition to the global application events that the `Global.asax` file provides access to, you can also use directives in this file as you can with other ASP.NET pages. The `Global.asax` file allows for the following directives:

- `@Application`
- `@Assembly`
- `@Import`

These directives perform in the same way when they are used with other ASP.NET page types.

An example of using the `Global.asax` file is shown in Listing 1-22. It demonstrates how to log when the ASP.NET application domain shuts down. When the ASP.NET application domain shuts down, the ASP.NET application abruptly comes to an end. Therefore, you should place any logging code in the `Application_End` method of the `Global.asax` file.

LISTING 1-22: Using the Application_End event in the Global.asax file

Available for download on Wrox.com

VB

```
<%@ Application Language="VB" %>
<%@ Import Namespace="System.Reflection" %>
<%@ Import Namespace="System.Diagnostics" %>

<script runat="server">

    Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
        Dim MyRuntime As HttpRuntime =
           GetType(System.Web.HttpRuntime).InvokeMember("_theRuntime",
           BindingFlags.NonPublic Or BindingFlags.Static Or _
           BindingFlags.GetField,
           Nothing, Nothing, Nothing)

        If (MyRuntime Is Nothing) Then
            Return
        End If

        Dim shutDownMessage As String =
           CType(MyRuntime.GetType().InvokeMember("_shutDownMessage",
           BindingFlags.NonPublic Or BindingFlags.Instance Or
           BindingFlags.GetField,
           Nothing, MyRuntime, Nothing), System.String)

        Dim shutDownStack As String =
           CType(MyRuntime.GetType().InvokeMember("_shutDownStack",
           BindingFlags.NonPublic Or BindingFlags.Instance Or
           BindingFlags.GetField,
           Nothing, MyRuntime, Nothing), System.String)

        If (Not EventLog.SourceExists(".NET Runtime")) Then
            EventLog.CreateEventSource(".NET Runtime", "Application")
        End If

        Dim logEntry As EventLog = New EventLog()
        logEntry.Source = ".NET Runtime"
        logEntry.WriteEntry(String.Format(
           "shutDownMessage={0}\r\n\r\n_shutDownStack={1}",
           shutDownMessage, shutDownStack), EventLogEntryType.Error)
    End Sub

</script>
```

C#

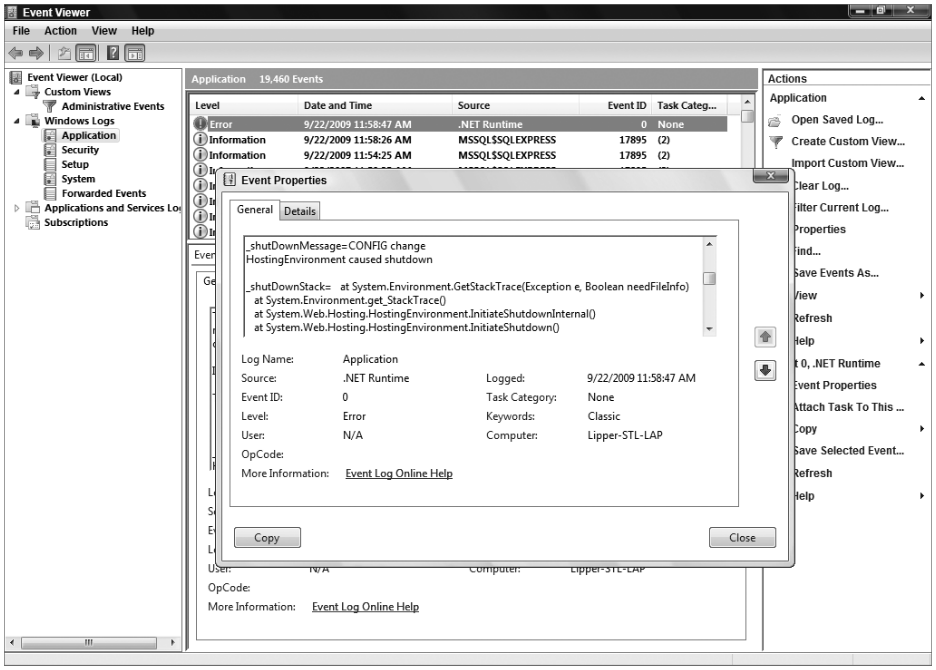
```
<%@ Application Language="C#" %>
<%@ Import Namespace="System.Reflection" %>
<%@ Import Namespace="System.Diagnostics" %>

<script runat="server">

    void Application_End(object sender, EventArgs e)
    {
        HttpRuntime runtime =

(HttpRuntime)typeof(System.Web.HttpRuntime).InvokeMember("_theRuntime",
           BindingFlags.NonPublic | BindingFlags.Static |
BindingFlags.GetField,
           null, null, null);

        if (runtime == null)
        {
            return;
        }

        string shutDownMessage =
           (string)runtime.GetType().InvokeMember("_shutDownMessage",
           BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.GetField,
           null, runtime, null);

        string shutDownStack =
           (string)runtime.GetType().InvokeMember("_shutDownStack",
           BindingFlags.NonPublic | BindingFlags.Instance | BindingFlags.GetField,
           null, runtime, null);

        if (!EventLog.SourceExists(".NET Runtime"))
        {
            EventLog.CreateEventSource(".NET Runtime", "Application");
        }
```

continues

LISTING 1-22 *(continued)*

```
            EventLog logEntry = new EventLog();
            logEntry.Source = ".NET Runtime";
            logEntry.WriteEntry(String.Format("\r\n\r\n_" +
               "shutDownMessage={0}\r\n\r\n_shutDownStack={1}",
               shutDownMessage, shutDownStack), EventLogEntryType.Error);
        }

    </script>
```

With this code in place in your `Global.asax` file, start your ASP.NET application. Next, do something to cause the application to restart. You could, for example, make a change to the `web.config` file while the application is running. This triggers the `Application_End` event, and you see the following addition (shown in Figure 1-17) to the event log.



```
_shutDownMessage=CONFIG change
HostingEnvironment caused shutdown

_shutDownStack=   at System.Environment.GetStackTrace(Exception e, Boolean needFileInfo)
   at System.Environment.get_StackTrace()
   at System.Web.Hosting.HostingEnvironment.InitiateShutdownInternal()
   at System.Web.Hosting.HostingEnvironment.InitiateShutdown()
```

Log Name:	Application		
Source:	.NET Runtime	Logged:	9/22/2009 11:58:47 AM
Event ID:	0	Task Category:	None
Level:	Error	Keywords:	Classic
User:	N/A	Computer:	Lipper-STL-LAP
OpCode:			
More Information:	Event Log Online Help		

FIGURE 1-17

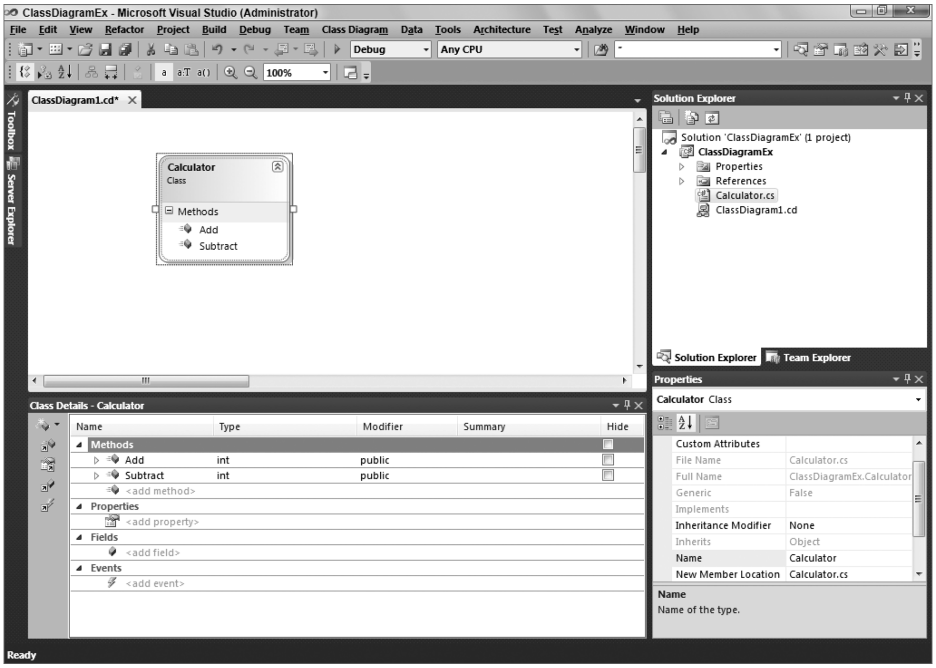
WORKING WITH CLASSES THROUGH VISUAL STUDIO 2010

So far, this chapter has shown you how to work with classes within your ASP.NET projects. In constructing and working with classes, you will find that Visual Studio 2010 is quite helpful. One particularly useful item is the class designer file. The class designer file has an extension of `.cd` and gives you a visual way to view your class, as well as all the available methods, properties, and other class items it contains.

To see this designer in action, create a new Class Library project in the language of your choice. This project has a single class file, `Class1.vb` or `.cs`. Delete this file and create a new class file called `Calculator.vb` or `.cs`, depending on the language you are using. From here, complete the class by creating a simple `Add()` and `Subtract()` method. Each of these methods takes in two parameters (of type `Integer`) and returns a single `Integer` with the appropriate calculation performed.

After you have the `Calculator` class in place, the easiest way to create your class designer file for this particular class is to right-click on the `Calculator.vb` file directly in the Solution Explorer and select View Class Diagram from the menu. This creates a `ClassDiagram1.cd` file in your solution.

Figure 1-18 presents the visual file, `ClassDiagram1.cd`.


ClassDiagramEx - Microsoft Visual Studio (Administrator)
File Edit View Refactor Project Build Debug Team Class Diagram Data Tools Architecture Test Analyze Window Help
Debug
Any CPU
100%
Toolbox
Server Explorer
ClassDiagram1.cd*
Calculator
Class
Methods
Add
Subtract
Solution Explorer
Solution 'ClassDiagramEx' (1 project)
ClassDiagramEx
Properties
References
Calculator.cs
ClassDiagram1.cd
Solution Explorer
Team Explorer
Class Details - Calculator
Properties
Ready

Name	Type	Modifier	Summary	Hide
Methods				
Add	int	public		
Subtract	int	public		
<add method>				
Properties				
<add property>				
Fields				
<add field>				
Events				
<add event>				

Calculator Class

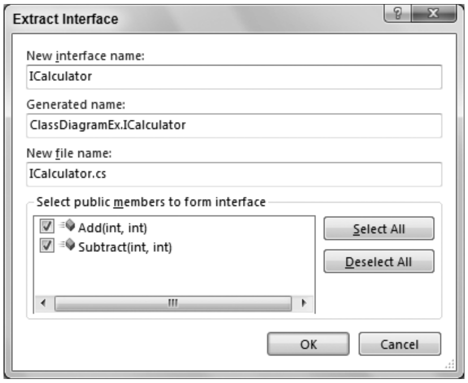
Custom Attributes	
File Name	Calculator.cs
Full Name	ClassDiagramEx.Calculator
Generic	False
Implements	
Inheritance Modifier	None
Inherits	Object
Name	Calculator
New Member Location	Calculator.cs

Name
Name of the type.

FIGURE 1-18

The new class designer file gives you a design view of your class. In the document window of Visual Studio, you see a visual representation of the `Calculator` class. The class is represented in a box and provides the name of the class, as well as two available methods that are exposed by the class. Because of the simplicity of this class, the details provided in the visual view are limited.

You can add additional classes to this diagram simply by dragging and dropping class files onto the design surface. You can then arrange the class files on the design surface as you want. A connection is in place for classes that are inherited from other class files or classes that derive from an interface or abstract class. In fact, you can extract an interface from the class you just created directly in the class designer by right-clicking on the Calculator class box and selecting Refactor ➪ Extract Interface from the provided menu (if you are working with C#). This launches the Extract Interface dialog (shown in Figure 1-19) that enables you to customize the interface creation.



New interface name: ICalculator
Generated name: ClassDiagramEx.ICalculator
New file name: ICalculator.cs
Select public members to form interface
[✓] Add(int, int)
[✓] Subtract(int, int)

FIGURE 1-19

After you click OK, the `ICalculator` interface is created and is then visually represented in the class diagram file, as illustrated in Figure 1-20.

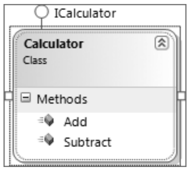
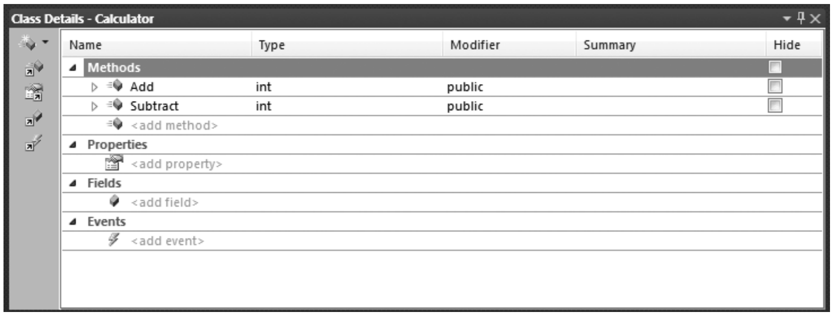


FIGURE 1-20

In addition to creating items such as interfaces on-the-fly, you can also modify your `Calculator` class by adding additional methods, properties, events, and more through the Class Details pane found in Visual Studio (see Figure 1-21).

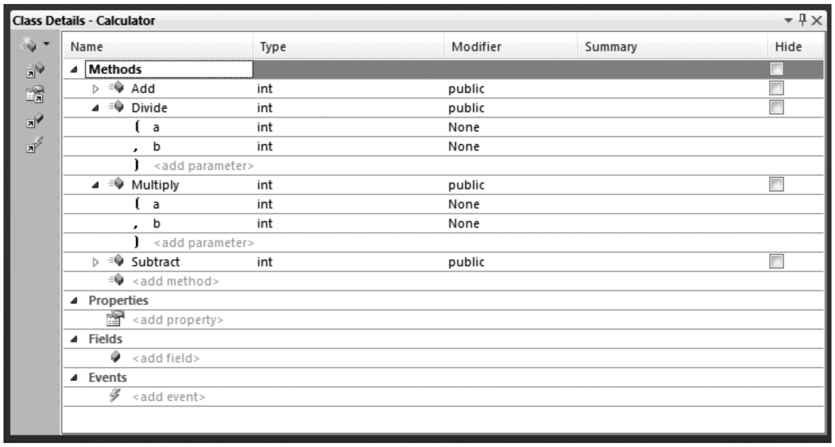


Class Details - Calculator

Name	Type	Modifier	Summary	Hide
Methods				☐
Add	int	public		☐
Subtract	int	public		☐
<add method>				
Properties				
<add property>				
Fields				
<add field>				
Events				
<add event>				

FIGURE 1-21

From this view of the class, you can directly add any additional methods, properties, fields, or events without directly typing code in your class file. When you enter these items in the Class Details view, Visual Studio generates the code for you on your behalf. For an example of this, add the additional `Multiply()` and `Divide()` methods that the `Calculator` class needs. Expanding the plus sign next to these methods shows the parameters needed in the signature. This is where you add the required `a` and `b` parameters. When you have finished, your Class Details screen should appear as shown in Figure 1-22.



Class Details - Calculator

Name	Type	Modifier	Summary	Hide
Methods				☐
Add	int	public		☐
Divide	int	public		☐
(a	int	None		
, b	int	None		
) <add parameter>				
Multiply	int	public		☐
(a	int	None		
, b	int	None		
) <add parameter>				
Subtract	int	public		☐
<add method>				
Properties				
<add property>				
Fields				
<add field>				
Events				
<add event>				

FIGURE 1-22

After you have added new `Multiply()` and `Divide()` methods and the required parameters, you see that the code in the `Calculator` class has changed to indicate these new methods are present. When the framework of the method is in place, you also see that the class has not been implemented in any fashion. The C# version of the `Multiply()` and `Divide()` methods created by Visual Studio is presented in Listing 1-23.

LISTING 1-23: The framework provided by Visual Studio's class designer

```
public int Multiply(int a, int b)
{
   throw new System.NotImplementedException();
}

public int Divide(int a, int b)
{
   throw new System.NotImplementedException();
}
```

The new class designer files give you a powerful way to view and understand your classes better — sometimes a picture really is worth a thousand words. One interesting last point on the `.cd` file is that Visual Studio is really doing all the work with this file. If you open the `ClassDesigner1.cd` file in Notepad, you see the results presented in Listing 1-24.

LISTING 1-24: The real ClassDesigner1.cd file as it appears in Notepad

```
<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
  <Class Name="ClassDiagramEx.Calculator">
    <Position X="1.25" Y="0.75" Width="1.5" />
    <TypeIdentifier>
      <HashCode>AAIAAAAAAQAAAAAAAAADAAAAAAAAAAAAAAAAAAAAAAA=</HashCode>
      <FileName>Calculator.cs</FileName>
    </TypeIdentifier>
    <Lollipop Position="0.2" />
  </Class>
  <Font Name="Segoe UI" Size="8.25" />
</ClassDiagram>
```

As you can see, it is a rather simple XML file that defines the locations of the class and the items connected to the class.

SUMMARY

This chapter covered a lot of ground. It discussed some of the issues concerning ASP.NET applications as a whole and the choices you have when building and deploying these new applications. With the help of Visual Studio 2010, you have options about which Web server to use when building your application and whether to work locally or remotely through the built-in FTP capabilities.

ASP.NET 4 and Visual Studio 2010 make it easy to build your pages using an inline coding model or to select a code-behind model that is simpler to use and easier to deploy than in the past. You also learned about the cross-posting capabilities and the fixed folders that ASP.NET 4 has incorporated to make your life easier. These folders make their resources available dynamically with no work on your part. You saw some of the outstanding compilation options that are at your disposal. Finally, you looked at ways in which Visual Studio 2010 makes it easy to work with the classes of your project.

As you worked through some of the examples, you may have been thinking, "WOW!" But wait . . . there's plenty more to come!

2

ASP.NET Server Controls and Client-Side Scripts

WHAT'S IN THIS CHAPTER?

- Building ASP.NET pages with server controls
- Working with HTML server controls
- Identifying server controls
- Modifying server controls with JavaScript

As discussed in the previous chapter, ASP.NET evolved from Microsoft's earlier Web technology called Active Server Pages (referred to as *ASP* then and *classic ASP* today). This model was completely different from today's ASP.NET. Classic ASP used interpreted languages to accomplish the construction of the final HTML document before it was sent to the browser. ASP.NET, on the other hand, uses true compiled languages to accomplish the same task. The idea of building Web pages based on objects in a compiled environment is one of the main focuses of this chapter.

This chapter looks at how to use a particular type of object in ASP.NET pages called a *server control* and how you can profit from using this control. We also introduce a particular type of server control — the HTML server control. The chapter also demonstrates how you can use JavaScript in ASP.NET pages to modify the behavior of server controls.

The rest of this chapter shows you how to use and manipulate server controls, both visually and programmatically, to help with the creation of your ASP.NET pages.

ASP.NET SERVER CONTROLS

In the past, one of the difficulties of working with classic ASP was that you were completely in charge of the entire HTML output from the browser by virtue of the server-side code you wrote. Although this might seem ideal, it created a problem because each browser interpreted the HTML given to it in a slightly different manner.

The two main browsers out there at the time were Microsoft's Internet Explorer and Netscape Navigator. This meant that not only did developers have to be cognizant of the browser type to which they were outputting HTML, but they also had to take into account which versions of those particular browsers might be making a request to their application. Some developers resolved the issue by creating two separate applications. When an end user made an initial request to the application, the code made a browser check to see what browser type was making the request. Then, the ASP page would redirect the request down one path for an IE user or down another path for a Netscape user.

Because requests came from so many different versions of the same browser, the developer often designed for the lowest possible version that might be used to visit the site. Essentially, everyone lost out by using the lowest common denominator as the target. This technique ensured that the page was rendered properly in most browsers making a request, but it also forced the developer to dumb-down his application. If applications were always built for the lowest common denominator, the developer could never take advantage of some of the more advanced features offered by newer browser versions.

ASP.NET server controls overcome these obstacles. When using the server controls provided by ASP.NET, you are not specifying the HTML to be output from your server-side code. Rather, you are specifying the functionality you want to see in the browser and letting ASP.NET decide on the output to be sent to the browser.

When a request comes in, ASP.NET examines the request to see which browser type is making the request, as well as the version of the browser, and then it produces HTML output specific to that browser. This process is accomplished by processing a User Agent header retrieved from the HTTP Request to *sniff* the browser. This means that you can now build for the best browsers out there without worrying about whether features will work in the browsers making requests to your applications. Because of the previously described capabilities, you will often hear these controls referred to as *smart controls*.

Types of Server Controls

ASP.NET provides two distinct types of server controls — HTML server controls and Web server controls. Each type of control is quite different and, as you work with ASP.NET, you will see that much of the focus is on the Web server controls. This does not mean that HTML server controls have no value. They do provide you with many capabilities — some that Web server controls do not give you.

You might be asking yourself which is the better control type to use. The answer is that it really depends on what you are trying to achieve. HTML server controls map to specific HTML elements. You can place an `HtmlTable` server control on your ASP.NET page that works dynamically with a `<table>` element. On the other hand, Web server controls map to specific functionality that you want on your ASP.NET pages. This means an `<asp:Panel>` control might use a `<table>` or another element altogether — it really depends on the capability of the browser making the request.

Table 2-1 provides a summary of information on when to use HTML server controls and when to use Web server controls.

TABLE 2-1

CONTROL TYPE	WHEN TO USE THIS CONTROL TYPE
HTML Server	When converting traditional ASP 3.0 Web pages to ASP.NET Web pages and speed of completion is a concern. It is a lot easier to change your HTML elements to HTML server controls than it is to change them to Web server controls.
	When you prefer a more HTML-type programming model.
	When you want to explicitly control the code that is generated for the browser. Though, simply using ASP.NET MVC for this (covered in Chapter 27) might be a better answer.
Web Server	When you require a richer set of functionality to perform complicated page requirements.
	When you are developing Web pages that will be viewed by a multitude of browser types and that require different code based upon these types.
	When you prefer a more Visual Basic–type programming model that is based on the use of controls and control properties.

Of course, some developers like to separate certain controls from the rest and place them in their own categories. For instance, you may see references to the following types of controls:

- **List controls:** These control types allow data to be bound to them for display purposes of some kind.
- **Rich controls:** Controls, such as the Calendar control, that display richer content and capabilities than other controls.

- **Validation controls:** Controls that interact with other form controls to validate the data that they contain.
- **User controls:** These are not really controls, but page templates that you can work with as you would a server control on your ASP.NET page.
- **Custom controls:** Controls that you build yourself and use in the same manner as the supplied ASP.NET server controls that come with the default install of ASP.NET 4.

When you are deciding between HTML server controls and Web server controls, remember that no hard and fast rules exist about which type to use. You might find yourself working with one control type more than another, but certain features are available in one control type that might not be available in the other. If you are trying to accomplish a specific task and you do not see a solution with the control type you are using, look at the other control type because it may very well hold the answer. Also, realize that you can mix and match these control types. Nothing says that you cannot use both HTML server controls and Web server controls on the same page or within the same application.

Building with Server Controls

You have a couple of ways to use server controls to construct your ASP.NET pages. You can actually use tools that are specifically designed to work with ASP.NET 4 that enable you to visually drag and drop controls onto a design surface and manipulate the behavior of the control. You can also work with server controls directly through code input.

Working with Server Controls on a Design Surface

Visual Studio 2010 enables you to visually create an ASP.NET page by dragging and dropping visual controls onto a design surface. You can get to this visual design option by clicking the Design tab at the bottom of the IDE when viewing your ASP.NET page. You can also show the Design view and the Source code view in the same document window. This is a feature available in Visual Studio 2008 and Visual Studio 2010. When the Design view is present, you can place the cursor on the page in the location where you want the control to appear and then double-click the control you want in the Toolbox window of Visual Studio. Unlike the 2002 and 2003 versions of Visual Studio, Visual Studio 2010 does a really good job of not touching your code when switching between the Design and Source tabs.

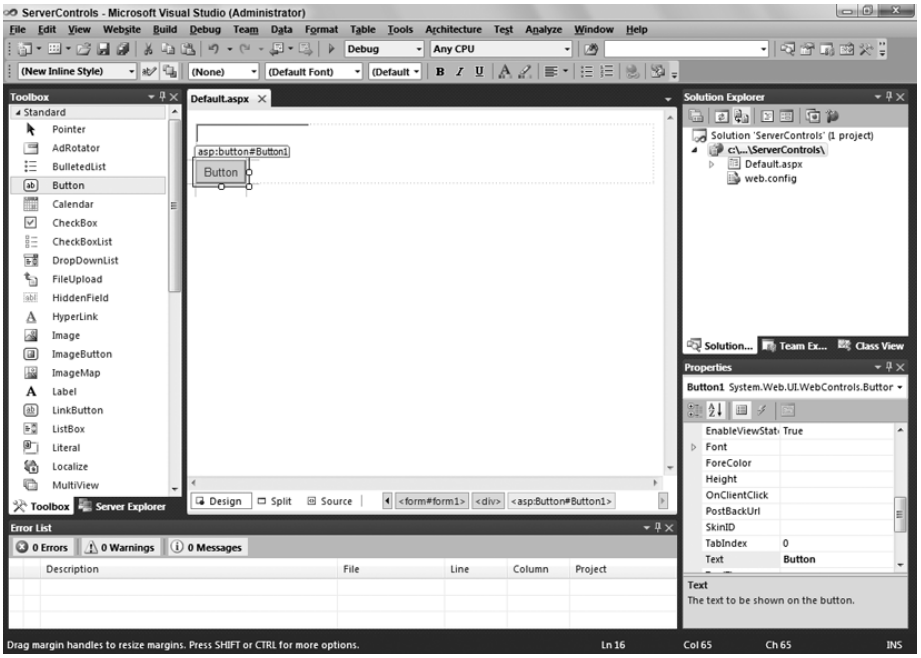
In the Design view of your page, you can highlight a control and the properties for the control appear in the Properties window. For example, Figure 2-1 shows a Button control selected in the design panel and its properties are displayed in the Properties window on the lower right.

Changing the properties in the window changes the appearance or behavior of the highlighted control. Because all controls inherit from a specific base class (`WebControl`), you can also highlight multiple controls at the same time and change the base properties of all the controls at once. You do this by holding down the Ctrl key as you make your control selections.

Coding Server Controls

You also can work from the code page directly. Because many developers prefer this, it is the default when you first create your ASP.NET page. Hand-coding your own ASP.NET pages may seem to be a slower approach than simply dragging and dropping controls onto a design surface, but it isn't as slow as you might think. You get plenty of assistance in coding your applications from Visual Studio 2010. As you start typing in Visual Studio, the IntelliSense features kick in and help you with code auto-completion. Figure 2-2, for example, shows an IntelliSense drop-down list of possible code completion statements that appeared as the code was typed.

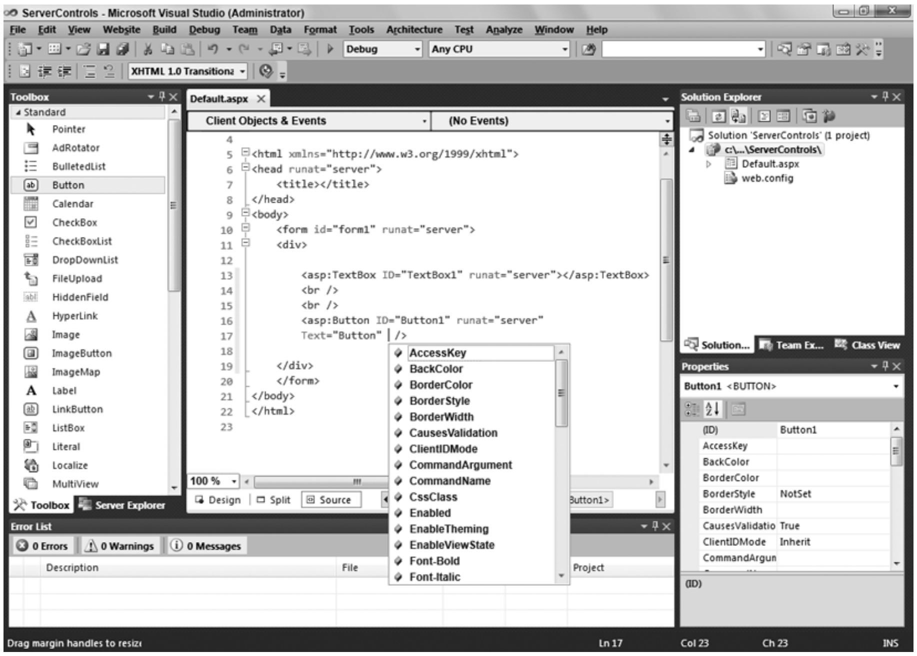
The IntelliSense focus is on the most commonly used attribute or statement for the control or piece of code that you are working with. Using IntelliSense effectively as you work is a great way to code with great speed.



EnableViewState	True
Font	
ForeColor	
Height	
OnClientClick	
PostBackUrl	
SkinID	
TabIndex	0
Text	Button

Text
The text to be shown on the button.

FIGURE 2-1



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

        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server"
        Text="Button" />

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

FIGURE 2-2

As with Design view, the Source view of your page lets you drag and drop controls from the Toolbox onto the code page itself. For example, dragging and dropping a TextBox control onto the code page produces the same results as dropping it on the design page:

```
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
```

You can also highlight a control in Source view or simply place your cursor in the code statement of the control, and the Properties window displays the properties of the control. Now, you can apply properties directly in the Properties window of Visual Studio, and these properties are dynamically added to the code of your control.

Working with Server Control Events

As discussed in Chapter 1, ASP.NET uses more of a traditional Visual Basic event model than classic ASP. Instead of working with interpreted code, you are actually coding an event-based structure for your pages. Classic ASP used an interpreted model — when the server processed the Web page, the code of the page was interpreted line-by-line in a linear fashion where the only "event" implied was the page loading. This meant that occurrences you wanted to be initiated early in the process were placed at the top of the page.

Today, ASP.NET uses an event-driven model. Items or coding tasks are initiated only when a particular event occurs. A common event in the ASP.NET programming model is `Page_Load`, which is illustrated in Listing 2-1.

LISTING 2-1: Working with specific page events

VB
```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   ' Code actions here
End Sub
```

C#
```
protected void Page_Load(object sender, EventArgs e)
{
   // Code actions here
}
```

Not only can you work with the overall page — as well as its properties and methods at particular moments in time through page events — but you can also work with the server controls contained on the page through particular control events. For example, one common event for a button on a form is `Button_Click`, which is illustrated in Listing 2-2.

LISTING 2-2: Working with a Button Click event

VB
```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   ' Code actions here
End Sub
```

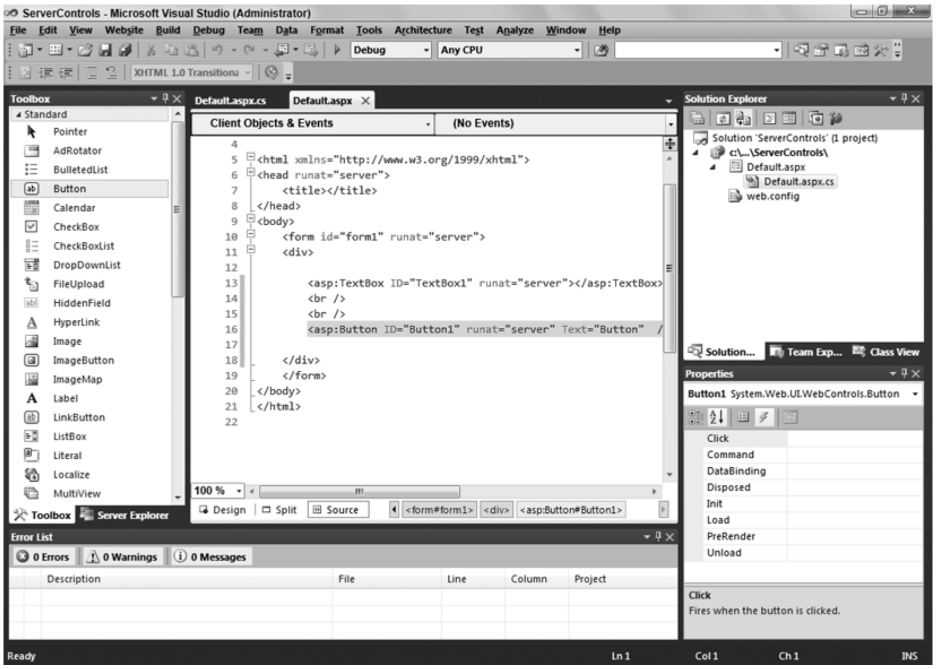
C#
```
protected void Button1_Click(object sender, EventArgs e)
{
   // Code actions here
}
```

The event shown in Listing 2-2 is fired only when the end user actually clicks the button on the form that has an `OnClick` attribute value of `Button1_Click`. Therefore, not only does the event handler exist in the server-side code of the ASP.NET page, but that handler is also hooked up using the `OnClick` property of the server control in the associated ASP.NET page markup, as illustrated in the following code:

```
<asp:Button ID="Button1" Runat="server" Text="Button" OnClick="Button1_Click" />
```

How do you fire these events for server controls? You have a couple of ways to go about it. The first way is to pull up your ASP.NET page in the Design view and double-click the control for which you want to create a server-side event. For instance, double-clicking a Button server control in Design view creates the structure of the `Button1_Click` event within your server-side code, whether the code is in a code-behind file or inline. This creates a stub handler for that server control's most popular event.

With that said, be aware that a considerable number of additional events are available to the Button control that you cannot get at by double-clicking the control. To access them, from any of the views within the IDE, choose the control from the Properties dialog. Then you find a lightning bolt icon that provides you a list of all the control's events. From here, you simply can double-click the event you are interested in, and Visual Studio creates the stub of the function you need. Figure 2-3 shows the event list displayed. You might, for example, want to work with the Button control's `PreRender` event rather than its `Click` event. The handler for the event you choose is placed in your server-side code.



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

        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" Text="Button"  /

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

Button1 System.Web.UI.WebControls.Button

Click
Command
DataBinding
Disposed
Init
Load
PreRender
Unload

Click
Fires when the button is clicked.

FIGURE 2-3

After you have an event structure in place, you can program specific actions that you want to occur when the event is fired.

APPLYING STYLES TO SERVER CONTROLS

More often than not, you want to change the default style (which is basically no style) to the server controls you implement in your applications. You most likely want to build your Web applications so that they reflect your own look-and-feel. One way to customize the appearance of the controls in your pages is to change the controls' properties.

As stated earlier in this chapter, to get at the properties of a particular control you simply highlight the control in the Design view of the page from Visual Studio. If you are working from the Source view, place the cursor in the code of the control. The properties presented in the Properties window allow you to control the appearance and behavior of the selected control.

Examining the Controls' Common Properties

Many of the default server controls that come with ASP.NET 4 are derived from the `WebControl` class and share similar properties that enable you to alter their appearance and behavior. Not all the derived controls use all the available properties (although many are implemented). Another important point is that not all server controls are implemented from the `WebControl` class. For instance, the Literal, PlaceHolder, Repeater, and XML server controls do not derive from the `WebControl` base class, but instead the `Control` class.

HTML server controls also do not derive from the `WebControl` base class because they are more focused on the set attributes of particular HTML elements. Table 2-2 lists the common properties the server controls share.

TABLE 2-2

PROPERTY	DESCRIPTION
`AccessKey`	Enables you to assign a character to be associated with the Alt key so that the end user can activate the control using quick-keys on the keyboard. For instance, you can assign a Button control an `AccessKey` property value of `K`. Now, instead of clicking the button on the ASP.NET page (using a pointer controlled by the mouse), the end user can simply press Alt + K.
`Attributes`	Enables you to define additional attributes for a Web server control that are not defined by a public property.
`BackColor`	Controls the color shown behind the control's layout on the ASP.NET page.
`BorderColor`	Assigns a color that is shown around the physical edge of the server control.
`BorderWidth`	Assigns a value to the width of the line that makes up the border of the control. Placing a number as the value assigns the number as a pixel-width of the border. The default border color is black if the `BorderColor` property is not used in conjunction with the `BorderWidth` property setting.
`BorderStyle`	Enables you to assign the design of the border that is placed around the server control. By default, the border is created as a straight line, but a number of different styles can be used for your borders. Other possible values for the `BorderStyle` property include `Dotted, Dashed, Solid, Double, Groove, Ridge, Inset,` and `Outset.`
`ClientIDMode`	Allows you to get or set the algorithm that is used to create the value of the `ClientID` property.
`CssClass`	Assigns a custom CSS (Cascading Style Sheet) class to the control.
`Enabled`	Enables you to turn off the functionality of the control by setting the value of this property to `False`. By default, the `Enabled` property is set to `True`.
`EnableTheming`	Enables you to turn on theming capabilities for the selected server control. The default value is `True`.
`EnableViewState`	Enables you to specify whether view state should be persisted for this control.
`Font`	Sets the font for all the text that appears anywhere in the control.
`ForeColor`	Sets the color of all the text that appears anywhere in the control.
`Height`	Sets the height of the control.
`SkinID`	Sets the skin to use when theming the control.
`Style`	Enables you to apply CSS styles to the control.
`TabIndex`	Sets the control's tab position in the ASP.NET page. This property works in conjunction with other controls on the page.
`ToolTip`	Assigns text that appears in a yellow box in the browser when a mouse pointer is held over the control for a short length of time. This can be used to add more instructions for the end user.
`Width`	Sets the width of the control.

You can see these common properties in many of the server controls you work with. Some of the properties of the `WebControl` class presented here work directly with the theming system built into ASP.NET such as the `EnableTheming` and `SkinID` properties. These properties are covered in more detail in Chapter 6. You also see additional properties that are specific to the control you are viewing. Learning about the properties from the preceding table enables you to quickly work with Web server controls and to modify them to your needs.

Next, look at some additional methods of customizing the look-and-feel of your server controls.

Changing Styles Using Cascading Style Sheets

One method of changing the look-and-feel of specific elements on your ASP.NET page is to apply a *style* to the element. The most rudimentary method of applying a defined look-and-feel to your page elements is to use various style-changing HTML elements such as `<font>`, `<b>`, and `<i>` directly.

All ASP.NET developers should have a good understanding of HTML. For more information on HTML, please read Wrox's Beginning Web Programming with HTML, XHTML, and CSS *(Wiley Publishing, Inc.; ISBN 978-0470-25931-3). You can also learn more about HTML and CSS design in ASP.NET by looking at Chapter 17 of this book.*

Using various HTML elements, you can change the appearance of many items contained on your pages. For instance, you can change a string's style as follows:

```
<font face="verdana">Pork chops and applesauce</font>
```

You can go through an entire application and change the style of page elements using any of the appropriate HTML elements. You will quickly find that this method works, but it is tough to maintain. To make any global style changes to your application, this method requires that you go through your application line-by-line to change each item individually. This can get cumbersome very fast!

Besides applying HTML elements to items to change their style, you can use another method known as *Cascading Style Sheets* (CSS). This alternative, but greatly preferred, styling technique allows you to assign formatting properties to HTML tags throughout your document in a couple of different ways. One way is to apply these styles directly to the HTML elements in your pages using *inline styles*. The other way involves placing these styles in an external stylesheet that either can be placed directly in an ASP.NET page or kept in a separate document that is simply referenced in the ASP.NET page. You explore these methods in the following sections.

Applying Styles Directly to HTML Elements

The first method of using CSS is to apply the styles directly to the tags contained in your ASP.NET pages. For instance, you apply a style to a string, as shown in Listing 2-3.

LISTING 2-3: Applying CSS styles directly to HTML elements

```
<p style="color:blue; font-weight:bold">
   Pork chops and applesauce
</p>
```

This text string is changed by the CSS included in the `<p>` element so that the string appears bold and blue. Using the style attribute of the `<p>` element, you can change everything that appears between the opening and closing `<p>` elements. When the page is generated, the first style change applied is to the text between the `<p>` elements. In this example, the text has changed to the color blue because of the `color: blue` declaration, and then the `font-weight:bold` declaration is applied. You can separate the styling declarations using semicolons, and you can apply as many styles as you want to your elements.

Applying CSS styles in this manner presents the same problem as simply applying various HTML style elements — this is a tough structure to maintain. If styles are scattered throughout your pages, making global style changes can be rather time consuming. Putting all the styles together in a stylesheet is the best approach. A couple of methods can be used to build your stylesheets.

Working with the Visual Studio Style Builder

Visual Studio 2010 includes Style Builder, a tool that makes the building of CSS styles fairly simple. It can be quite a time saver because so many possible CSS definitions are available to you. If you are new to CSS, this tool can make all the difference.

The Visual Studio Style Builder enables you to apply CSS styles to individual elements or to construct your own stylesheets. To access the New Style tool when applying a style to a single page element, highlight the page element and then while in the Design view of the IDE, select Format ➪ New Style from the VS 2010 menu. The Style Builder is shown in Figure 2-4.

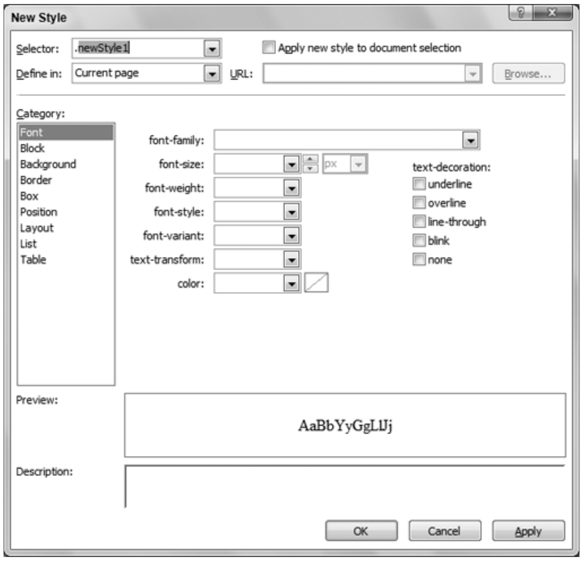
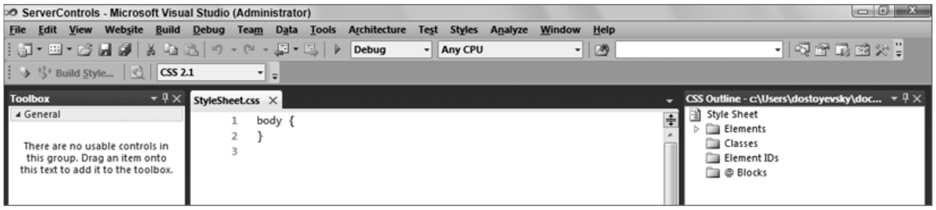


FIGURE 2-4

You can use the Visual Studio Style Builder to change quite a bit about your selected item. After making all the changes you want and clicking OK, you see the styles you chose applied to the selected element.

Creating External StyleSheets

You can use a couple of different methods to create stylesheets. The most common method is to create an *external* stylesheet — a separate stylesheet file that is referenced in the pages that employ the defined styles. To begin the creation of your external stylesheet, add a new item to your project. From the Add New Item dialog box, create a stylesheet called `StyleSheet.css`. Add the file to your project by pressing the Add button. Figure 2-5 shows the result.


ServerControls - Microsoft Visual Studio (Administrator)
File Edit View Website Build Debug Team Data Tools Architecture Test Styles Analyze Window Help
Debug
Any CPU
Build Style...
CSS 2.1
Toolbox
General
StyleSheet.css
CSS Outline - c:\Users\dostoyevsky\doc...
Style Sheet
Elements
Classes
Element IDs
@ Blocks

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
body {
}
```

FIGURE 2-5

Using an external stylesheet within your application enables you to make global changes to the look-and-feel of your application quickly. Simply making a change at this central point cascades the change as defined by the stylesheet to your entire application.

Creating Internal Stylesheets

The second method for applying a stylesheet to a particular ASP.NET page is to bring the defined stylesheet into the actual document by creating an *internal* stylesheet. Instead of making a reference to an external stylesheet file, you bring the style definitions into the document. Note, however, that it is considered best practice to use external, rather than internal, stylesheets.

Consider using an internal stylesheet only if you are applying certain styles to a small number of pages within your application. Listing 2-4 shows the use of an internal stylesheet.

LISTING 2-4: Using an internal stylesheet

Available for download on Wrox.com

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
```

continues

LISTING 2-4 *(continued)*

```
        <title>My ASP.NET Page</title>

        <style type="text/css">
           <!--
              body {
                 font-family: Verdana;
              }

              a:link {
                 text-decoration: none;
                 color: blue;
              }

              a:visited {
                 text-decoration: none;
                 color: blue;
              }

              a:hover {
                 text-decoration: underline;
                 color: red;
              }

           -->
        </style>

    </head>
    <body>
        <form id="form1" runat="server">
        <div>
            <a href="Default.aspx">Home</a>
        </div>
        </form>
    </body>
    </html>
```

Filename InternalStyleSheet.aspx

In this document, the internal stylesheet is set inside the opening and closing `<head>` elements. Although this is not a requirement, it is considered best practice. The stylesheet itself is placed between `<style>` tags with a type attribute defined as `text/css`.

HTML comment tags are included because not all browsers support internal stylesheets (it is generally the older browsers that do not accept them). Putting HTML comments around the style definitions hides these definitions from very old browsers. Except for the comment tags, the style definitions are handled in the same way they are done in an external stylesheet.

CSS Changes in ASP.NET 4

Prior to this release of ASP.NET, the rendered HTML from the ASP.NET server controls that you used weren't always compliant with the latest HTML standards that were out there.

An example of this is that when disabling a server control prior to ASP.NET 4, you would only need to set the `Enabled` property of the server control to `false`. This would render the control on the page but with a disabled attribute as illustrated here:

```
<span id="Label1" disabled="disabled">Hello there!</span>
```

The latest HTML standards doesn't allow for this construct. You are only allowed to use the `disabled` attribute on `<input>` elements. When working with ASP.NET 4, you will now have a property in the

`<pages>` element within the `web.config` file that will instruct ASP.NET what version style to use when rendering controls.

```
<pages controlRenderingCompatibilityVersion="4.0" />
```

If you have this set to `4.0`, as shown in the preceding code line, ASP.NET will now disable the control using CSS correctly as shown here:

```
<span id="Label1" class="aspNetDisabled">Hello there!</span>
```

As you can see, this time ASP.NET sets the `class` attribute rather than the `disabled` attribute. However, it is always possible to set the `controlRenderingCompatibilityVersion` value to `3.5` to revert to the old way of control rendering if you wish.

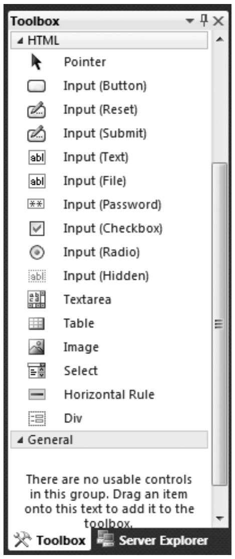


FIGURE 2-6

HTML SERVER CONTROLS

ASP.NET enables you to take HTML elements and, with relatively little work on your part, turn them into server-side controls. Afterward, you can use them to control the behavior and actions of elements implemented in your ASP.NET pages.

Of course, you can place any HTML you want in your pages. You have the option of using the HTML placed in the page as a server-side control. You can also find a list of HTML elements contained in the Toolbox of Visual Studio (shown in Figure 2-6).

Dragging and dropping any of these HTML elements from the Toolbox to the Design or Source view of your ASP.NET page in the Document window simply produces the appropriate HTML element. For instance, placing an HTML Button control on your page produces the following results in your code:

```
<input id="Button1" type="button" value="button" />
```

In this state, the Button control is not a server-side control. It is simply an HTML element and nothing more. You can turn this into an HTML server control very easily. In Source view, you simply change the HTML element by adding a `runat="server"` to the control:

```
<input id="Button1" type="button" value="button" runat="server" />
```

After the element is converted to a server control (through the addition of the `runat="server"` attribute and value), you can work with the selected element on the server side as you would work with any of the Web server controls. Listing 2-5 shows an example of some HTML server controls.

LISTING 2-5: Working with HTML server controls

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_ServerClick(ByVal sender As Object, _
       ByVal e As System.EventArgs)
        Response.Write("Hello " & Text1.Value)
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using HTML Server Controls</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        What is your name?<br />
        <input id="Text1" type="text" runat="server" />
        <input id="Button1" type="button" value="Submit" runat="server"
```

continues

LISTING 2-5 *(continued)*

```
          onserverclick="Button1_ServerClick" />
     </div>
     </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_ServerClick(object sender, EventArgs e)
    {
        Response.Write("Hello " + Text1.Value);
    }
</script>
```

Filename HTMLServerControls.aspx

In this example, you can see two HTML server controls on the page. Both are simply typical HTML elements with the additional `runat="server"` attribute added. If you are working with HTML elements as server controls, you must include an `id` attribute so that the server control can be identified in the server-side code.

The Button control includes a reference to a server-side event using the `OnServerClick` attribute. This attribute points to the server-side event that is triggered when an end user clicks the button — in this case, `Button1_ServerClick`. Within the `Button1_ServerClick` event, the value placed in the text box is output by using the `Value` property.

Looking at the HtmlControl Base Class

All the HTML server controls use a class that is derived from the `HtmlControl` base class (fully qualified as `System.Web.UI.HtmlControls.HtmlControl`). These classes expose many properties from the control's derived class. Table 2-3 details some of the properties available from this base class. Some of these items are themselves derived from the base `Control` class.

TABLE 2-3

METHOD OR PROPERTY	DESCRIPTION
`Attributes`	Provides a collection of name/value of all the available attributes specified in the control, including custom attributes.
`Disabled`	Allows you to get or set whether the control is disabled using a Boolean value.
`EnableTheming`	Enables you, using a Boolean value, to get or set whether the control takes part in the page theming capabilities.
`EnableViewState`	Allows you to get or set a Boolean value that indicates whether the control participates in the page's view state capabilities.
`ID`	Allows you to get or set the unique identifier for the control.
`Page`	Allows you to get a reference to the `Page` object that contains the specified server control.
`Parent`	Gets a reference to the parent control in the page control hierarchy.
`Site`	Provides information about the container for which the server control belongs.
`SkinID`	When the `EnableTheming` property is set to `True`, the `SkinID` property specifies the named skin that should be used in setting a theme.
`Style`	Makes references to the CSS style collection that applies to the specified control.
`TagName`	Provides the name of the element that is generated from the specified control.
`Visible`	Specifies whether the control is visible (rendered) on the generated page.

You can find a more comprehensive list in the SDK.

Looking at the HtmlContainerControl Class

The `HtmlControl` base class is used for those HTML classes that are focused on HTML elements that can be contained within a single node. For instance, the `<img>`, `<input>`, and `<link>` elements work from classes derived from the `HtmlControl` class.

Other HTML elements such as `<a>`, `<form>`, and `<select>`, require an opening and closing set of tags. These elements use classes that are derived from the `HtmlContainerControl` class — a class specifically designed to work with HTML elements that require a closing tag.

Because the `HtmlContainerControl` class is derived from the `HtmlControl` class, you have all the `HtmlControl` class's properties and methods available to you as well as some new items that have been declared in the `HtmlContainerControl` class itself. The most important of these are the `InnerText` and `InnerHtml` properties:

- `InnerHtml`: Enables you to specify content that can include HTML elements to be placed between the opening and closing tags of the specified control.
- `InnerText`: Enables you to specify raw text to be placed between the opening and closing tags of the specified control.

Looking at All the HTML Classes

It is quite possible to work with every HTML element because a corresponding class is available for each one of them. The .NET Framework documentation shows the following classes for working with your HTML server controls:

- `HtmlAnchor` controls the `<a>` element.
- `HtmlButton` controls the `<button>` element.
- `HtmlForm` controls the `<form>` element.
- `HtmlHead` controls the `<head>` element.
- `HtmlImage` controls the `<img>` element.
- `HtmlInputButton` controls the `<input type="button">` element.
- `HtmlInputCheckBox` controls the `<input type="checkbox">` element.
- `HtmlInputFile` controls the `<input type="file">` element.
- `HtmlInputHidden` controls the `<input type="hidden">` element.
- `HtmlInputImage` controls the `<input type="image">` element.
- `HtmlInputPassword` controls the `<input type="password">` element.
- `HtmlInputRadioButton` controls the `<input type="radio">` element.
- `HtmlInputReset` controls the `<input type="reset">` element.
- `HtmlInputSubmit` controls the `<input type="submit">` element.
- `HtmlInputText` controls the `<input type="text">` element.
- `HtmlLink` controls the `<link>`element.
- `HtmlMeta` controls the `<meta>` element.
- `HtmlSelect` controls the `<select>` element.
- `HtmlTable` controls the `<table>` element.
- `HtmlTableCell` controls the `<td>` element.
- `HtmlTableRow` controls the `<tr>` element.
- `HtmlTextArea` controls the `<textarea>` element.
- `HtmlTitle` controls the `<title>` element.

You gain access to one of these classes when you convert an HTML element to an HTML server control. For example, convert the `<title>` element to a server control this way:

```
<title id="Title1" runat="Server"/>
```

That gives you access to the `HtmlTitle` class for this particular HTML element. Using this class instance, you can perform a number of tasks including providing a text value for the page title dynamically:

VB

```
Title1.Text = DateTime.Now.ToString()
```

C#

```
Title1.Text = DateTime.Now.ToString();
```

You can get most of the HTML elements you need by using these classes, but a considerable number of other HTML elements are at your disposal that are not explicitly covered by one of these HTML classes. For example, the `HtmlGenericControl` class provides server-side access to any HTML element you want.

Using the HtmlGenericControl Class

You should be aware of the importance of the `HtmlGenericControl` class; it gives you some capabilities that you do not get from any other server control offered by ASP.NET. For instance, using the `HtmlGenericControl` class, you can get server-side access to the `<meta>`, `<p>`, `<span>`, or other elements that would otherwise be unreachable.

Listing 2-6 shows you how to change the `<meta>` element in your page using the `HtmlGenericControl` class.

LISTING 2-6: Changing the <meta> element using the HtmlGenericControl class

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Meta1.Attributes("Name") = "description"
        Meta1.Attributes("CONTENT") = "Generated on: " & DateTime.Now.ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Using the HtmlGenericControl class</title>
    <meta id="Meta1" runat="server" />
</head>
<body>
    <form id="form1" runat="server">
    <div>
        The rain in Spain stays mainly in the plains.
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Meta1.Attributes["Name"] = "description";
        Meta1.Attributes["CONTENT"] = "Generated on: " + DateTime.Now.ToString();
    }
</script>
```

Filename HTMLGenericControl.aspx

In this example, the page's `<meta>` element is turned into an HTML server control with the addition of the `id` and `runat` attributes. Because the `HtmlGenericControl` class (which inherits from `HtmlControl`) can work with a wide range of HTML elements, you cannot assign values to HTML attributes in the same manner as you do when working with the other HTML classes (such as `HtmlInputButton`). You assign values to the attributes of an HTML element using the `HtmlGenericControl` class's `Attributes` property, specifying the attribute you are working with as a string value.

The following is a partial result of running the example page:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta id="Meta1" Name="description"
    CONTENT="Generated on: 2/5/2010 2:42:52 PM"></meta>
   <title>Using the HtmlGenericControl class</title>
</head>
```

By using the `HtmlGenericControl` class, along with the other HTML classes, you can manipulate every element of your ASP.NET pages from your server-side code.

IDENTIFYING ASP.NET SERVER CONTROLS

When you create your ASP.NET pages with a series of controls, many of the controls are nested and many are even dynamically laid out by ASP.NET itself. For instance, when you are working with user controls, the GridView, ListView, Repeater, and more, ASP.NET is constructing a complicated control tree that is rendered to the page.

What happens when this occurs is that ASP.NET needs to provide these dynamic controls with IDs. When it does this, you end up with IDs such as `GridView1$ctl02$ctl00`. These sorts of control IDs are not a good thing because they are unpredictable and make it difficult to work with the control from client-side code.

To help this situation, ASP.NET 4 is the first release that includes the ability to control the IDs that are used for your controls. To demonstrate the issue, Listing 2-7 shows some code that results in some unpredictable client IDs for the controls. To start, first create a user control.

LISTING 2-7: A user control with some simple controls

Available for download on Wrox.com

```
<%@ Control Language="C#" AutoEventWireup="true"
    CodeFile="WebUserControl.ascx.cs" Inherits="WebUserControl" %>

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<br />
<asp:Button ID="Button1" runat="server" Text="Button" />
```

Then the next step is to use this user control within one of your ASP.NET pages. This is illustrated here in Listing 2-8.

LISTING 2-8: Making use of the user control within a simple ASP.NET page

Available for download on Wrox.com

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

<%@ Register src="WebUserControl.ascx" tagname="WebUserControl" tagprefix="uc1" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Working with Control IDs</title>
</head>
<body>
    <form id="form1" runat="server">
```

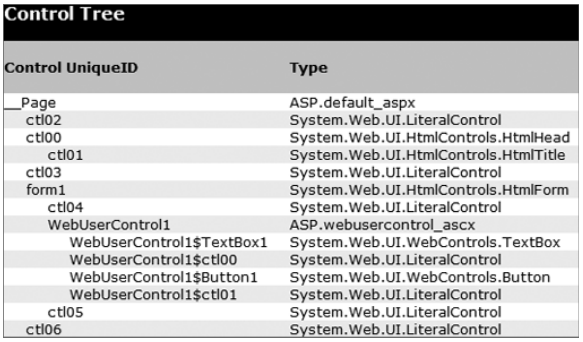
continues

LISTING 2-8 *(continued)*

```
    <div>
        <uc1:WebUserControl ID="WebUserControl1" runat="server" />

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

So this user control on the page contains only two simple server controls that are then rendered onto the page. If you look back at Listing 2-7, you can see that they have pretty simple control IDs assigned to them. There is a TextBox server control with the `ID` value of `TextBox1` and a Button server control with the `ID` value of `Button1`.

Looking at the page code from Listing 2-8, you can see in the `@Page` directive that the `Trace` attribute is set to `true`. This gives you the ability to see the `ClientID` that is produced in the control tree of the page. Running this page, you see the following results in the page, as shown in Figure 2-7.



Control Tree

Control UniqueID	Type
__Page	ASP.default_aspx
ctl02	System.Web.UI.LiteralControl
ctl00	System.Web.UI.HtmlControls.HtmlHead
ctl01	System.Web.UI.HtmlControls.HtmlTitle
ctl03	System.Web.UI.LiteralControl
form1	System.Web.UI.HtmlControls.HtmlForm
ctl04	System.Web.UI.LiteralControl
WebUserControl1	ASP.webusercontrol_ascx
WebUserControl1$TextBox1	System.Web.UI.WebControls.TextBox
WebUserControl1$ctl00	System.Web.UI.LiteralControl
WebUserControl1$Button1	System.Web.UI.WebControls.Button
WebUserControl1$ctl01	System.Web.UI.LiteralControl
ctl05	System.Web.UI.LiteralControl
ctl06	System.Web.UI.LiteralControl

FIGURE 2-7

If you look at the source code for the page, you see the following snippet of code:

```
<div>
   <input name="WebUserControl1$TextBox1" type="text" id="WebUserControl1_TextBox1" />
   <br />
   <input type="submit" name="WebUserControl1$Button1" value="Button"
    id="WebUserControl1_Button1" />
</div>
```

From this, you can see the ASP.NET assigned control IDs are lengthy and something you probably wouldn't choose yourself. The TextBox server control was output with a `name` value of `WebUserControl1$TextBox1` and an `id` value of `WebUserControl1_TextBox1`. A lot of this is done to make sure that the controls end up with a unique ID.

ASP.NET 4 includes the ability to control these assignments through the use of the `ClientIDMode` attribute. The possible values of this attribute include `AutoID`, `Inherit`, `Predictable`, and `Static`. An example of setting this value is provided here:

```
<uc1:WebUserControl ID="WebUserControl1" runat="server" ClientIDMode="AutoID" />
```

This example uses `AutoID`, forcing the naming to abide by how it was done in the .NET Framework 3.5 and earlier. Using this gives you the following results:

- **name:** `WebUserControl1$TextBox1`
  `WebUserControl1$Button1`
- **id:** `WebUserControl1_TextBox1`
  `WebUserControl1_Button1`

If you use `Inherit`, it simply copies how it is done by the containing control, the page, or the application. Therefore, for this example, you would end up with the same values as if you used `AutoID`. The `Inherit` value is the default value for all controls.

`Predictable` is generally used for databound controls that have a nesting of other controls (for example, the Repeater control). When used with a `ClientIDRowSuffix` property value, it appends this value rather than increments with a number (for example, `ctrl1`, `ctrl2`).

A value of `Static` gives you the name of the control you have assigned. It is up to you to ensure the uniqueness of the identifiers. Setting the `ClientIDMode` to `Static` for the user control in our example gives you the following values:

- **name:** `WebUserControl1$TextBox1`
  `WebUserControl1$Button1`
- **id:** `TextBox1`
  `Button1`

You can set the `ClientID` property at the control, container control, user control, page, or even application level via the `<pages>` element in the `machine.config` or `web.config` file.

Now with this new capability, you will find that working with your server controls using technologies like JavaScript on the client is far easier than before. The next section takes a look at using JavaScript within your ASP.NET pages.

MANIPULATING PAGES AND SERVER CONTROLS WITH JAVASCRIPT

Developers generally like to include some of their own custom JavaScript functions in their ASP.NET pages. You have a couple of ways to do this. The first is to apply JavaScript directly to the controls on your ASP. NET pages. For example, look at a simple TextBox server control, shown in Listing 2-9, which displays the current date and time.

LISTING 2-9: Showing the current date and time

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   TextBox1.Text = DateTime.Now.ToString()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e) {
   TextBox1.Text = DateTime.Now.ToString();
}
```

This little bit of code displays the current date and time on the page of the end user. The problem is that the date and time displayed are correct for the Web server that generated the page. If someone sits in the Pacific time zone (PST), and the Web server is in the Eastern time zone (EST), the page won't be correct for that viewer. If you want the time to be correct for anyone visiting the site, regardless of where they reside in the world, you can employ JavaScript to work with the TextBox control, as illustrated in Listing 2-10.

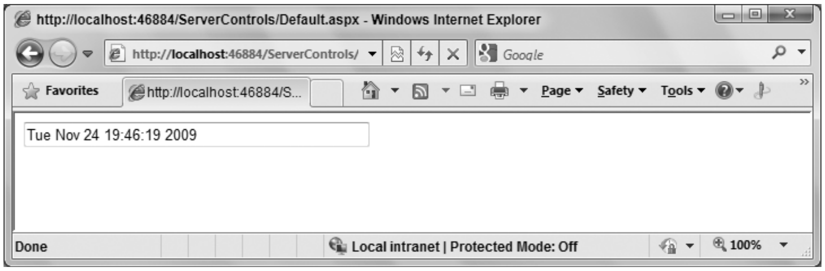
LISTING 2-10: Using JavaScript to show the current time for the end user

Available for download on Wrox.com

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Using JavaScript</title>
</head>
<body onload="javascript:document.forms[0]['TextBox1'].value=Date();">
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" Runat="server" Width="300"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

Filename CurrentTimeJS.aspx

In this example, even though you are using a standard TextBox server control from the Web server control family, you can get at this control using JavaScript that is planted in the `onload` attribute of the `<body>` element. The value of the `onload` attribute actually points to the specific server control via an anonymous function by using the value of the `ID` attribute from the server control: `TextBox1`. You can get at other server controls on your page by employing the same methods. This bit of code produces the result illustrated in Figure 2-8.


http://localhost:46884/ServerControls/Default.aspx - Windows Internet Explorer
http://localhost:46884/ServerControls/
Google
Favorites
http://localhost:46884/S...
Page
Safety
Tools
Done
Local intranet | Protected Mode: Off
100%

Tue Nov 24 19:46:19 2009

FIGURE 2-8

ASP.NET uses the `Page.ClientScript` property to register and place JavaScript functions on your ASP.NET pages. Three of these methods are reviewed here. More methods and properties than just these three are available through the `ClientScript` object (which references an instance of `System.Web.UI.ClientScriptManager`), but these are the more useful ones. You can find the rest in the SDK documentation.

The `Page.RegisterStartupScript` *and the* `Page.RegisterClientScriptBlock` *methods from the .NET Framework 1.0/1.1 are now considered obsolete. Both of these possibilities for registering scripts required a key/script set of parameters. Because two separate methods were involved, there was an extreme possibility that some key name collisions would occur. The* `Page.ClientScript` *property is meant to bring all the script registrations under one umbrella, making your code less error prone.*

Using Page.ClientScript.RegisterClientScriptBlock

The `RegisterClientScriptBlock` method allows you to place a JavaScript function at the top of the page. This means that the script is in place for the startup of the page in the browser. Its use is illustrated in Listing 2-11.

LISTING 2-11: Using the RegisterClientScriptBlock method

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
      Dim myScript As String = "function AlertHello() { alert('Hello ASP.NET'); }"
      Page.ClientScript.RegisterClientScriptBlock(Me.GetType(), "MyScript",
         myScript, True)
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Adding JavaScript</title>
</head>
```

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" Runat="server" Text="Button"
         OnClientClick="AlertHello()" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
      string myScript = @"function AlertHello() { alert('Hello ASP.NET'); }";
      Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
         "MyScript", myScript, true);
    }
</script>
```

Filename RegisterClientScriptBlock.aspx

From this example, you can see that you create the JavaScript function `AlertHello()` as a string called `myScript`. Then using the `Page.ClientScript.RegisterClientScriptBlock` method, you program the script to be placed on the page. The two possible constructions of the `RegisterClientScriptBlock` method are the following:

- `RegisterClientScriptBlock` (*type*, *key*, *script*)
- `RegisterClientScriptBlock` (*type*, *key*, *script*, *script tag specification*)

In the example from Listing 2-11, you are specifying the type as `Me.GetType()`, the key, the script to include, and then a `Boolean` value setting of `True` so that .NET places the script on the ASP.NET page with `<script>` tags automatically. When running the page, you can view the source code for the page to see the results:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
    Adding JavaScript
</title></head>
<body>
    <form method="post" action="JavaScriptPage.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE"
 value="/wEPDwUKMTY3NzE5MjIyMGRkiyYSRMg+bcXi9DiawYlbxndiTDo=" />
</div>


<script type="text/javascript">
<!--
function AlertHello() { alert('Hello ASP.NET'); }// -->
</script>

    <div>
        <input type="submit" name="Button1" value="Button" onclick="AlertHello();"
         id="Button1" />
    </div>
    </form>
</body>
</html>
```

From this, you can see that the script specified was indeed included on the ASP.NET page before the page code. Not only were the `<script>` tags included, but the proper comment tags were added around the script (so older browsers will not break).

Using Page.ClientScript.RegisterStartupScript

The `RegisterStartupScript` method is not too much different from the `RegisterClientScriptBlock` method. The big difference is that the `RegisterStartupScript` places the script at the bottom of the ASP.NET page instead of at the top. In fact, the `RegisterStartupScript` method even takes the same constructors as the `RegisterClientScriptBlock` method:

- ➤ `RegisterStartupScript` (*type*, *key*, *script*)
- ➤ `RegisterStartupScript` (type, key, script, script tag specification)

So what difference does it make where the script is registered on the page? A lot, actually!

If you have a bit of JavaScript that is working with one of the controls on your page, in most cases you want to use the `RegisterStartupScript` method instead of `RegisterClientScriptBlock`. For example, you'd use the following code to create a page that includes a simple `<asp:TextBox>` control that contains a default value of `Hello ASP.NET`.

```
<asp:TextBox ID="TextBox1" Runat="server">Hello ASP.NET</asp:TextBox>
```

Then use the `RegisterClientScriptBlock` method to place a script on the page that utilizes the value in the `TextBox1` control, as illustrated in Listing 2-12.

LISTING 2-12: Improperly using the RegisterClientScriptBlock method

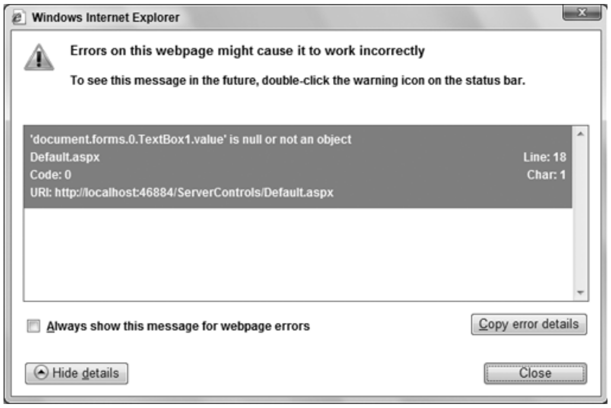
VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim myScript As String = "alert(document.forms[0]['TextBox1'].value);"
   Page.ClientScript.RegisterClientScriptBlock(Me.GetType(), "myKey", myScript,
      True)
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   string myScript = @"alert(document.forms[0]['TextBox1'].value);";
   Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
      "MyScript", myScript, true);
}
```

Running this page (depending on the version of IE you are using) gives you a JavaScript error, as shown in Figure 2-9.


Windows Internet Explorer

Errors on this webpage might cause it to work incorrectly

To see this message in the future, double-click the warning icon on the status bar.

```
'document.forms.0.TextBox1.value' is null or not an object
Default.aspx                                         Line: 18
Code: 0                                              Char: 1
URI: http://localhost:46884/ServerControls/Default.aspx
```

Always show this message for webpage errors
Copy error details
Hide details
Close

FIGURE 2-9

The reason for the error is that the JavaScript function fired before the text box was even placed on the screen. Therefore, the JavaScript function did not find `TextBox1`, and that caused an error to be thrown by the page. Now try the `RegisterStartupScript` method shown in Listing 2-13.

Available for download on Wrox.com

LISTING 2-13: Using the RegisterStartupScript method

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim myScript As String = "alert(document.forms[0]['TextBox1'].value);"
   Page.ClientScript.RegisterStartupScript(Me.GetType(), "myKey", myScript,
      True)
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   string myScript = @"alert(document.forms[0]['TextBox1'].value);";
   Page.ClientScript.RegisterStartupScript(this.GetType(),
      "MyScript", myScript, true);
}
```

Filename RegisterStartupScript.aspx

This approach puts the JavaScript function at the bottom of the ASP.NET page, so when the JavaScript actually starts, it finds the `TextBox1` element and works as planned. The result is shown in Figure 2-10.

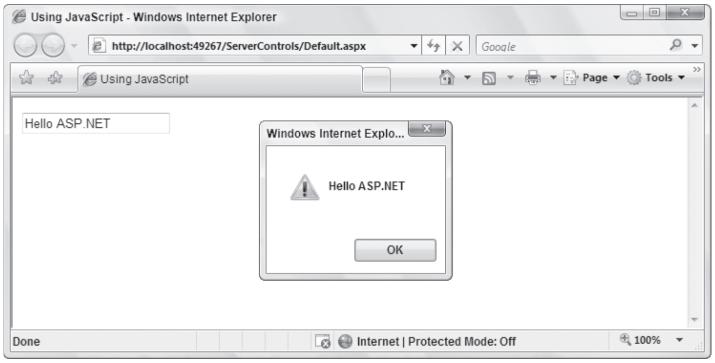

Using JavaScript - Windows Internet Explorer
http://localhost:49267/ServerControls/Default.aspx
Google
Using JavaScript
Page
Tools
Hello ASP.NET
Windows Internet Explo...
Hello ASP.NET
OK
Done
Internet | Protected Mode: Off
100%

FIGURE 2-10

Using Page.ClientScript.RegisterClientScriptInclude

The final method is `RegisterClientScriptInclude`. Many developers place their JavaScript inside a `.js` file, which is considered a best practice because it makes it very easy to make global JavaScript changes to the application. You can register the script files on your ASP.NET pages using the `RegisterClientScriptInclude` method illustrated in Listing 2-14.

LISTING 2-14: Using the RegisterClientScriptInclude method

VB

```
Dim myScript As String = "myJavaScriptCode.js"
Page.ClientScript.RegisterClientScriptInclude("myKey", myScript)
```

C#

```
string myScript = "myJavaScriptCode.js";
Page.ClientScript.RegisterClientScriptInclude("myKey", myScript);
```

This creates the following construction on the ASP.NET page:

```
<script src="myJavaScriptCode.js" type="text/javascript"></script>
```

CLIENT-SIDE CALLBACK

ASP.NET 4 includes a client callback feature that enables you to retrieve page values and populate them to an already-generated page without regenerating the page. This was introduced with ASP.NET 2.0. This capability makes it possible to change values on a page without going through the entire postback cycle; that means you can update your pages without completely redrawing the page. End users will not see the page flicker and reposition, and the pages will have a flow more like the flow of a thick-client application.

To work with the callback capability, you have to know a little about working with JavaScript. This book does not attempt to teach you JavaScript. If you need to get up to speed on this rather large topic, check out Wrox's *Beginning JavaScript, Fourth Edition*, by Paul Wilton and Jeremy McPeak (Wiley Publishing, Inc., ISBN: 978-0-470-52593-7).

You can also accomplish client callbacks in a different manner using ASP.NET AJAX. You will find more information on this in Chapters 18 and 19.

Comparing a Typical Postback to a Callback

Before you jump into some examples of the callback feature, first look at a comparison to the current postback feature of a typical ASP.NET page.

When a page event is triggered on an ASP.NET page that is working with a typical postback scenario, a lot is going on. The diagram in Figure 2-11 illustrates the process.

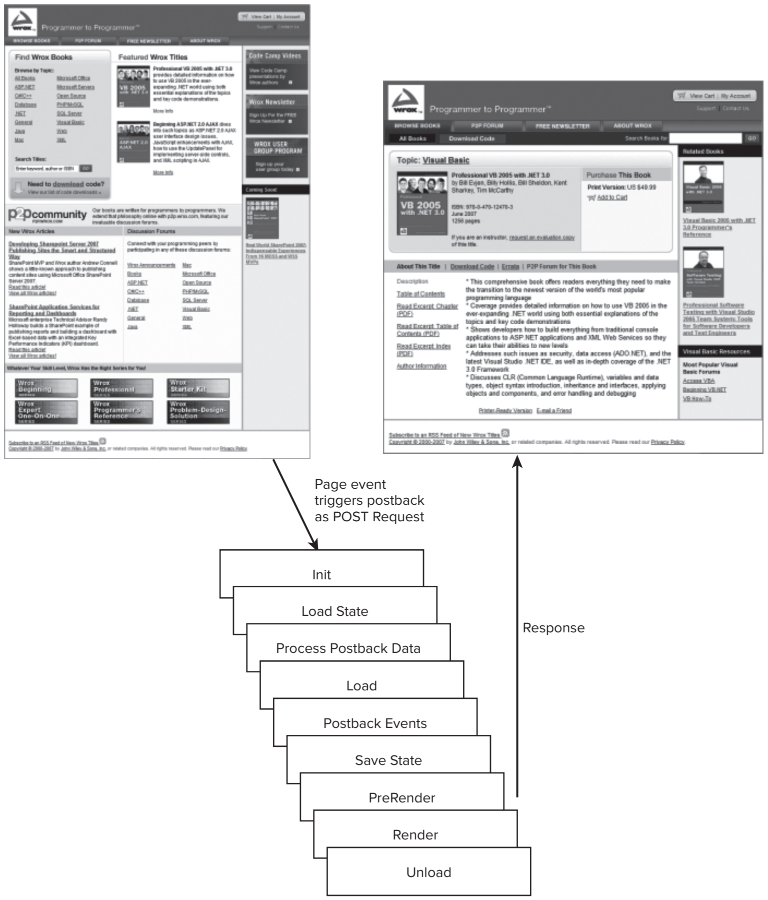


FIGURE 2-11

In a normal postback situation, an event of some kind triggers an HTTP Post request to be sent to the Web server. An example of such an event might be the end user clicking a button on the form. This sends the HTTP Post request to the Web server, which then processes the request with the `IPostbackEventHandler` and runs the request through a series of page events. These events include loading the state (as found in the view state of the page), processing data, processing postback events, and finally rendering the page to be interpreted by the consuming browser once again. The process completely reloads the page in the browser, which is what causes the flicker and the realignment to the top of the page.

On the other hand, you have the alternative of using the callback capabilities, as shown in the diagram in Figure 2-12.

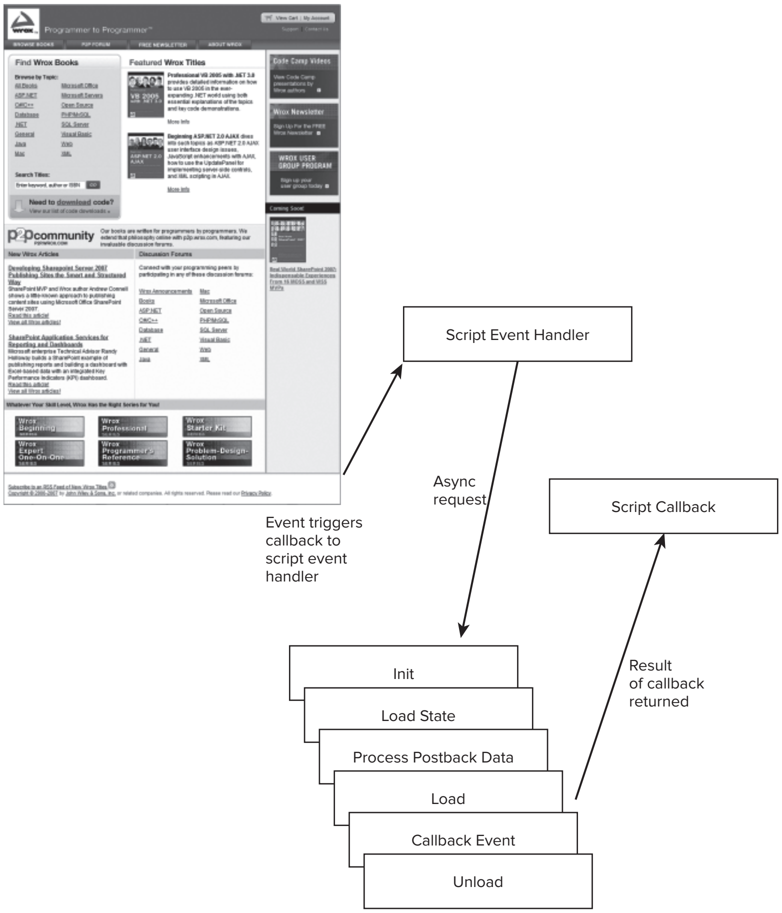


FIGURE 2-12

In this case, an event (again, such as a button click) causes the event to be posted to a script event handler (a JavaScript function) that sends off an asynchronous request to the Web server for processing. `ICallbackEventHandler` runs the request through a pipeline similar to what is used with the postback — but you notice that some of the larger steps (such as rendering the page) are excluded from the process chain. After the information is loaded, the result is returned to the script callback object. The script code then pushes this data into the Web page using JavaScript's capabilities to do this without refreshing the page. To understand how this all works, look at the simple example in the following section.

Using the Callback Feature — A Simple Approach

Begin examining the callback feature by looking at how a simple ASP.NET page uses it. For this example, you have only an HTML button control and a TextBox server control (the Web server control version). The idea is that when the end user clicks the button on the form, the callback service is initiated and a random number is populated into the text box. Listing 2-15 shows an example of this in action.

LISTING 2-15: Using the callback feature to populate a random value to a Web page

Available for download on Wrox.com

.aspx page (VB version)

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="RandomNumber.aspx.vb"
    Inherits="RandomNumber" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Callback Page</title>

    <script type="text/javascript">
        function GetNumber(){
            UseCallback();
        }

        function GetRandomNumberFromServer(TextBox1, context){
            document.forms[0].TextBox1.value = TextBox1;
        }
    </script>

</head>
<body>
    <form id="form1" runat="server">
    <div>
        <input id="Button1" type="button" value="Get Random Number"
         onclick="GetNumber()" />
        <br />
        <br />
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

VB (code-behind)

```
Partial Class RandomNumber
    Inherits System.Web.UI.Page
    Implements System.Web.UI.ICallbackEventHandler

    Dim _callbackResult As String = Nothing

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
       Handles Me.Load

        Dim cbReference As String =
           Page.ClientScript.GetCallbackEventReference(
             Me, "arg", "GetRandomNumberFromServer", "context")
        Dim cbScript As String = "function UseCallback(arg, context)" & _
           "{" & cbReference & ";" & "}"

        Page.ClientScript.RegisterClientScriptBlock(Me.GetType(),
           "UseCallback", cbScript, True)
    End Sub
    Public Sub RaiseCallbackEvent(ByVal eventArgument As String)
       Implements System.Web.UI.ICallbackEventHandler.RaiseCallbackEvent

       _callbackResult = Rnd().ToString()
    End Sub

    Public Function GetCallbackResult() As String _
       Implements System.Web.UI.ICallbackEventHandler.GetCallbackResult

        Return _callbackResult
    End Function
End Class
```

C# (code-behind)

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
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 RandomNumber : System.Web.UI.Page,
   System.Web.UI.ICallbackEventHandler
{
    private string _callbackResult = null;

    protected void Page_Load(object sender, EventArgs e)
    {
        string cbReference =
          Page.ClientScript.GetCallbackEventReference(this,
             "arg", "GetRandomNumberFromServer", "context");
        string cbScript = "function UseCallback(arg, context)" +
           "{" + cbReference + ";" + "}";

        Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
           "UseCallback", cbScript, true);
    }

    public void RaiseCallbackEvent(string eventArg)
    {
        Random rnd = new Random();
        _callbackResult = rnd.Next().ToString();
    }

    public string GetCallbackResult()
    {
        return _callbackResult;
    }
}
```

Filenames RandomNumber.aspx, RandomNumber.aspx.vb, and RandomNumber.aspx.cs

When this page is built and run in the browser, you get the results shown in Figure 2-13.

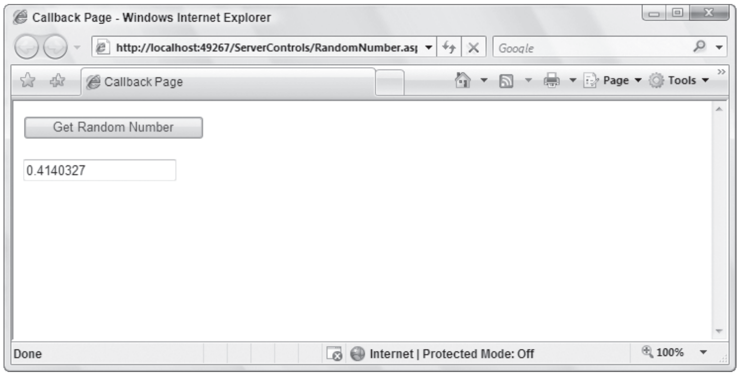

Callback Page - Windows Internet Explorer
http://localhost:49267/ServerControls/RandomNumber.as
Google
Callback Page
Page
Tools
Get Random Number
0.4140327
Done
Internet | Protected Mode: Off
100%

FIGURE 2-13

Clicking the button on the page invokes the client callback capabilities of the page, and the page then makes an asynchronous request to the code behind of the same page. After getting a response from this part of the page, the client script takes the retrieved value and places it inside the text box — all without doing a page refresh!

Now look at the `.aspx` page, which simply contains an HTML button control and a TextBox server control. Notice that a standard HTML button control is used because a typical `<asp:button>` control does not work here. No worries. When you work with the HTML button control, just be sure to include an `onclick` event to point to the JavaScript function that initiates this entire process:

```
<input id="Button1" type="button" value="Get Random Number"
 onclick="GetNumber()" />
```

You do not have to do anything else with the controls themselves. The final thing to include in the page is the client-side JavaScript functions to take care of the callback to the server-side functions. `GetNumber()` is the first JavaScript function that's instantiated. It starts the entire process by calling the name of the client script handler that is defined in the page's code behind. A `string` type result from `GetNumber()` is retrieved using the `GetRandomNumberFromServer()` function. `GetRandomNumberFromServer()` simply populates the `string` value retrieved and makes that the value of the Textbox control — specified by the value of the `ID` attribute of the server control (`TextBox1`):

```
<script type="text/javascript">
   function GetNumber(){
      UseCallback();
   }

   function GetRandomNumberFromServer(TextBox1, context){
      document.forms[0].TextBox1.value = TextBox1;
   }
</script>
```

Now turn your attention to the code behind.

The `Page` class of the Web page implements the `System.Web.UI.ICallbackEventHandler` interface:

```
Partial Class RandomNumber
    Inherits System.Web.UI.Page
    Implements System.Web.UI.ICallbackEventHandler

   ' Code here

End Class
```

This interface requires you to implement a couple of methods — the RaiseCallbackEvent and the GetCallbackResult methods, both of which work with the client script request. RaiseCallback Event enables you to do the work of retrieving the value from the page, but the value can be only of type string:

```
Public Sub RaiseCallbackEvent(ByVal eventArgument As String)
   Implements System.Web.UI.ICallbackEventHandler.RaiseCallbackEvent

   _callbackResult = Rnd().ToString()
End Sub
```

The GetCallbackResult is the method that actually grabs the returned value to be used:

```
Public Function GetCallbackResult() As String
   Implements System.Web.UI.ICallbackEventHandler.GetCallbackResult

        Return _callbackResult
End Function
```

In addition, the Page_Load event includes the creation and placement of the client callback script manager (the function that will manage requests and responses) on the client:

```
Dim cbReference As String = Page.ClientScript.GetCallbackEventReference(Me, "arg",
   "GetRandomNumberFromServer", "context")
Dim cbScript As String = "function UseCallback(arg, context)" &
   "{" & cbReference & ";" & "}"

Page.ClientScript.RegisterClientScriptBlock(Me.GetType(),
   "UseCallback", cbScript, True)
```

The function placed on the client for the callback capabilities is called UseCallback(). This string is then populated to the Web page itself using the Page.ClientScript.RegisterClientScripBlock that also puts <script> tags around the function on the page. Make sure that the name you use here is the same name you use in the client-side JavaScript function presented earlier.

In the end, you have a page that refreshes content without refreshing the overall page. This opens the door to a completely new area of possibilities. One caveat is that the callback capabilities described here use XmlHTTP and, therefore, the client browser needs to support XmlHTTP (Microsoft's Internet Explorer and FireFox do support this feature). Because of this, .NET Framework 2.0, 3.5, and 4 have the SupportsCallBack and the SupportsXmlHttp properties. To ensure this support, you could put a check in the page's code behind when the initial page is being generated. It might look similar to the following:

VB

```
If (Page.Request.Browser.SupportsXmlHTTP) Then

End If
```

C#

```
if (Page.Request.Browser.SupportsXmlHTTP == true) {

}
```

Using the Callback Feature with a Single Parameter

Now you will build a Web page that utilizes the callback feature but requires a parameter to retrieve a returned value. At the top of the page, place a text box that gathers input from the end user, a button, and another text box to populate the page with the result from the callback.

The page asks for a ZIP Code from the user and then uses the callback feature to instantiate an XML Web service request on the server. The Web service returns the latest weather for that particular ZIP Code in a string format. Listing 2-16 shows an example of the page.

Available for download on Wrox.com

LISTING 2-16: Using the callback feature with a Web service

.aspx page (VB version)

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="WSCallback.aspx.vb"
    Inherits="WSCallback" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Web Service Callback</title>

    <script type="text/javascript">
        function GetTemp(){
            var zipcode = document.forms[0].TextBox1.value;
            UseCallback(zipcode, "");
        }

        function GetTempFromServer(TextBox2, context){
            document.forms[0].TextBox2.value = "Zipcode: " +
            document.forms[0].TextBox1.value + " | Temp: " + TextBox2;
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <br />
        <input id="Button1" type="button" value="Get Temp" onclick="GetTemp()" />
        <br />
        <asp:TextBox ID="TextBox2" Runat="server" Width="400px">
        </asp:TextBox>
        <br />
        <br />
    </div>
    </form>
</body>
</html>
```

VB (code-behind)

```
Partial Class WSCallback
    Inherits System.Web.UI.Page
    Implements System.Web.UI.IcallbackEventHandler

    Dim _callbackResult As String = Nothing

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
       Handles Me.Load

        Dim cbReference As String = Page.ClientScript.GetCallbackEventReference(
           Me, "arg", "GetTempFromServer", "context")
        Dim cbScript As String = "function UseCallback(arg, context)" & _
           "{" & cbReference & ";" & "}"

        Page.ClientScript.RegisterClientScriptBlock(Me.GetType(),
           "UseCallback", cbScript, True)
    End Sub

    Public Sub RaiseCallbackEvent(ByVal eventArgument As String)
       Implements System.Web.UI.ICallbackEventHandler.RaiseCallbackEvent
```

```
        Dim ws As New Weather.TemperatureService
        _callbackResult = ws.getTemp(eventArgument).ToString()
    End Sub
    Public Function GetCallbackResult() As String
       Implements System.Web.UI.ICallbackEventHandler.GetCallbackResult

        Return _callbackResult
    End Function
End Class
```

C# (code-behind)

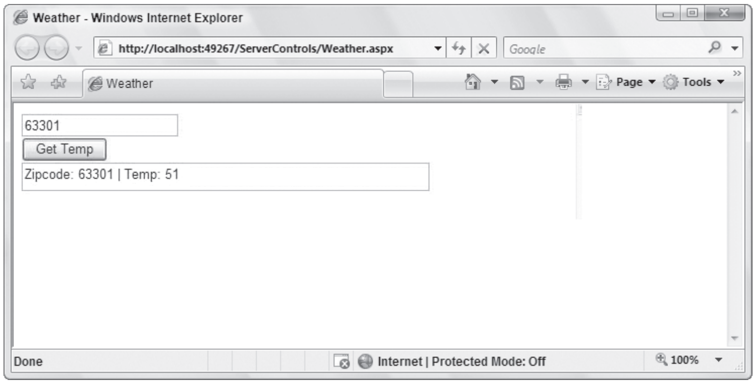
```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class WSCallback : System.Web.UI.Page,
   System.Web.UI.ICallbackEventHandler
{
    private string _callbackResult = null;

    protected void Page_Load(object sender, EventArgs e)
    {
        string cbReference = Page.ClientScript.GetCallbackEventReference(this,
           "arg", "GetTempFromServer", "context");
        string cbScript = "function UseCallback(arg, context)" +
           "{" + cbReference + ";" + "}";

        Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
           "UseCallback", cbScript, true);
    }

    public void RaiseCallbackEvent(string eventArg)
    {
        Weather.TemperatureService ws = new Weather.TemperatureService();
        _callbackResult = ws.getTemp(eventArg).ToString();
    }
    public string GetCallbackResult()
    {
        return _callbackResult;
    }
}
```

Filenames WSCallback.aspx, WSCallback.aspx.vb, and WSCallback.aspx.cs

What you do not see on this page from the listing is that a Web reference has been made to a theoretical remote Web service that returns the latest weather to the application based on a ZIP Code the user supplied.

For more information on working with Web services in your ASP.NET applications, check out Chapter 31.

After building and running this page, you get the results illustrated in Figure 2-14.


Weather - Windows Internet Explorer
http://localhost:49267/ServerControls/Weather.aspx
Google
Weather
Page
Tools
Get Temp
Done
Internet | Protected Mode: Off
100%

63301

Zipcode: 63301 | Temp: 51

FIGURE 2-14

The big difference with the client callback feature is that this example sends in a required parameter. That is done in the `GetTemp()` JavaScript function on the `.aspx` part of the page:

```
function GetTemp(){
   var zipcode = document.forms[0].TextBox1.value;
   UseCallback(zipcode, "");
}
```

The JavaScript function shows the population that the end user input into `TextBox1` and places its value in a variable called `zipcode` that is sent as a parameter in the `UseCallback()` method.

This example, like the previous one, updates the page without doing a complete page refresh.

Using the Callback Feature — A More Complex Example

So far, you have seen an example of using the callback feature to pull back a single item as well as to pull back a string whose output is based on a single parameter that was passed to the engine. The next example takes this operation one step further and pulls back a collection of results based upon a parameter provided.

This example works with an instance of the Northwind database found in SQL Server. For this example, create a single page that includes a TextBox server control and a button. Below that, place a table that will be populated with the customer details from the customer ID provided in the text box. The `.aspx` page for this example is provided in Listing 2-17.

Available for download on Wrox.com

LISTING 2-17: An ASP.NET page to collect the CustomerID from the end user

.aspx Page

```
<%@ Page Language="VB" AutoEventWireup="false"
    CodeFile="Default.aspx.vb" Inherits="_Default" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Customer Details</title>

    <script type="text/javascript">
        function GetCustomer(){
            var customerCode = document.forms[0].TextBox1.value;
            UseCallback(customerCode, "");
        }
```

```
            function GetCustDetailsFromServer(result, context){
                var i = result.split("|");
                customerID.innerHTML = i[0];
                companyName.innerHTML = i[1];
                contactName.innerHTML = i[2];
                contactTitle.innerHTML = i[3];
                address.innerHTML = i[4];
                city.innerHTML = i[5];
                region.innerHTML = i[6];
                postalCode.innerHTML = i[7];
                country.innerHTML = i[8];
                phone.innerHTML = i[9];
                fax.innerHTML = i[10];
            }
        </script>
    </head>
    <body>
        <form id="form1" runat="server">
        <div>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>&nbsp;
            <input id="Button1" type="button" value="Get Customer Details"
             onclick="GetCustomer()" /><br />
            <br />
          <table cellspacing="0" cellpadding="4" rules="all" border="1"
           id="DetailsView1"
           style="background-color:White;border-color:#3366CC;border-width:1px;
              border-style:None;height:50px;width:400px;border-collapse:collapse;">
            <tr style="color:#003399;background-color:White;">
               <td>CustomerID</td><td><span id="customerID" /></td>
            </tr><tr style="color:#003399;background-color:White;">
               <td>CompanyName</td><td><span id="companyName" /></td>
            </tr><tr style="color:#003399;background-color:White;">
               <td>ContactName</td><td><span id="contactName" /></td>
            </tr><tr style="color:#003399;background-color:White;">
               <td>ContactTitle</td><td><span id="contactTitle" /></td>
            </tr><tr style="color:#003399;background-color:White;">
               <td>Address</td><td><span id="address" /></td>
            </tr><tr style="color:#003399;background-color:White;">
               <td>City</td><td><span id="city" /></td>
            </tr><tr style="color:#003399;background-color:White;">
               <td>Region</td><td><span id="region" /></td>
            </tr><tr style="color:#003399;background-color:White;">
               <td>PostalCode</td><td><span id="postalCode" /></td>
            </tr><tr style="color:#003399;background-color:White;">
               <td>Country</td><td><span id="country" /></td>
            </tr><tr style="color:#003399;background-color:White;">
               <td>Phone</td><td><span id="phone" /></td>
            </tr><tr style="color:#003399;background-color:White;">
               <td>Fax</td><td><span id="fax" /></td>
            </tr>
          </table>
        </div>
        </form>
    </body>
    </html>
```

Filename CallbackNorthwind.aspx

As in the previous examples, two JavaScript functions are contained in the page. The first, `GetCustomer()`, is the function that passes in the parameter to be processed by the code-behind file on the application server. This is quite similar to what appeared in the previous example.

The second JavaScript function, however, is different. Looking over this function, you can see that it is expecting a long string of multiple values:

```
function GetCustDetailsFromServer(result, context){
   var i = result.split("|");
   customerID.innerHTML = i[0];
   companyName.innerHTML = i[1];
   contactName.innerHTML = i[2];
   contactTitle.innerHTML = i[3];
   address.innerHTML = i[4];
   city.innerHTML = i[5];
   region.innerHTML = i[6];
   postalCode.innerHTML = i[7];
   country.innerHTML = i[8];
   phone.innerHTML = i[9];
   fax.innerHTML = i[10];
}
```

The multiple results expected are constructed in a pipe-delimited string, and each of the values is placed into an array. Then each string item in the array is assigned to a particular `<span>` tag in the ASP.NET page. For instance, look at the following bit of code:

```
customerID.innerHTML = i[0];
```

The `i[0]` variable is the first item found in the pipe-delimited string, and it is assigned to the `customerID` item on the page. This `customerID` identifier comes from the following `<span>` tag found in the table:

```
<span id="customerID" />
```

Now, turn your attention to the code-behind file for this page, as shown in Listing 2-18.

LISTING 2-18: The code-behind file for the Customer Details page

Available for download on Wrox.com

VB

```
Imports System.Data
Imports System.Data.SqlClient

Partial Class CallbackNorthwind
    Inherits System.Web.UI.Page
    Implements System.Web.UI.ICallbackEventHandler

    Dim _callbackResult As String = Nothing

    Public Function GetCallbackResult() As String _
       Implements System.Web.UI.ICallbackEventHandler.GetCallbackResult
        Return _callbackResult
    End Function

    Public Sub RaiseCallbackEvent(ByVal eventArgument As String)
       Implements System.Web.UI.ICallbackEventHandler.RaiseCallbackEvent
        Dim conn As SqlConnection = New _
           SqlConnection("Data Source=.;Initial Catalog=Northwind;User ID=sa")
        Dim cmd As SqlCommand = New _
           SqlCommand("Select * From Customers Where CustomerID ='" &
           eventArgument & "'", conn)

        conn.Open()

        Dim MyReader As SqlDataReader
        MyReader = cmd.ExecuteReader(CommandBehavior.CloseConnection)
        Dim MyValues(10) As String

        While MyReader.Read()
            MyValues(0) = MyReader("CustomerID").ToString()
            MyValues(1) = MyReader("CompanyName").ToString()
            MyValues(2) = MyReader("ContactName").ToString()
            MyValues(3) = MyReader("ContactTitle").ToString()
            MyValues(4) = MyReader("Address").ToString()
            MyValues(5) = MyReader("City").ToString()
            MyValues(6) = MyReader("Region").ToString()
            MyValues(7) = MyReader("PostalCode").ToString()
            MyValues(8) = MyReader("Country").ToString()
            MyValues(9) = MyReader("Phone").ToString()
            MyValues(10) = MyReader("Fax").ToString()
        End While

        Conn.Close()

        _callbackResult = String.Join("|", MyValues)
    End Sub

    Protected Sub Page_Load(ByVal sender As Object, _
       ByVal e As System.EventArgs) Handles Me.Load
        Dim cbReference As String = _
           Page.ClientScript.GetCallbackEventReference(Me, "arg", _
              "GetCustDetailsFromServer", "context")
        Dim cbScript As String = "function UseCallback(arg, context)" & _
           "{" & cbReference & ";" & "}"

        Page.ClientScript.RegisterClientScriptBlock(Me.GetType(), _
           "UseCallback", cbScript, True)
    End Sub
End Class
```

C#

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
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.Data.SqlClient;

public partial class CallbackNorthwind : System.Web.UI.Page,
   System.Web.UI.ICallbackEventHandler
{
    private string _callbackResult = null;

    protected void Page_Load(object sender, EventArgs e)
    {
        string cbReference = Page.ClientScript.GetCallbackEventReference(this,
           "arg", "GetCustDetailsFromServer", "context");
        string cbScript = "function UseCallback(arg, context)" +
           "{" + cbReference + ";" + "}";

        Page.ClientScript.RegisterClientScriptBlock(this.GetType(),
           "UseCallback", cbScript, true);
    }
```

contiuues

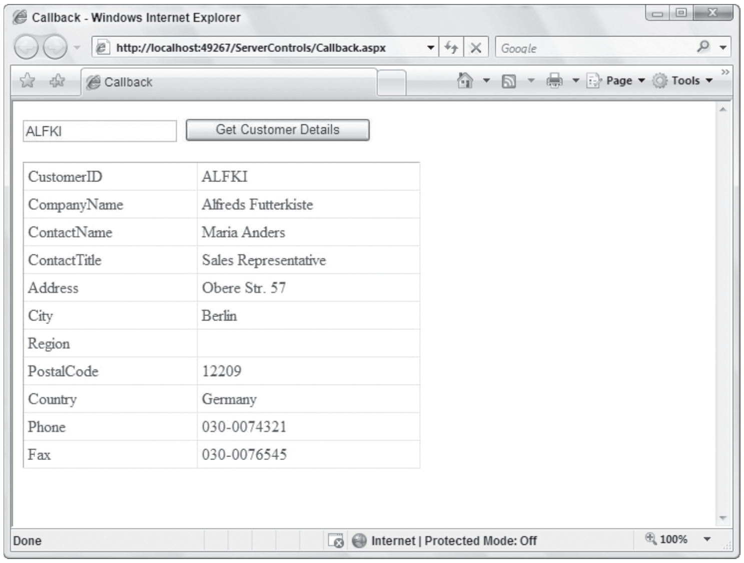
LISTING 2-18 *(continued)*

```
        #region ICallbackEventHandler Members

        public string GetCallbackResult()
        {
            return _callbackResult;
        }

        public void RaiseCallbackEvent(string eventArgument)
        {
            SqlConnection conn = new
               SqlConnection("Data Source=.;Initial Catalog=Northwind;User ID=sa");
            SqlCommand cmd = new
               SqlCommand("Select * From Customers Where CustomerID ='" +
               eventArgument + "'", conn);

            conn.Open();

            SqlDataReader MyReader;
            MyReader = cmd.ExecuteReader(CommandBehavior.CloseConnection);

            string[] MyValues = new string[11];

            while (MyReader.Read())
            {
                MyValues[0] = MyReader["CustomerID"].ToString();
                MyValues[1] = MyReader["CompanyName"].ToString();
                MyValues[2] = MyReader["ContactName"].ToString();
                MyValues[3] = MyReader["ContactTitle"].ToString();
                MyValues[4] = MyReader["Address"].ToString();
                MyValues[5] = MyReader["City"].ToString();
                MyValues[6] = MyReader["Region"].ToString();
                MyValues[7] = MyReader["PostalCode"].ToString();
                MyValues[8] = MyReader["Country"].ToString();
                MyValues[9] = MyReader["Phone"].ToString();
                MyValues[10] = MyReader["Fax"].ToString();
            }

            _callbackResult = String.Join("|", MyValues);
        }

        #endregion
    }
```

Filenames CallbackNorthwind.aspx.vb and CallbackNorthwind.aspx.cs

Much of this document is quite similar to the document in the previous example using the callback feature. The big difference comes in the `RaiseCallbackEvent()` method. This method first performs a `SELECT` statement on the Customers database based upon the `CustomerID` passed in via the `eventArgument` variable. The result retrieved from this `SELECT` statement is then made part of a string array, which is finally concatenated using the `String.Join()` method before being passed back as the value of the `_callbackResult` object.

With this code in place, you can now populate an entire table of data using the callback feature. This means that the table is populated with no need to refresh the page. The results from this code operation are presented in Figure 2-15.


Callback - Windows Internet Explorer
http://localhost:49267/ServerControls/Callback.aspx
Callback
ALFKI
Get Customer Details
Done
Internet | Protected Mode: Off
100%

CustomerID	ALFKI
CompanyName	Alfreds Futterkiste
ContactName	Maria Anders
ContactTitle	Sales Representative
Address	Obere Str. 57
City	Berlin
Region	
PostalCode	12209
Country	Germany
Phone	030-0074321
Fax	030-0076545

FIGURE 2-15

SUMMARY

This chapter gave you one of the core building blocks of an ASP.NET page — the server control. The server control is an object-oriented approach to page development that encapsulates page elements into modifiable and expandable components.

The chapter also introduced you to how to customize the look-and-feel of your server controls using Cascading Style Sheets (CSS). Working with CSS in ASP.NET 4 is easy and quick, especially if you have Visual Studio 2010 to assist you. Finally, this chapter looked at both using HTML server controls and adding JavaScript to your pages to modify the behaviors of your controls.

3 ASP.NET Web Server Controls

WHAT'S IN THIS CHAPTER?

- Reviewing key Web server controls
- Differentiating between Web server control features
- Removing items from a collection

Of the two types of server controls, HTML server controls and Web server controls, the latter is considered the more powerful and flexible. The previous chapter looked at how to use HTML server controls in applications. HTML server controls enable you to manipulate HTML elements from your server-side code. On the other hand, Web server controls are powerful because they are not explicitly tied to specific HTML elements; rather, they are more closely aligned to the specific functionality that you want to generate. As you will see throughout this chapter, Web server controls can be very simple or rather complex depending on the control you are working with.

The purpose of the large collection of controls is to make you more productive. These controls give you advanced functionality that, in the past, you would have had to laboriously program or simply omit. In the classic ASP days, for example, few calendars were used on Internet Web sites. With the introduction of the Calendar server control in ASP.NET 1.0, calendar creation on a site became a trivial task. Building an image map on top of an image was another task that was difficult to achieve in ASP.NET 1.*x*, but this capability was introduced as a new server control in ASP.NET 2.0. As ASP.NET evolves through the releases, new controls are always added that help to make you a more productive Web developer.

This chapter introduces some of the available Web server controls. The first part of the chapter focuses on the Web server controls that were around since the first days of ASP.NET. Then the chapter explores the server controls that were introduced after the initial release of ASP.NET. This chapter does not discuss every possible control because some server controls are introduced and covered in other chapters throughout the book as they might be more related to that particular topic.

AN OVERVIEW OF WEB SERVER CONTROLS

The Web server control is ASP.NET's most-used component. Although you may have seen a lot of potential uses of the HTML server controls shown in the previous chapter, Web server controls are definitely a notch higher in capability. They allow for a higher level of functionality that becomes more apparent as you work with them.

The HTML server controls provided by ASP.NET work in that they map to specific HTML elements. You control the output by working with the HTML attributes that the HTML element provides. The attributes can be changed dynamically on the server side before they are finally output to the client. There is a lot of power in this, and you have some HTML server control capabilities that you simply do not have when you work with Web server controls.

Web server controls work differently. They do not map to specific HTML elements, but instead enable you to define functionality, capability, and appearance without the attributes that are available to you through a collection of HTML elements. When constructing a Web page that is made up of Web server controls, you are describing the functionality, the look-and-feel, and the behavior of your page elements. You then let ASP.NET decide how to output this construction. The output, of course, is based on the capabilities of the container that is making the request. This means that each requestor might see a different HTML output because each is requesting the same page with a different browser type or version. ASP.NET takes care of all the browser detection and the work associated with it on your behalf.

Unlike HTML server controls, Web server controls are not only available for working with common Web page form elements (such as text boxes and buttons), but they can also bring some advanced capabilities and functionality to your Web pages. For instance, one common feature of many Web applications is a calendar. No HTML form element places a calendar on your Web forms, but a Web server control from ASP.NET can provide your application with a full-fledged calendar, including some advanced capabilities. In the past, adding calendars to your Web pages was not a small programming task. Today, adding calendars with ASP.NET is rather simple and is achieved with a single line of code!

Remember that when you are constructing your Web server controls, you are actually constructing a control — *a set of instructions* — that is meant for the server (not the client). By default, all Web server controls provided by ASP.NET use an `asp:` at the beginning of the control declaration. The following is a typical Web server control:

```
<asp:Label ID="Label1" runat="server" Text="Hello World"></asp:Label>
```

Like HTML server controls, Web server controls require an `ID` attribute to reference the control in the server-side code, as well as a `runat="server"` attribute declaration. As you do for other XML-based elements, you need to properly open and close Web server controls using XML syntax rules. In the preceding example, you can see the `<asp:Label>` control has a closing `</asp:Label>` element associated with it. You could have also closed this element using the following syntax:

```
<asp:Label ID="Label1" Runat="server" Text="Hello World" />
```

The rest of this chapter examines some of the Web server controls available to you in ASP.NET.

THE LABEL SERVER CONTROL

The Label server control is used to display text in the browser. Because this is a server control, you can dynamically alter the text from your server-side code. As you saw from the preceding examples of using the `<asp:Label>` control, the control uses the `Text` attribute to assign the content of the control as shown here:

```
<asp:Label ID="Label1" runat="server" Text="Hello World" />
```

Instead of using the `Text` attribute, however, you can place the content to be displayed between the `<asp:Label>` elements like this:

```
<asp:Label ID="Label1" runat="server">Hello World</asp:Label>
```

You can also provide content for the control through programmatic means, as illustrated in Listing 3-1.

LISTING 3-1: Programmatically providing text to the Label control

VB

```
Label1.Text = "Hello ASP.NET"
```

C#

```
Label1.Text = "Hello ASP.NET";
```

The Label server control has always been a control that simply showed text. Ever since ASP.NET 2.0, it has a little bit of extra functionality. The big change since this release of the framework is that you can now give items in your form hot-key functionality (also known as *accelerator* keys). This causes the page to focus on a particular server control that you declaratively assign to a specific hot-key press (for example, using Alt+N to focus on the first text box on the form).

A hot key is a quick way for the end user to initiate an action on the page. For instance, if you use Microsoft Internet Explorer, you can press Ctrl+N to open a new instance of IE. Hot keys have always been quite common in thick-client applications (Windows Forms), and now you can use them in ASP.NET. Listing 3-2 shows an example of how to give hot-key functionality to two text boxes on a form.

LISTING 3-2: Using the Label server control to provide hot-key functionality

Available for download on Wrox.com

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Label Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <p>
            <asp:Label ID="Label1" runat="server" AccessKey="N"
             AssociatedControlID="Textbox1">User<u>n</u>ame</asp:Label>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></p>
        <p>
            <asp:Label ID="Label2" runat="server" AccessKey="P"
             AssociatedControlID="Textbox2"><u>P</u>assword</asp:Label>
            <asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox></p>
        <p>
            <asp:Button ID="Button1" runat="server" Text="Submit" />
        </p>
    </form>
</body>
</html>
```

Hot keys are assigned with the `AccessKey` attribute. In this case, `Label1` uses `N`, and `Label2` uses `P`. The second attribute for the Label control is the `AssociatedControlID` attribute. The `String` value placed here associates the Label control with another server control on the form. The value must be one of the other server controls on the form. If not, the page gives you an error when invoked.

With these two controls in place, when the page is called in the browser, you can press Alt+N or Alt+P to automatically focus on a particular text box in the form. In Figure 3-1, HTML-declared underlines indicate the letters to be pressed along with the Alt key to create focus on the control adjoining the text. This is not required, but we highly recommend it because it is what the end user expects when working with hot keys. In this example, the letter `n` in `Username` and the letter `P` in `Password` are underlined.

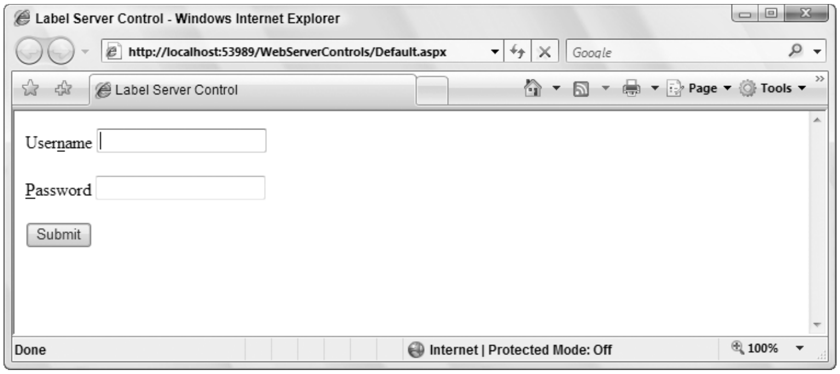

Label Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Label Server Control
Page
Tools
Username
Password
Submit
Done
Internet | Protected Mode: Off
100%

FIGURE 3-1

When working with hot keys, be aware that not all letters are available to use with the Alt key. Microsoft Internet Explorer already uses Alt+F, E, V, I, O, T, A, W, and H. If you use any of these letters, IE actions supersede any actions you place on the page.

THE LITERAL SERVER CONTROL

The Literal server control works very much like the Label server control does. This control was always used in the past for text that you wanted to push out to the browser but keep unchanged in the process (a literal state). A Label control alters the output by placing `<span>` elements around the text as shown:

```
<span id="Label1">Here is some text</span>
```

The Literal control just outputs the text without the `<span>` elements. One feature found in this server control is the attribute `Mode`. This attribute enables you to dictate how the text assigned to the control is interpreted by the ASP.NET engine.

If you place some HTML code in the string that is output (for instance, `<b>Here is some text</b>`), the Literal control outputs just that and the consuming browser shows the text as bold:

```
Here is some text
```

Try using the `Mode` attribute as illustrated here:

```
<asp:Literal ID="Literal1" runat="server" Mode="Encode"
 Text="<b>Here is some text</b>"></asp:Literal>
```

Adding `Mode="Encode"` encodes the output before it is received by the consuming application:

```
&lt;b&gt;Label&lt;/b&gt;
```

Now, instead of the text being converted to a bold font, the `<b>` elements are displayed:

```
<b>Here is some text</b>
```

This is ideal if you want to display code in your application. Other values for the `Mode` attribute include `Transform` and `PassThrough`. `Transform` looks at the consumer and includes or removes elements as needed. For instance, not all devices accept HTML elements so, if the value of the `Mode` attribute is set to `Transform`, these elements are removed from the string before it is sent to the consuming application. A value of `PassThrough` for the `Mode` property means that the text is sent to the consuming application without any changes being made to the string.

THE TEXTBOX SERVER CONTROL

One of the main features of Web pages is to offer forms that end users can use to submit their information for collection. The TextBox server control is one of the most used controls in this space. As its name suggests, the control provides a text box on the form that enables the end user to input text. You can map the TextBox control to three different HTML elements used in your forms.

First, the TextBox control can be used as a standard HTML text box, as shown in the following code snippet:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
```

This code creates a text box on the form that looks like the one shown in Figure 3-2.

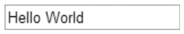

Hello World

FIGURE 3-2

Second, the TextBox control can allow end users to input their passwords into a form. This is done by changing the `TextMode` attribute of the TextBox control to `Password`, as illustrated here:

```
<asp:TextBox ID="TextBox1" runat="server" TextMode="Password"></asp:TextBox>
```

When asking end users for their passwords through the browser, it is best practice to provide a text box that encodes the content placed in this form element. Using an attribute and value of `TextMode="Password"` ensures that the text is encoded with either a star (`*`) or a dot, as shown in Figure 3-3.

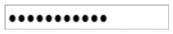

•••••••••••

FIGURE 3-3

Third, the TextBox server control can be used as a multiline text box. The code for accomplishing this task is as follows:

```
<asp:TextBox ID="TextBox1" runat="server" TextMode="MultiLine"
 Width="300px" Height="150px"></asp:TextBox>
```

Giving the `TextMode` attribute a value of `MultiLine` creates a multilined text box in which the end user can enter a larger amount of text in the form. The `Width` and `Height` attributes set the size of the text area, but these are optional attributes — without them, the text area is produced in its smallest size. Figure 3-4 shows the use of the preceding code after adding some text.

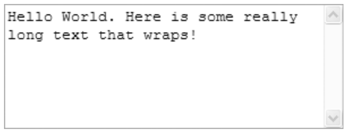

Hello World. Here is some really long text that wraps!

FIGURE 3-4

When working with a multilined text box, be aware of the `Wrap` attribute. When set to `True` (which is the default), the text entered into the text area wraps to the next line if needed. When set to `False`, the end user can type continuously in a single line until she presses the Enter key, which brings the cursor down to the next line.

Using the Focus() Method

Because the TextBox server control is derived from the base class of `WebControl`, one of the methods available to it is `Focus()`. The `Focus()` method enables you to dynamically place the end user's cursor in an appointed form element (not just the TextBox control, but in any of the server controls derived from the `WebControl` class). With that said, it is probably most often used with the TextBox control, as illustrated in Listing 3-3.

LISTING 3-3: Using the Focus() method with the TextBox control

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
   TextBox1.Focus()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
   TextBox1.Focus();
}
```

When the page using this method is loaded in the browser, the cursor is already placed inside of the text box, ready for you to start typing. There is no need to move your mouse to get the cursor in place so you can start entering information in the form. This is ideal for those folks who take a keyboard approach to working with forms.

Using AutoPostBack

ASP.NET pages work in an event-driven way. When an action on a Web page triggers an event, server-side code is initiated. One of the more common events is an end user clicking a button on the form. If you double-click the button in Design view of Visual Studio 2010, you can see the code page with the structure of the `Button1_Click` event already in place. This is because `OnClick` is the most common event of the Button control. Double-clicking the TextBox control constructs an `OnTextChanged` event. This event is triggered when the end user moves the cursor focus outside the text box, either by clicking another element on the page after entering something into a text box, or by simply tabbing out of the text box. The use of this event is shown in Listing 3-4.

LISTING 3-4: Triggering an event when a TextBox change occurs

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" %>

<script runat="server">
```

continues

LISTING 3-4 *(continued)*

```
    Protected Sub TextBox1_TextChanged(ByVal sender As Object,
       ByVal e As System.EventArgs)

        Response.Write("OnTextChanged event triggered")
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object,
       ByVal e As System.EventArgs)

        Response.Write("OnClick event triggered")
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>OnTextChanged Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server" AutoPostBack="True"
         OnTextChanged="TextBox1_TextChanged"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" Text="Button"
         OnClick="Button1_Click" />
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void TextBox1_TextChanged(object sender, EventArgs e)
    {
        Response.Write("OnTextChanged event triggered");
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Write("OnClick event triggered");
    }
</script>
```

As you build and run this page, notice that you can type something in the text box, but once you tab out of it, the `OnTextChanged` event is triggered and the code contained in the `TextBox1_TextChanged` event runs. To make this work, you must add the `AutoPostBack` attribute to the TextBox control and set it to `True`. This causes the Web page to look for any text changes prior to an actual page postback. For the `AutoPostBack` feature to work, the browser viewing the page must support ECMAScript.

Using AutoCompleteType

You want the forms you build for your Web applications to be as simple to use as possible. You want to make them easy and quick for the end user to fill out the information and proceed. If you make a form too onerous, the people who come to your site may leave without completing it.

One of the great capabilities for any Web form is smart auto-completion. You may have seen this yourself when you visited a site for the first time. As you start to fill out information in a form, a drop-down list appears below the text box as you type, showing you a value that you have typed in a previous form. The plain text box you were working with has become a smart text box. Figure 3-5 shows an example of this feature.

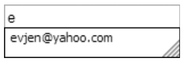


FIGURE 3-5

A great aspect of the TextBox control is the `AutoCompleteType` attribute, which enables you to apply the auto-completion feature to your own forms. You have to help the text boxes on your form to recognize the type of information that they should be looking for. What does that mean? Well, first look at the possible values of the `AutoCompleteType` attribute:

```
BusinessCity            Disabled              HomeStreetAddress
BusinessCountryRegion   DisplayName           HomeZipCode
BusinessFax             Email                 JobTitle
BusinessPhone           FirstName             LastName
BusinessState           Gender                MiddleName
BusinessStateAddress    HomeCity              None
BusinessUrl             HomeCountryRegion     Notes
BusinessZipCode         HomeFax               Office
Cellular                Homepage              Pager
Company                 HomePhone             Search
Department              HomeState
```

From this list, you can see that if your text box is asking for the end user's home street address, you want to use the following in your TextBox control:

```
<asp:TextBox ID="TextBox1" runat="server"
 AutoCompleteType="HomeStreetAddress"></asp:TextBox>
```

As you view the source of the text box you created, you can see that the following construction has occurred:

```
<input name="TextBox1" type="text" vcard_name="vCard.Home.StreetAddress"
 id="TextBox1" />
```

This feature makes your forms easier to work with. Yes, it is a simple thing but sometimes the little things keep the viewers coming back again and again to your Web site.

THE BUTTON SERVER CONTROL

Another common control for your Web forms is a button that can be constructed using the Button server control. Buttons are the usual element used to submit forms. Most of the time you are simply dealing with items contained in your forms through the Button control's `OnClick` event, as illustrated in Listing 3-5.

LISTING 3-5: The Button control's OnClick event

VB

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
  ' Code here
End Sub
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
   // Code here
}
```

The Button control is one of the easier controls to use, but there are a couple of properties of which you must be aware: `CausesValidation` and `CommandName`. They are discussed in the following sections.

The CausesValidation Property

If you have more than one button on your Web page and you are working with the validation server controls, you may not want to fire the validation for each button on the form. Setting the `CausesValidation` property to `False` is a way to use a button that will not fire the validation process. This is explained in more detail in Chapter 4.

The CommandName Property

You can have multiple buttons on your form all working from a single event. The nice thing is that you can also tag the buttons so that the code can make logical decisions based on which button on the form was clicked. You must construct your Button controls in the manner illustrated in Listing 3-6 to take advantage of this behavior.

LISTING 3-6: Constructing multiple Button controls to work from a single function

```
<asp:Button ID="Button1" runat="server" Text="Button 1"
 OnCommand="Button_Command" CommandName="DoSomething1" />
<asp:Button ID="Button2" runat="server" Text="Button 2"
 OnCommand="Button_Command" CommandName="DoSomething2" />
```

Looking at these two instances of the Button control, you should pay attention to several things. The first thing to notice is what is not present — any attribute mention of an `OnClick` event. Instead, you use the `OnCommand` event, which points to an event called `Button_Command`. You can see that both Button controls are working from the same event. How does the event differentiate between the two buttons being clicked? Through the value placed in the `CommandName` property. In this case, they are indeed separate values — `DoSomething1` and `DoSomething2`.

The next step is to create the `Button_Command` event to deal with both these buttons by simply typing one out or by selecting the `Command` event from the drop-down list of available events for the Button control from the code view of Visual Studio. In either case, you should end up with an event like the one shown in Listing 3-7.

LISTING 3-7: The Button_Command event

VB

```
Protected Sub Button_Command(ByVal sender As Object,
  ByVal e As System.Web.UI.WebControls.CommandEventArgs)

   Select Case e.CommandName
      Case "DoSomething1"
         Response.Write("Button 1 was selected")
      Case "DoSomething2"
         Response.Write("Button 2 was selected")
   End Select

End Sub
```

C#

```
protected void Button_Command(Object sender,
   System.Web.UI.WebControls.CommandEventArgs e)
{
   switch (e.CommandName)
   {
      case("DoSomething1"):
         Response.Write("Button 1 was selected");
         break;
      case("DoSomething2"):
         Response.Write("Button 2 was selected");
         break;
   }
}
```

Notice that this method uses `System.Web.UI.WebControls.CommandEventArgs` instead of the typical `System.EventArgs`. This gives you access to the member `CommandName` used in the `Select Case` (`switch`) statement as `e.CommandName`. Using this object, you can check for the value of the `CommandName` property used by the button that was clicked on the form and take a specific action based upon the value passed.

You can add some parameters to be passed in to the `Command` event beyond what is defined in the `CommandName` property. You do this by using the Button control's `CommandArgument` property. Adding values to the property enables you to define items a bit more granularly if you want. You can get at this value via server-side code using `e.CommandArgument` from the `CommandEventArgs` object.

Buttons That Work with Client-Side JavaScript

Buttons are frequently used for submitting information and causing actions to occur on a Web page. Before ASP.NET 1.0/1.1, people intermingled quite a bit of JavaScript in their pages to fire JavaScript events when a button was clicked. The process became more cumbersome in ASP.NET 1.0/1.1, but ever since ASP.NET 2.0, it has been much easier.

You can create a page that has a JavaScript event, as well as a server-side event, triggered when the button is clicked, as illustrated in Listing 3-8.

LISTING 3-8: Two types of events for the button

Available for download on Wrox.com

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, _
       ByVal e As System.EventArgs)

        Response.Write("Postback!")
    End Sub
</script>

<script language="javascript">
   function AlertHello()
   {
      alert('Hello ASP.NET');
   }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Button Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Button ID="Button1" runat="server" Text="Button"
         OnClientClick="AlertHello()" OnClick="Button1_Click" />
    </form>
</body>

</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Response.Write("Postback!");
    }
</script>
```

The first thing to notice is the attribute for the Button server control: `OnClientClick`. It points to the client-side function, unlike the `OnClick` attribute that points to the server-side event. This example uses a JavaScript function called `AlertHello()`.

One cool thing about Visual Studio 2010 is that it can work with server-side script tags that are right alongside client-side script tags. It all works together seamlessly. In the example, after the JavaScript alert dialog is issued (see Figure 3-6) and the end user clicks OK, the page posts back as the server-side event is triggered.

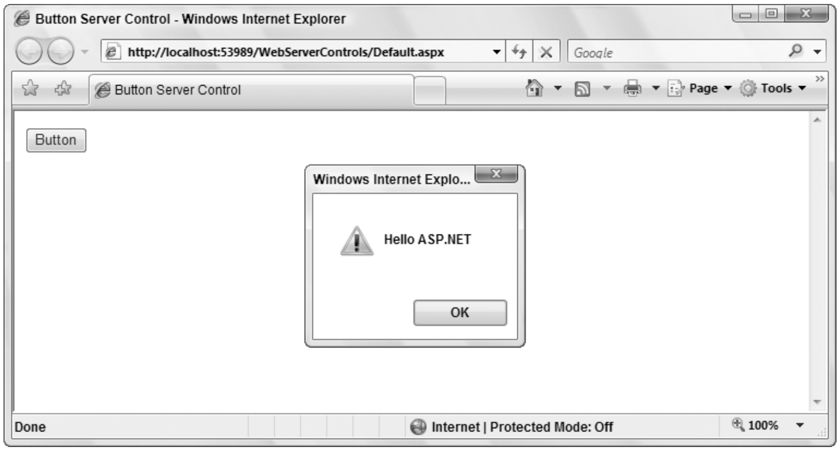

Button Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
Button Server Control
Page
Tools
Button
Windows Internet Explo...
Hello ASP.NET
OK
Done
Internet | Protected Mode: Off
100%

FIGURE 3-6

Another interesting attribute for the button controls is `PostBackUrl`. It enables you to perform cross-page posting, instead of simply posting your ASP.NET pages back to the same page, as shown in the following example:

```
<asp:Button ID="Button2" runat="server" Text="Submit page to Page2.aspx"
 PostBackUrl="Page2.aspx" />
```

Cross-page posting is covered in greater detail in Chapter 1.

THE LINKBUTTON SERVER CONTROL

The LinkButton server control is a variation of the Button control. It is the same except that the LinkButton control takes the form of a hyperlink. Nevertheless, it is not a typical hyperlink. When the end user clicks the link, it behaves like a button. This is an ideal control to use if you have a large number of buttons on your Web form.

A LinkButton server control is constructed as follows:

```
<asp:LinkButton ID="LinkButton1" Runat="server" OnClick="LinkButton1_Click">
   Submit your name to our database
</asp:LinkButton>
```

Using the LinkButton control gives you the results shown in Figure 3-7.

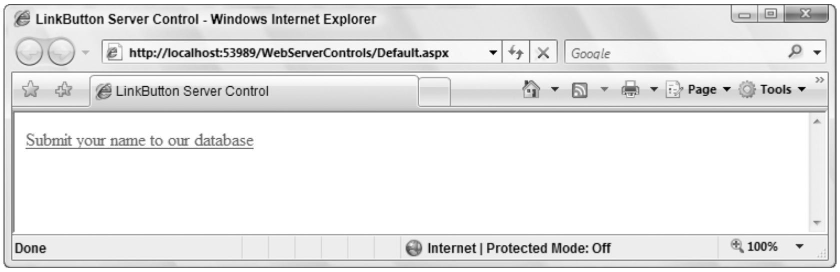

LinkButton Server Control - Windows Internet Explorer
http://localhost:53989/WebServerControls/Default.aspx
Google
LinkButton Server Control
Page
Tools
Submit your name to our database
Done
Internet | Protected Mode: Off
100%

FIGURE 3-7