Pro ASP.NET 2.0 in C# 2005

Matthew MacDonald and Mario Szpuszta,
Revising Authors

K. Scott Allen
James Avery
Russ Basiura
Mike Batongbacal
Marco Bellinaso
Matt Butler
Andreas Eide
Daniel Cazzulino
Michael Clark
Richard Conway
Robert Eisenberg
Brady Gaster
James Greenwood
Kevin Hoffman
Erik Johansson
Angelo Kastroulis
Dan Kent
Sitaraman Lakshminarayanan
Don Lee
Christopher Miller
Matt Milner
Jan Narkiewicz
Matt Odhner
Ryan O'Keefe
Andrew Reid
Matthew Reynolds
Enrico Sabbadin
Bill Sempf
Doug Seven
Srinivasa Sivakumar
Thiru Thangarathinam
Doug Thews

Apress®

Pro ASP.NET 2.0 in C# 2005

ISBN (pbk): 1-59059-496-7

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Ewan Buckingham
Technical Reviewers: Robert Lair, Jason Lefebvre
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Associate Publisher: Grace Wong
Project Manager: Kylie Johnston
Copy Edit Manager: Nicole LeClerc
Copy Editor: Kim Wimpsett
Assistant Production Director: Kari Brooks-Copony
Production Editor: Laura Cheu
Compositor: Dina Quan
Proofreaders: Liz Welch and Lori Bring
Indexer: Broccoli Information Management
Artist: Kinetic Publishing Services, LLC
Interior Designer: Diana Van Winkle
Cover Designer: Kurt Krames
Manufacturing Manager: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section.

CHAPTER 2

Visual Studio 2005

With ASP.NET, you have several choices for developing web applications. If you're inclined (and don't mind the work), you can code every web page and class by hand using a bare-bones text editor. This approach is appealingly straightforward but tedious and error-prone for anything other than a simple page. Professional ASP.NET developers rarely go this route.

Instead, almost all large-scale ASP.NET websites are built using Visual Studio. This professional development tool supports a rich set of design tools, including a legendary set of debugging tools and IntelliSense, which catches errors and offers suggestions as you type. Visual Studio also supports the robust code-behind model, which separates the .NET code you write from the web-page markup tags. To seal the deal, Visual Studio 2005 adds a built-in test web server that makes debugging Web sites easy and hassle free.

In this chapter, you'll tour the Visual Studio IDE and explore its key features. You'll also learn about the coding model used for ASP.NET 2.0 web pages.

Note Visual Studio 2005 is available in several versions. This chapter assumes you are using the full Visual Studio 2005 Professional or Visual Studio 2005 Team System. If you are using the scaled-down Visual Web Developer 2005 Express Edition, you will lose some features. Most notably, you won't be able to create separate components with class library projects.

VISUAL STUDIO 2005 CHANGES

If you're a seasoned ASP.NET developer, you're most interested in what's new in Visual Studio 2005. Although most of the editing features and debugging tools in Visual Studio 2005 are the same as those in Visual Studio 2003, the underlying model has a few significant changes. Here are the four most significant changes, all of which you'll learn more about in this chapter:

- **Projectless development**: Visual Studio no longer clutters your web projects with extra development files (such as .csproj and .sln). One obvious benefit of this model is that you can deploy exactly what you develop, without needing to filter out just a subset of the files. However, as you'll see in this chapter, the concept of projectless development is slightly overstated. Visual Studio still stores some information in a solution file (such as breakpoints and build settings), and it quietly stows that file away under a user-specific directory. However, there's a significant difference—these hidden solution files aren't required. Essential details (such as project references) are stored right in the web.config file. You'll learn about projectless development in the "Websites in Visual Studio" section of this chapter.

- **New compilation model**: Visual Studio is no longer responsible for compiling your code. Instead, ASP.NET takes on that responsibility exclusively. This gives Visual Studio more flexible debugging, and it simplifies deployment on different platforms (for example, 32-bit and 64-bit Windows). It also allows you to combine web pages written in C# with web pages written in another .NET language (such as Visual Basic) in the same project.
- **New code model**: The shift in the compilation model also reduces the differences between the code-behind model and the code-inline model of writing web pages, both of which Visual Studio now supports. However, the syntax for code-behind is subtly different from that used for Visual Studio 2003 web pages, and you'll need to perform a one-way conversion operation to edit your web application in Visual Studio 2005. You'll learn about the coding model in "The Coding Model" later in this chapter.
- **Integrated test web server**: If you've programmed with Web Matrix (a scaled-down design tool used with ASP.NET 1.*x*), you'll recognize the new integrated web server, which allows you to run your web pages without setting up virtual directories or deploying your website.

Along with these changes, a new edition of Visual Studio, called Visual Studio 2005 Team System, adds advanced collaboration and code versioning support (which is far beyond that available in simpler tools such as Visual SourceSafe). Although Visual Studio Team System isn't discussed in this chapter, you can learn more from `http://lab.msdn.microsoft.com/teamsystem` or *Pro Visual Studio 2005 Team System* (Apress, 2005).

Another interesting new tool is the freely downloadable ASP.NET Development Helper, which gives you the ability to see view state, tracing, and caching information in your web browser. You'll learn about the ASP.NET Development Helper in the later "ASP.NET Development Helper" section.

The .NET Development Model

To create an ASP.NET application, you need two high-level areas of functionality:

- The compiler, which inspects the developer code and translates it into lower-level code (in this case, IL)
- The IDE, which allows a developer to write code

The Compiler

.NET separates these two pieces. That way, every language can use the same design tools. The .NET language compilers include the following:

- The Visual Basic compiler (vbc.exe)
- The C# compiler (csc.exe)
- The JScript compiler (jsc.exe)
- The J# compiler (vjc.exe)

■ **Note** For a more comprehensive list that includes third-party languages, check out `http://www.dotnetpowered.com/languages.aspx`.

If you want to use these compilers manually, you can invoke them from the command line. You'll find all of them in c:\[WinDir]\Microsoft.NET\[Version], where WinDir is the directory of the operating system (like c:\Windows) and Version is the version number of .NET you've installed, like v2.0.50215. However, using the .NET compilers is awkward because you need to specify the files you want to compile and the other .NET assemblies they use. You also need to compile your entire application at once or compile each web page separately. To avoid these headaches, most developers rely on ASP.NET's built-in support for compiling pages.

■ **Note** In ASP.NET 1.*x*, Visual Studio used the precompiled code-behind model and was responsible for compiling all web pages into a single DLL assembly. In Visual Studio 2005, this behavior changes. Now, Visual Studio lets ASP.NET perform the compilation for each page the first time it's requested. This speeds up debugging and allows you to create websites that combine pages written in different languages. The original problems that motivated Visual Studio's precompilation model—optimizing the performance for the first request and reducing the need to deploy source code files—can now be solved using ASP.NET's precompilation features, which you'll learn about in Chapter 18.

The Visual Studio IDE

For those who are used to the previous version of the Visual Studio IDE, it's an obvious choice to use the new Visual Studio IDE. After all, it offers all the benefits of the previous version but with significant advancements in operability, syntax, and integration with other languages. For those who haven't tried Visual Studio before, the reasons to use Visual Studio may not be immediately obvious. Some of its advantages include the following:

WYSIWYG: Who writes HTML pages by hand? Using Visual Studio, you can tweak and fine-tune even static HTML content, applying fonts and styles.

Less code to write: Most applications require a fair bit of standard boilerplate code, and ASP.NET web pages are no exception. For example, when you add a new control to a web page, you also need to define a variable that allows you to manipulate that control in your code. With Visual Studio, these basic tasks are performed for you. Similar automation is provided for connecting to web services.

Intuitive coding style: By default, Visual Studio formats your code as you type, indenting automatically and using color-coding to distinguish elements such as comments. These minor differences make code much more readable and less prone to error. You can even configure what automatic formatting Visual Studio applies, which is great if you prefer different brace styles (such as K&R style, which always puts the opening brace on the same line as the preceding declaration).

■ **Tip** To see the formatting options, select Tools ➤ Options, make sure the Show All Settings check box is checked, and then find the Text Editor ➤ C# ➤ Formatting group of settings. You'll see a slew of options that control where curly braces should be placed.

An integrated web server: To host an ASP.NET web application, you need web server software like IIS, which waits for web requests and serves the appropriate pages. Setting up your web server isn't difficult, but it is inconvenient. Thanks to the integrated development web server in Visual Studio, you can run a website directly from the design environment. You also have the added security of knowing no external computer can run your test website.

Multilanguage development: Visual Studio allows you to code in your language or languages of choice using the same interface (IDE) at all times. Even better, Visual Studio 2005 adds the ability to put web pages coded in C# in the same project as web pages written in Visual Basic. The only limitation is that you can't use more than one language in the same web page (which would create obvious compilation problems).

Faster development times: Many of the features in Visual Studio are geared toward helping you get your work done faster. Convenience features such as powerful search-and-replace and automatic comment and uncomment features, which can temporarily hide a block of code, allow you to work quickly and efficiently.

Debugging: The Visual Studio debugging tools are the best way to track down mysterious errors and diagnose strange behavior. You can execute your code one line at a time, set intelligent breakpoints that you can save for later use, and view current in-memory information at any time.

Visual Studio also has a wealth of features that you won't see in this chapter, including project management, integrated source code control, and a rich extensibility model.

Websites in Visual Studio

When the IDE first loads, it shows an initial start page. You can use various user-specific options from this page and access online information such as recent MSDN articles. But to get right to work, choose File ➤ New Website to create a new ASP.NET application. Visual Studio will then show the New Web Site dialog box (see Figure 2-1). Notice that you *don't* use Visual Studio's File ➤ New Project command. That's because web applications aren't projects, as you'll see later in this chapter.

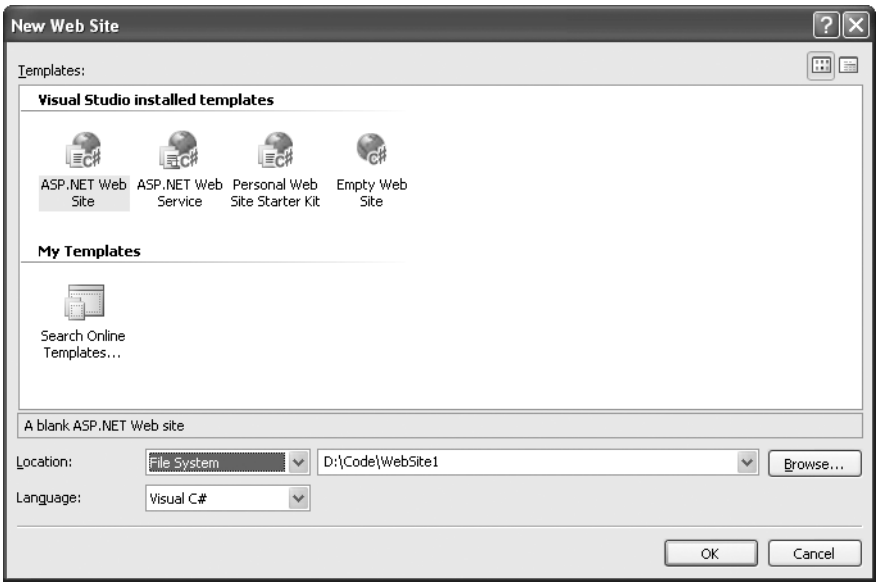

Figure 2-1. *The New Web Site window*

The New Web Site window allows you to specify three details:

Template: The template determines what files your website starts with. Visual Studio supports two types of basic ASP.NET applications: website applications and web service applications. These applications are actually compiled and executed in the same way. In fact, you can add web pages to a web service application and can add web services to an ordinary web application. The only difference is the files that Visual Studio creates by default. In a web application, you'll start with one sample web page in your project. In a web service application, you'll start with a sample web service. Additionally, Visual Studio includes more sophisticated templates for certain types of sites, and you can even create your own templates (or download third-party offerings).

Location: The location specifies where the website files will be stored. Typically, you'll choose File System and then use a folder on the local computer or a network path. However, you can also edit a website directly over HTTP or FTP (File Transfer Protocol). This is occasionally useful if you want to perform live website edits on a remote web server. However, it also introduces additional overhead. Of course, you should never edit a production web server directly because changes are automatic and irreversible. Instead, limit your changes to test servers.

Language: The language identifies the .NET programming language you'll use to code your website. The language you choose is simply the default language for the project. This means you can explicitly add Visual Basic web pages to a C# website, and vice versa (a feat that wasn't possible with earlier versions of Visual Studio).

Instead of typing the location in hand, you can click the Browse button, which shows the Choose Location dialog box. Along the left side of Choose Location dialog box you'll see four buttons that let you connect to different types of locations:

File System: This is the easiest choice—you simply need to browse through a tree of drives and directories or through the shares provided by other computers on the network. If you want to create a new directory for your application, just click the Create New Folder icon above the top-right corner of the directory tree. (You can also coax Visual Studio into creating a directory by adding a new directory name to the end of your path.)

Local IIS: This choice allows you to browse the virtual directories made available through the IIS web hosting software, assuming it's running on the current computer. Chapter 18 describes virtual directories in detail and shows you how to create them with IIS Manager. Impressively, you can also create them in Visual Studio using the Create New Web Application icon at the top-right corner of the virtual directory tree.

FTP Site: This option isn't quite as convenient as browsing for a directory—instead, you'll need to enter all the connection information, including the FTP site, the port, the directory, a user name, and a password before you can connect.

Remote Web Server: This option accesses a website at a specified URL (uniform resource locator) using HTTP. For this to work, the web server must have the FrontPage Extensions installed. When you connect, you'll be prompted for a user name and password.

Figure 2-2 shows all these location types.

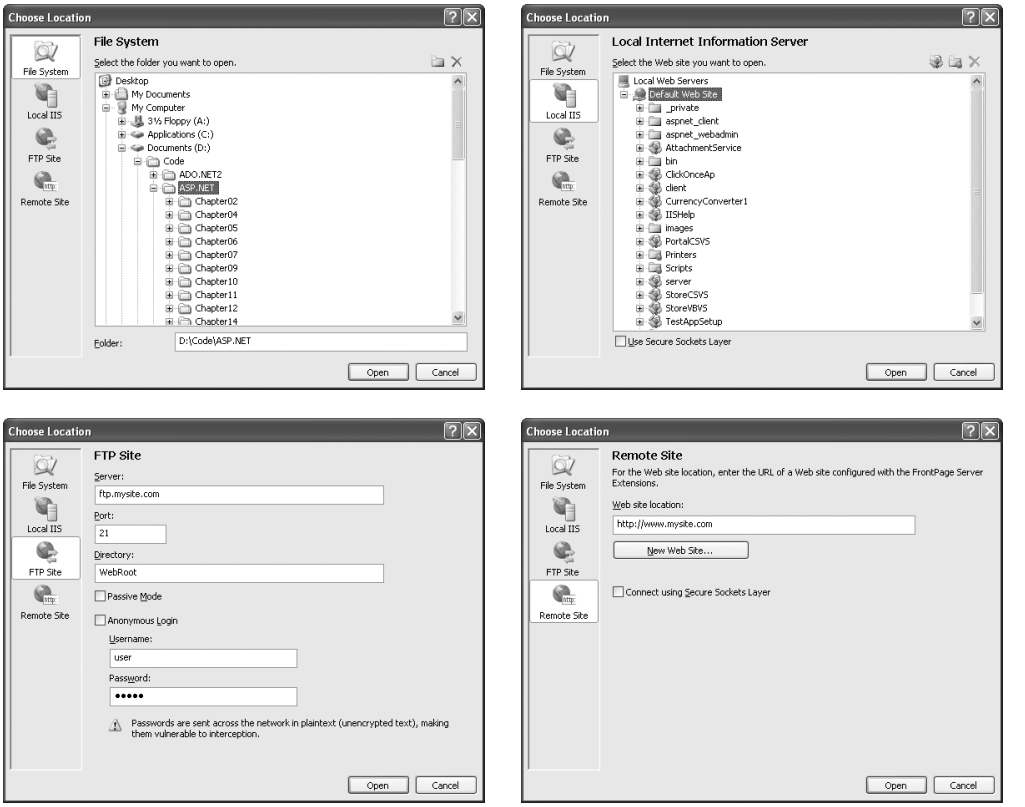

Figure 2-2. *Browsing to a website location*

Once you make your selection and click Open, Visual Studio returns you to the Create Web Site dialog box. Click OK, and Visual Studio will create the new web application. A new website starts with exactly one file—a default.aspx start page.

Projectless Development

In many ways, Visual Studio 2005 web applications are more remarkable for what they *don't* contain than what they do. Unlike previous versions of Visual Studio, Visual Studio 2005 web applications don't include extra files, such as .csproj project files and .sln solution files. Instead, every file in your web folder automatically is considered part of the web application. (This model makes sense, because every web page in a virtual directory is independently accessible, whether or not you consider it an official part of your project.)

Clearing out this clutter has several benefits:

- It's less work to deploy your website, because you don't need to specifically exclude these files. There's also less duplication of settings, because most of what Visual Studio needs (such as assembly references) is stored in the web.config configuration file.
- Team collaboration is greatly simplified, because different people can work independently on different pages without needing to lock the project files.

- It's easier to author websites with other tools, because no extra project files need to be maintained.
- Files can easily be transferred from one web application to another—all you need to do is copy the file.

Although this simplifies life dramatically, under the radar there are still the last vestiges of Visual Studio's solution-based system.

When you create a web application, Visual Studio actually creates solution files (.sln and .suo) in a user-specific directory, like c:\Documents and Settings\[UserName]\Visual Studio 2005\Projects\[ProjectName]. This file provides a few Visual Studio–specific features that aren't directly related to ASP.NET, such as debugging settings. For example, if you add a breakpoint to the code in a web page (as discussed in the "Visual Studio Debugging" section later in this chapter), Visual Studio stores the breakpoint in the .suo file so it's still there when you open the project later. Similarly, Visual Studio tracks the currently open files so it can restore your view when you return to the project. This approach to solution management is fragile—obviously, if you move the project from one location to another, you lose all this information. However, because this information isn't really all that important (think of it as a few project-specific preferences), losing it isn't a serious problem. The overall benefits of a projectless system are worth the trade-off.

Migrating a Visual Studio .NET Project

If you have an existing web application created with Visual Studio .NET 2002 or 2003, you can open the project or solution file using the File ➤ Open Project command. When you do, Visual Studio begins the Conversion Wizard.

The Conversion Wizard is exceedingly simple. It prompts you to choose whether to create a backup and, if so, where it should be placed (see Figure 2-3). If this is your only copy of the application, a backup is a good idea in case some aspects of your application can't be converted successfully. Otherwise, you can skip this option.

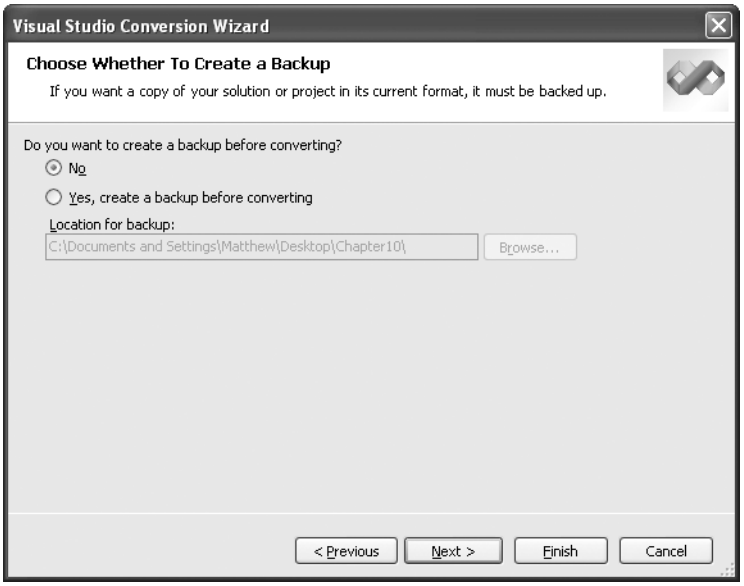

Figure 2-3. *Importing a Visual Studio .NET 2003 project*

When you click Finish, Visual Studio performs an in-place conversion. The conversion tool is fairly aggressive, and it attempts to convert every web page to use Visual Studio's new code-behind model. Any errors and warnings are added to a conversion log, which you can display when the conversion is complete. In a typical website, the conversion operation runs without any errors but generates a long list of warnings. These inform you when Visual Studio removes precompiled files, changes pages to use automatic event wire-up, and modifies the accessibility of event handlers (switching them from private to protected). All of these changes are minor modifications designed to apply the new coding model, which is described in the section "The Coding Model" later in this chapter. Figure 2-4 shows a sample log.

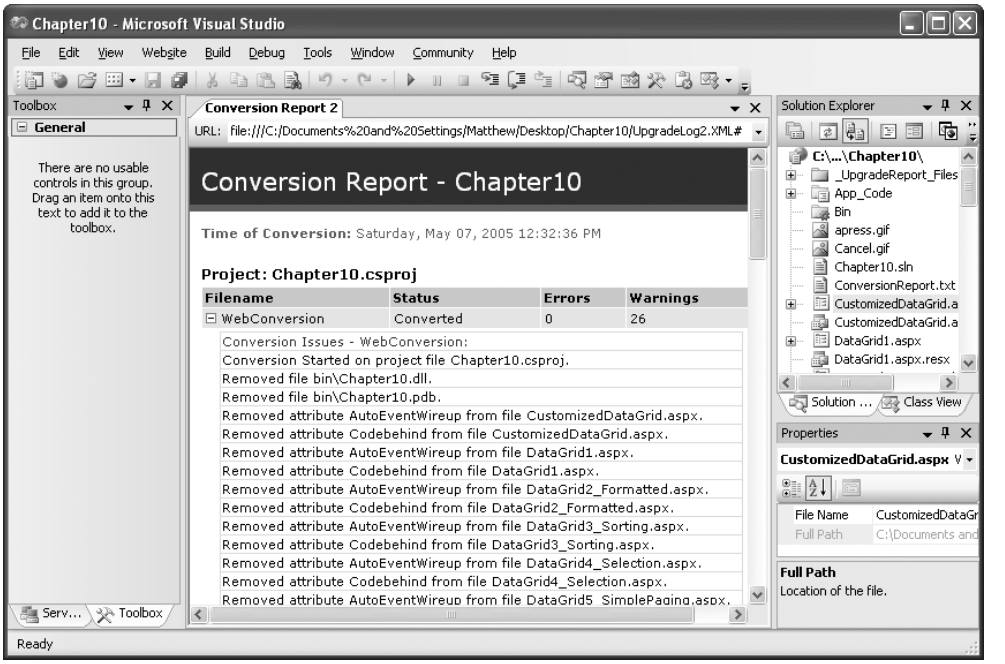

Figure 2-4. *A conversion log with typical warnings*

Visual Studio 2005 doesn't support adding old web pages to a new web application using the Website ➤ Add Existing Item. If you take this step and try to run your web application, you'll receive an error informing you that the Visual Studio .NET 2003 version of the code-behind model is no longer supported. Instead, Visual Studio will recommend you use the Open Project feature to start the Conversion Wizard.

Designing a Web Page

To start designing a web page, double-click the web page in the Solution Explorer (start with default.aspx if you haven't added any pages). A blank page will appear in the designer.

To add controls, choose the control type from the Toolbox on the left. (The controls in the Toolbox are grouped in numerous categories based on their functions, but you'll find basic ingredients in the Standard tab.) Once you've added a control, you can resize it and configure its properties

in the Properties window. Every time you add a web control, Visual Studio automatically adds the corresponding tag to your .aspx web-page file. You can switch your view to look at the tags by clicking the Source button at the bottom of the web designer window. Click Design to revert to the graphical web form designer.

Figure 2-5 shows two views of the same web page that contain a label and a button. One view is in HTML mode, and the other is in design mode.

Figure 2-5. *The two modes for editing web pages*

Using the HTML view, you can manually add attributes or rearrange controls. In fact, Visual Studio even provides limited IntelliSense features that automatically complete opening tags and alert you if you use an invalid tag. Generally, you won't need to use the HTML view in Visual Studio. Instead, you can use the design view and configure controls through the Properties window.

■ **Note** Unlike previous versions, Visual Studio 2005 doesn't tamper with your HTML markup. Instead, it always preserves the indenting you use. You can fine-tune this behavior using the Text Editor ➤ HTML group of settings in the Tools ➤ Options dialog box. For example, one handy option that isn't turned on by default is Format HTML on Paste, which indents arbitrary blocks of markup when you paste them into a page.

To configure a control, click once to select it, or choose it by name in the drop-down list at the top of the Properties window. Then, modify the appropriate properties in the window, such as Text, ID, and ForeColor. These settings are automatically translated to the corresponding ASP.NET control tag attributes and define the initial appearance of your control. Visual Studio even provides special "choosers" (technically known as UITypeEditors) that allow you to select extended properties. For example, you can select a color from a drop-down list that shows you the color, and you can configure the font from a standard font selection dialog box.

To position a control on the page, you need to use all the usual tricks of HTML, such as paragraphs, line breaks, and tables. Unlike previous versions, Visual Studio 2005 doesn't support a grid-layout mode for absolute positioning with CSS (Cascading Style Sheets). Instead, it encourages you to use the more flexible flow-layout mode, where content can grow and shrink dynamically without creating a problem. However, there is a way to get back to the grid-layout behavior. All you need to do is add an inline CSS style for your control that specifies absolute positioning. (This style will already exist in any pages you've created with a previous version of Visual Studio .NET in grid-layout mode.) Here's an example:

```
<asp:Button id="cmd" style="POSITION: absolute; left: 100px; top: 50px;"
 runat="server"  ... />
```

Once you've made this change, you're free to drag the button around the window at will. Of course, you shouldn't go this route just because it seems closer to the Windows GUI (graphical user interface) model. Few great web pages rely on absolute positioning, because it's just too awkward and inflexible.

Smart Tags

Another timesaving feature that's new in Visual Studio 2005 is the *smart tag*; smart tags make it easier to configure complex controls. Smart tags aren't offered for all controls, but they are used for rich controls such as the GridView, TreeView, and Calendar.

You'll know a smart tag is available if, when you select a control, you see a small arrow in the top-right corner. If you click this arrow, a window will appear with links that trigger other, higher-level tasks. For example, Figure 2-6 shows how you can use this technique to access Calendar autoformatting. (Smart tags can include many more features, but the Calendar smart tag provides only a single link.)

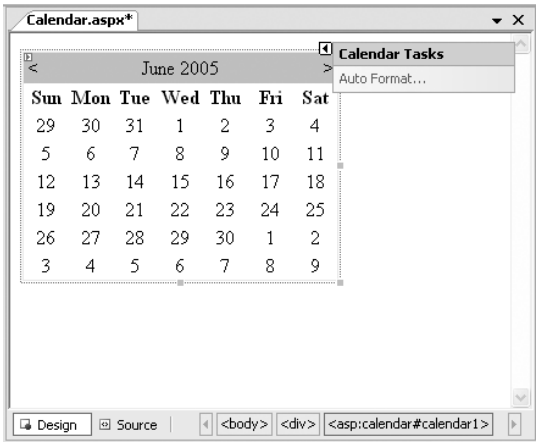

Figure 2-6. *A smart tag for the Calendar control*

Static HTML Tags

Along with full-fledged web controls, you can also add ordinary HTML tags. You simply drag these from the HTML tab of the Toolbox.

For example, you might want to create a simple tag to group some web controls with a border. Visual Studio provides a valuable style builder for formatting any static HTML element with CSS style properties. To test it, add the Div from the HTML section of the Toolbox, which appears on your page as a panel. Then right-click the panel, and choose Style. The Style Builder dialog box (shown in Figure 2-7) will appear, with options for configuring the colors, font, layout, and border for the element. As you configure these properties, the web page's HTML will be updated to reflect your settings.

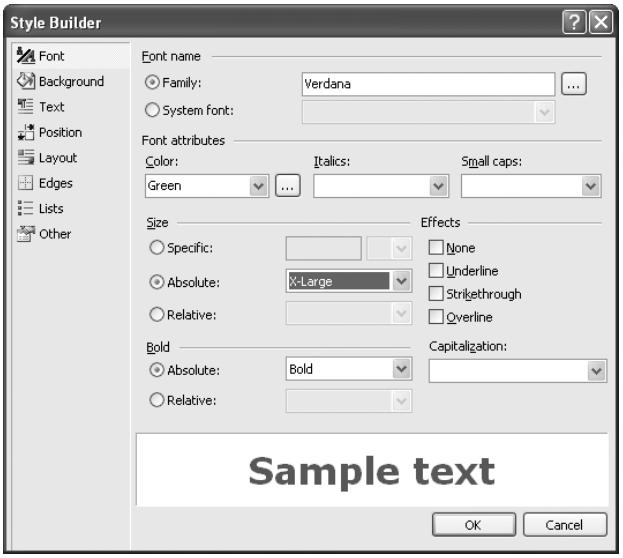

Figure 2-7. *Building HTML styles*

If you want to configure the HTML element as a server control so that you can handle events and interact with it in code, you need to right-click it in the web page and select Run As Server Control. This adds the required runat="server" attribute to the control tag. Alternatively, you could switch to design view and type this in on your own.

HTML Tables

One convenient way to organize content in a web page is to place it in the different cells of an HTML table using the <table> tag. In previous versions of Visual Studio, the design-time support for this strategy was poor. But in Visual Studio 2005, life gets easier. To try it, drag a table from the HTML tab of the Toolbox. You'll start with a standard 3×3 table, but you can quickly transform it using editing features that more closely resemble a word processor than a programming tool. Here are some of the tricks you'll want to use:

- To move from one cell to another in the table, press the Tab key or use the arrow keys. The current cell is highlighted with a blue border. Inside each cell you can type static HTML or drag and drop controls from the Toolbox.
- To add new rows and columns, right-click inside a cell, and choose from one of the many options in the Insert submenu to insert rows, columns, and individual cells.
- To resize a part of the table, just click and drag.
- To format a cell, right-click inside it, and choose Style. This shows the same Style Builder dialog box you saw in Figure 2-7.
- To work with several cells at once, hold down Ctrl while you click each cell. You can then right-click to perform a batch formatting operation.
- To merge cells (in other words, change two cells into one cell that spans two columns), just select the cells, right-click, and choose Merge.

With these conveniences, you might never need to resort to a design tool like Dreamweaver.

The Visual Studio IDE

Now that you've created a basic website, it's a good time to take a tour of the different parts of the Visual Studio interface. Figure 2-8 identifies each part of the Visual Studio window, and Table 2-1 describes each one.

Table 2-1. *Visual Studio Windows*

Windows	Description
Solution Explorer	Lists the files and subfolders that are in the web application folder.
Toolbox	Shows ASP.NET's built-in server controls and any third-party controls or custom controls that you build yourself and add to the Toolbox. Controls can be written in any language and used in any language.
Server Explorer	Allows access to databases, system services, message queues, and other server-side resources.
Properties	Allows you to configure the currently selected element, whether it's a file in the Solution Explorer or a control on the design surface of a web form.

Windows	Description
Error List	Reports on errors that Visual Studio has detected in your code but that you haven't resolved yet.
Task List	Lists comments that start with a predefined moniker so that you can keep track of portions of code that you want to change and also jump to the appropriate position quickly.
Document	Allows you to design a web page by dragging and dropping and to edit the code files you have within your Solution Explorer. Also supports non-ASP.NET file types, such as static HTML and XML files.
Macro Explorer	Allows you to see all the macros you've created and execute them. Macros are an advanced Visual Studio feature; they allow you to automate time-consuming tasks. Visual Studio exposes a rich extensibility model, and you can write a macro using pure .NET code.
Class View	Shows a different view of your application that is organized to show all the classes you've created (and their methods, properties, and events).

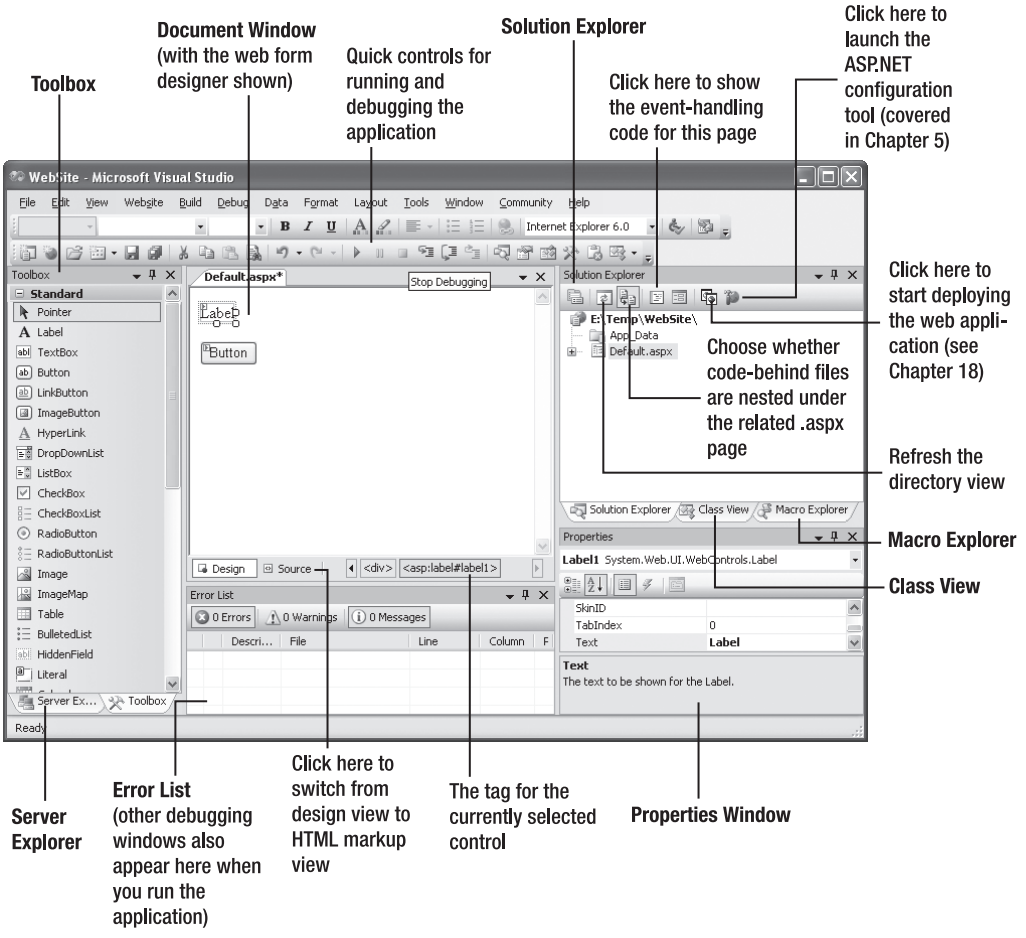

Figure 2-8. *The Visual Studio interface*

Tip The Visual Studio interface is highly configurable. You can drag the various windows and dock them to the sides of the main Visual Studio window. Also, some windows on the side automatically slide into and out of view as you move your mouse. If you want to freeze these windows in place, just click the thumbtack icon in the top-right corner of the window.

Solution Explorer

The Solution Explorer is, at its most basic, a visual filing system. It allows you to see the files that are in the web application directory.

Table 2-2 lists some of the file types you're likely to see in an ASP.NET web application.

Table 2-2. *ASP.NET File Types*

File	Description
Ends with .aspx	These are ASP.NET web pages (the .NET equivalent of the .asp file in an ASP application). They contain the user interface and, optionally, the underlying application code. Users request or navigate directly to one of these pages to start your web application.
Ends with .ascx	These are ASP.NET user controls. User controls are similar to web pages, except that they can't be accessed directly. Instead, they must be hosted inside an ASP.NET web page. User controls allow you to develop an important piece of the user interface and reuse it in as many web forms as you want without repetitive code.
Ends with .asmx	These are ASP.NET web services. Web services work differently than web pages, but they still share the same application resources, configuration settings, and memory.
web.config	This is the XML-based configuration file for your ASP.NET application. It includes settings for customizing security, state management, memory management, and much more. Visual Studio adds a web.config file when you need it. (For example, it adds a web.config file that supports debugging if you attempt to run your web application.) When you first create a website, you won't have a web.config file.
global.asax	This is the global application file. You can use this file to define global variables and react to global events, such as when a web application first starts (see Chapter 5 for a detailed discussion). Visual Studio doesn't create a global.asax file by default—you need to add it if it's appropriate.
Ends with .cs	These are code-behind files that contain C# code. They allow you to separate the application from the user interface of a web page. The code-behind model is introduced in this chapter and used extensively in this book.

In addition, your web application can contain other resources that aren't ASP.NET file types. For example, your virtual directory can hold image files, HTML files, or CSS files. These resources might be used in one of your ASP.NET web pages, or they can be used independently.

Visual Studio distinguishes between different file types. When you right-click a file in the list, a context menu appears with the menu options that apply for that file type. For example, if you right-click a web page, you'll have the option of building it and launching it in a browser window.

Using the Solution Explorer, you can rename, rearrange, and add files. All these options are just a right-click away. To delete a file, just select it in the Solution Explorer, and press the Delete key.

You can also add new files by right-clicking the Solution Explorer and selecting Add ➤ Add New Item. You can add various different types of files, including web forms, web services, stand-alone

classes, and so on. You can also copy files that already exist elsewhere on your computer (or an accessible network path) by selecting Add ➤ Add Existing Item. Use the Add ➤ New Folder to create a new subdirectory inside your web application. You can then drag web pages and other files into or out of this directory.

Visual Studio also checks for project management events such as when another process changes a file in a project you currently have open. When this occurs, Visual Studio will notify you and give you the option to refresh the file.

Document Window

The document window is the portion of Visual Studio that allows you to edit various types of files using different designers. Each file type has a default editor. To learn a file's default editor, simply right-click that file in the Solution Explorer, and then select Open With from the pop-up menu. The default editor will have the word *Default* alongside it.

Depending on the applications you've installed, you may see additional designers that plug into Visual Studio. For example, if you've installed FrontPage 2003, you'll have the option of editing web pages with a FrontPage designer (which actually opens your web page in a stand-alone FrontPage window).

Toolbox

The Toolbox works in conjunction with the document window. Its primary use is providing the controls that you can drag onto the design surface of a web form. However, it also allows you to store code and HTML snippets.

The content of the Toolbox depends on the current designer you're using as well as the project type. For example, when designing a web page, you'll see the set of tabs described in Table 2-3. Each tab contains a group of buttons. You can see only one tab at a time. To view a tab, click the heading, and the buttons will slide into view.

Table 2-3. *Toolbox Tabs for an ASP.NET Project*

Tab	Description
Standard	This tab includes the rich web server controls that are the heart of ASP.NET's web form model.
Data	These components allow you to connect to a database. This tab includes nonvisual data source controls that you can drop onto a form and configure at design time (without using any code) and data display controls such as grids.
Validation	These controls allow you to verify an associated input control against user-defined rules. For example, you can specify the input can't be empty, it must be a number, it must be greater than a certain value, and so on. Chapter 4 has more details.
Navigation	These controls are designed to display site maps and allow the user to navigate from one page to another. You'll learn about the navigation controls in Chapter 16.
Login	These controls provide prebuilt security solutions, such as login boxes and a wizard for creating users. You'll learn about the login controls in Chapter 20.
WebParts	This set of controls supports web parts, an ASP.NET model for building componentized, highly configurable web portals. You'll learn about web parts in Chapter 29.
HTML	This tab allows you to drag and drop static HTML elements. If you want, you can also use this tab to create server-side HTML controls—just drop a static HTML element onto a page, right-click it, and choose Run As Server Control.
General	Provides a repository for code snippets and control objects. Just drag and drop them here, and pull them off when you need to use them later.

You can customize both the tabs and the items in each tab. To modify the tab groups, right-click a tab heading, and select Rename Tab, Add Tab, or Delete Tab. To add an item, right-click the blank space on the Toolbox, and Choose Items. You can also drag items from one tab group to another.

Error List and Task List

The Error List and Task List are two versions of the same window. The Error List catalogs error information that's generated by Visual Studio when it detects problematic code. The Task List shows a similar view with to-do tasks and other code annotations you're tracking. Each entry in the Error List and Task List consists of a text description and, optionally, a link that leads you to a specific line of code somewhere in your project.

With the default Visual Studio settings, the Error List appears automatically whenever you build a project that has errors (see Figure 2-9).

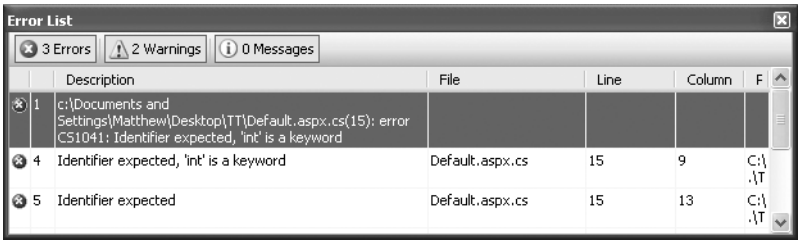

Figure 2-9. *Viewing build errors in a project*

To see the Task List, choose View ➤ Other Windows ➤ Task List. Two types of tasks exist—user tasks and comments. You can choose which you want to see from the drop-down list at the top of the Task List. User tasks are entries you've specifically added to the Task List. You create these by clicking the Create User Task icon (which looks like a clipboard with a check mark) in the Task List. You can give your task a basic description, a priority, and a check mark to indicate when it's complete.

■ **Note** As with breakpoints, any custom tasks you add by hand are stored in the hidden solution files. This makes them fairly fragile—if you rename or move your project, these tasks will disappear without warning (or without even a notification the next time you open the website).

The comment entries are more interesting, because they're added automatically and they link to a specific line in your code. To try the comment feature, move somewhere in your code, and enter the comment marker (`//`) followed by the word *TODO* (which is commonly referred to as a *token tag*). Now type in some descriptive text:

```
// TODO: Replace this hard-coded value with a configuration file setting.
string fileName = @"c:\myfile.txt"
```

Because your comment uses the recognized token tag TODO, Visual Studio recognizes it and automatically adds it to the Task List (as shown in Figure 2-10).

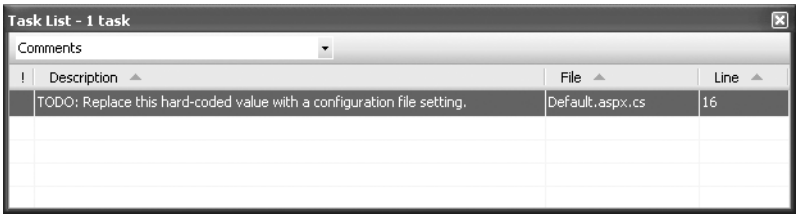

Figure 2-10. *Keeping track of tasks*

To move to the line of code, double-click the new task entry. Notice that if you remove the comment, the task entry is automatically removed as well.

Three token tags are built-in—HACK, TODO, and UNDONE. However, you can add more. Simply select Tools ➤ Options. In the Options dialog box, navigate to the Environment ➤ Task List tab. You'll see a list of comment tokens, which you can modify, remove, and add to. Figure 2-11 shows this window with a new ASP comment token that you could use to keep track of sections of code that have been migrated from classic ASP pages.

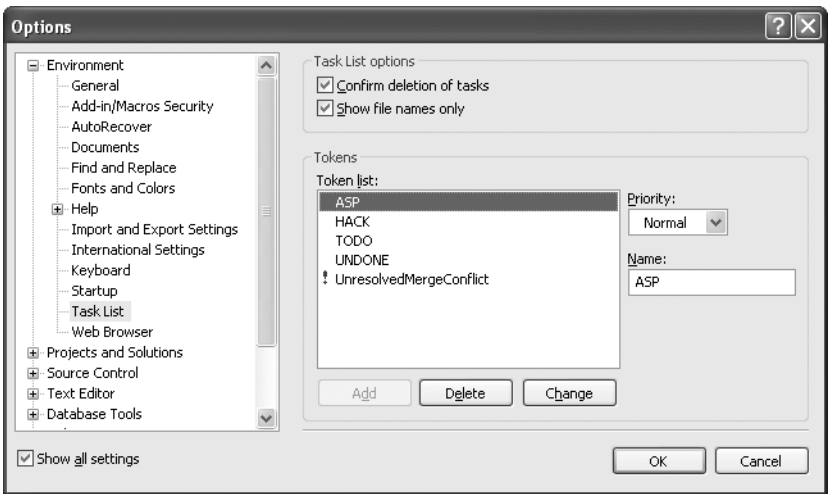

Figure 2-11. *Adding a new comment token*

Server Explorer

The Server Explorer provides a tree that allows you to explore various types of services on the current computer (and other servers on the network). It's similar to the Computer Management administrative tool. Typically, you'll use the Server Explorer to learn about available event logs, message queues, performance counters, system services, and SQL Server databases on your computer.

The Server Explorer is particularly noteworthy because it doesn't just provide a way for you to browse server resources; it also allows you to interact with them. For example, you can create databases, execute queries, and write stored procedures using the Server Explorer in much the same way that you would using the Enterprise Manager administrative utility that's included with SQL Server. To find out what you can do with a given item, right-click it. Figure 2-12 shows the Server Explorer window listing the databases in a local SQL Server and allowing you to retrieve all the records in the selected table.

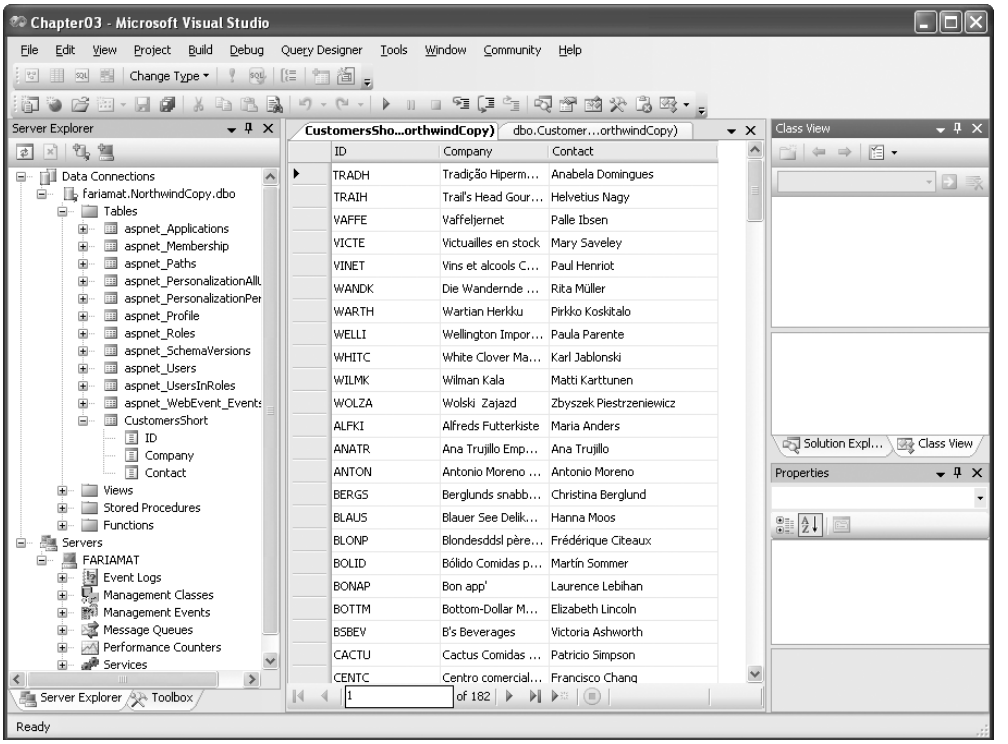

Figure 2-12. *Querying data in a database table*

The Code Editor

Many of Visual Studio's most welcome enhancements appear when you start to write the code that supports your user interface. To start coding, you need to switch to the code-behind view. To switch back and forth, you can use two buttons that are placed just above the Solution Explorer window. The tooltips identify these buttons as View Code and View Designer. When you switch to code view, you'll see the page class for your web page. You'll learn more about code-behind later in this chapter.

ASP.NET is event-driven, and everything in your web-page code takes place in response to an event. To create a simple event handler for the Button.Click event, double-click the button in design view. Here's a simple example that displays the current date and time in a label:

```
protected void Button1_Click(object sender, EventArgs e)
{
    Label1.Text = "Current time: " + DateTime.Now.ToLongTimeString();
}
```

To test this page, select Debug ➤ Start Debugging from the menu. Because this is the first time running any page in this application, Visual Studio will inform you that you need a configuration file that specifically enables debugging (see Figure 2-13).

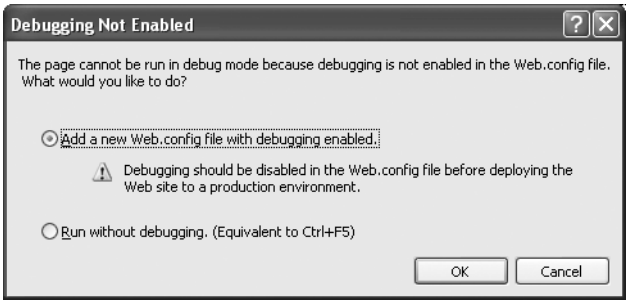

Figure 2-13. *Adding a web.config file automatically*

Click OK to add this configuration file. Then, Visual Studio will launch your default browser, with the URL set to your page. At this point, your request will be passed to ASP.NET, which will compile the page and execute it.

To test your event-handling logic, click the button on the page. The page will then be submitted to ASP.NET, which will run your event-handling code and return a new HTML page with the data (as shown in Figure 2-14).

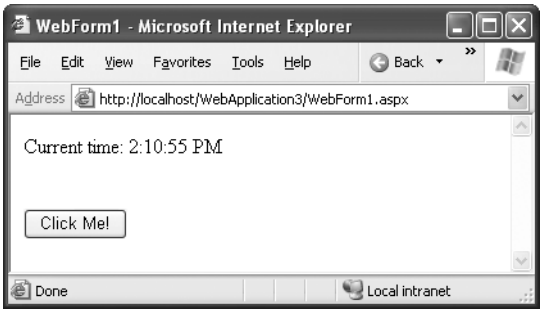

Figure 2-14. *Testing a simple web page*

Adding Assembly References

By default, ASP.NET makes a small set of commonly used .NET assemblies available to all web pages. These assemblies (listed in Table 2-4) are configured through a special machine-wide configuration file. You don't need to take any extra steps to use the classes in these assemblies.

Table 2-4. *Core Assemblies for ASP.NET Pages*

Assembly	Description
mscorlib.dll and System.dll	Includes the core set of .NET data types, common exception types, and numerous other fundamental building blocks.
System.Configuration.dll	Includes classes for reading and writing configuration information in the web.config file, including your custom settings.
System.Data.dll	Includes the data container classes for ADO.NET, along with the SQL Server data provider.

Continued

Table 2-4. *Continued*

Assembly	Description
System.Drawing.dll	Includes classes representing colors, fonts, and shapes. Also includes the GDI+ drawing logic you need to build graphics on the fly.
System.Web.dll	Includes the core ASP.NET classes, including classes for building web forms, managing state, handling security, and much more.
System.Web.Services.dll	Includes classes for building web services—units of code that can be remotely invoked over HTTP.
System.Xml.dll	Includes .NET classes for reading, writing, searching, transforming, and validating XML.
System.EnterpriseServices.dll	Includes .NET classes for COM+ services such as transactions.
System.Web.Mobile.dll	Includes .NET classes for the mobile web controls, which are targeted for small devices such as web-enabled cell phones.

If you want to use additional features or a third-party component, you may need to import more assemblies. For example, if you want to use an Oracle database, you need to add a reference to the System.Data.OracleClient.dll assembly. To add a reference, select Website ➤ Add Reference. The Add Reference dialog box will appear, with a list of registered .NET assemblies (see Figure 2-15).

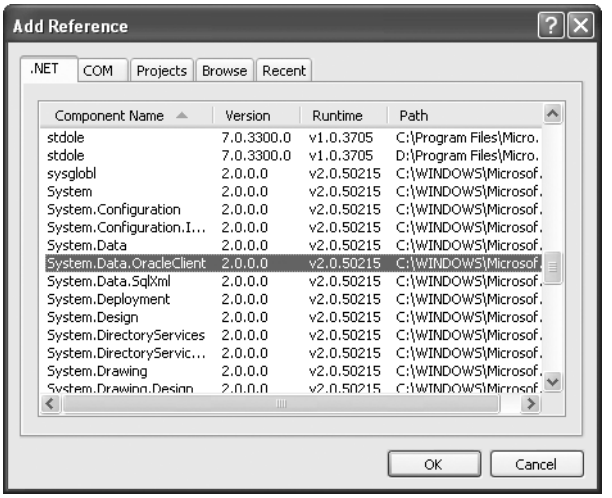

Figure 2-15. *Adding a reference*

In the Add Reference dialog box, select the component you want to use. If you want to use a component that isn't listed here, you'll need to click the Browse tab and select the DLL file from the appropriate directory.

When you add a reference, Visual Studio modifies the web.config file to indicate that you use this assembly. Here's an example of what you might see after you add a reference to the System.EnterpriseServices.dll file:

```
<?xml version="1.0"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <compilation debug="true">
      <assemblies>
        <add assembly="System.Data.OracleClient, Version=2.0.0.0, ..."/>
      </assemblies>
    </compilation>
    <!-- Other settings omitted. -->
  </system.web>
</configuration>
```

Chapter 5 explores the web.config file in greater detail.

If you add a reference to an assembly that isn't stored in the GAC (global assembly cache), Visual Studio will create a Bin subdirectory in your web application and copy the DLL into that directory. This step isn't required for assemblies in the GAC because they are shared with all the .NET applications on the computer.

■ **Note** Unlike previous versions of Visual Studio, in Visual Studio 2005 you won't see a list of assembly references in the Solution Explorer. Instead, you need to crack open the web.config file to get that information.

If you look at the code for a web-page class, you'll notice that Visual Studio imports a lengthy number of core .NET namespaces. Here's the code you'll see:

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

Adding a reference isn't the same as importing the namespace with the using statement. The using statement allows you to use the classes in a namespace without typing the long, fully qualified class names. However, if you're missing a reference, it doesn't matter what using statements you include—the classes won't be available. For example, if you import the System.Web.UI namespace, you can write Page instead of System.Web.UI.Page in your code. But if you haven't added a reference to the System.Web.dll assembly that contains these classes, you still won't be able to access the classes in the System.Web.UI namespace.

IntelliSense and Outlining

As you program with Visual Studio, you'll become familiar with its many timesaving conveniences. The following sections outline the most important features you'll use (none of which is new in Visual Studio 2005).

■ **Tip** Visual Studio does include one new IntelliSense feature—XHTML (Extensible HTML) validation. You'll learn about this feature, and how to control the level of XHTML support you want, in Chapter 3.

Outlining

Outlining allows Visual Studio to "collapse" a subroutine, block structure, or region to a single line. It allows you to see the code that interests you, while hiding unimportant code. To collapse a portion of code, click the minus box next to the first line. Click the box again (which will now have a plus symbol) to expand it (see Figure 2-16).

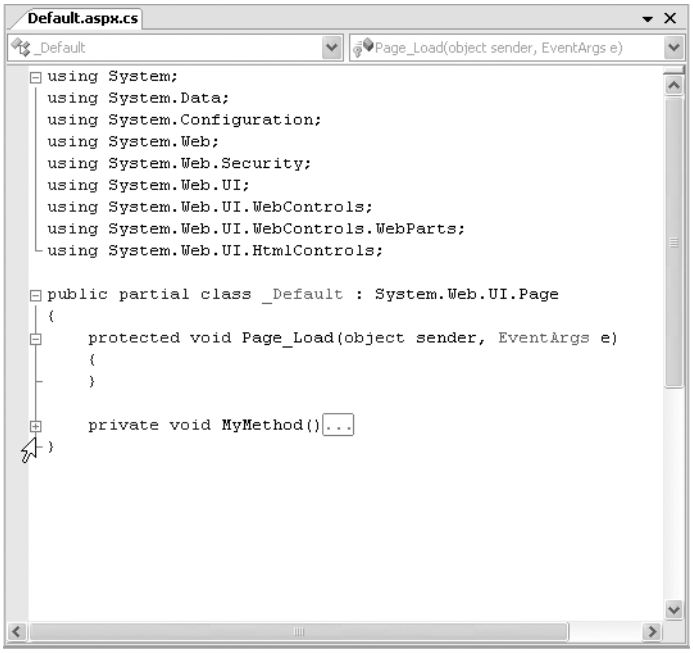

Figure 2-16. *Collapsing code*

You can hide any section of code that you want. Simply select the code, right-click the selection, and choose Outlining ➤ Hide Selection.

Member List

Visual Studio makes it easy for you to interact with controls and classes. When you type a class or object name, Visual Studio pops up a list of available properties and methods (see Figure 2-17). It uses a similar trick to provide a list of data types when you define a variable and to provide a list of valid values when you assign a value to an enumeration.

Visual Studio also provides a list of parameters and their data types when you call a method or invoke a constructor. This information is presented in a tooltip above the code and is shown as you type. Because the .NET class library heavily uses function overloading, these methods may have multiple different versions. When they do, Visual Studio indicates the number of versions and allows you to see the method definitions for each one by clicking the small up and down arrows in the tooltip. Each time you click the arrow, the tooltip displays a different version of the overloaded method (see Figure 2-18).

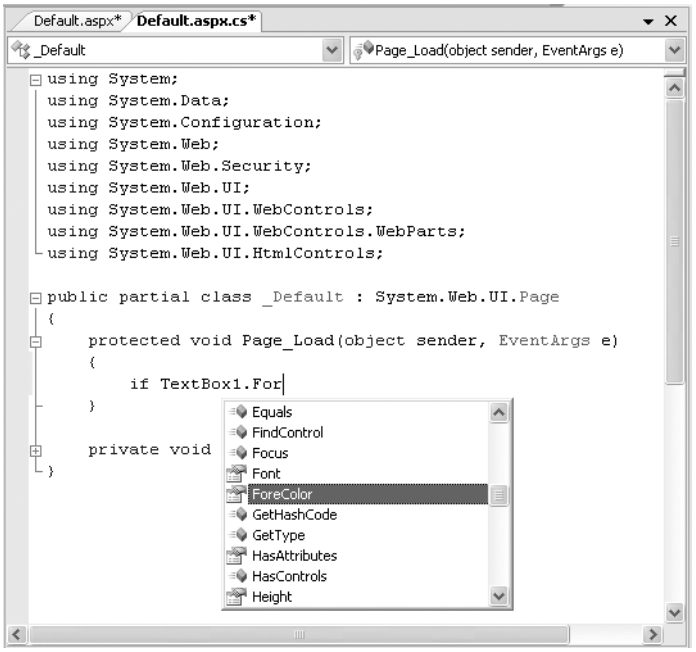

Figure 2-17. *IntelliSense at work*

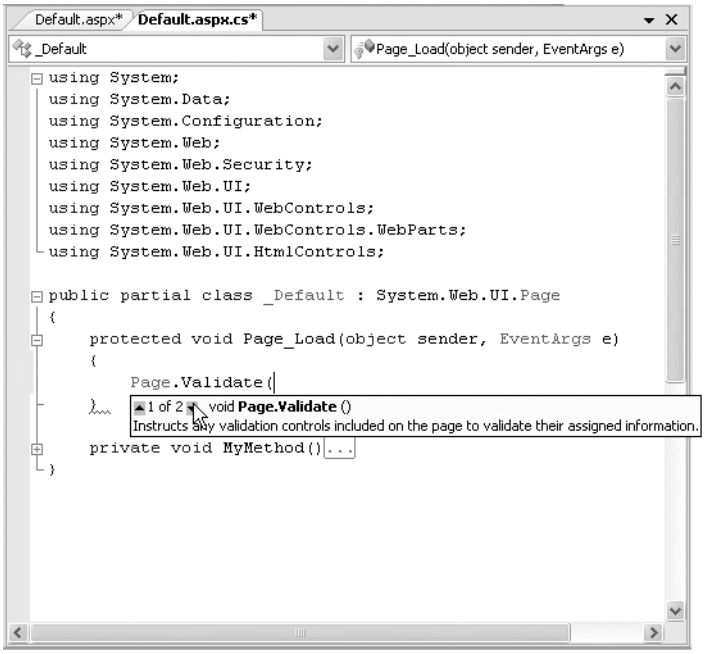

Figure 2-18. *IntelliSense with overloaded methods*

Error Underlining

One of the code editor's most useful features is error underlining. Visual Studio is able to detect a variety of error conditions, such as undefined variables, properties, or methods; invalid data type conversions; and missing code elements. Rather than stopping you to alert you that a problem exists, the Visual Studio editor quietly underlines the offending code. You can hover your mouse over an underlined error to see a brief tooltip description of the problem (see Figure 2-19).

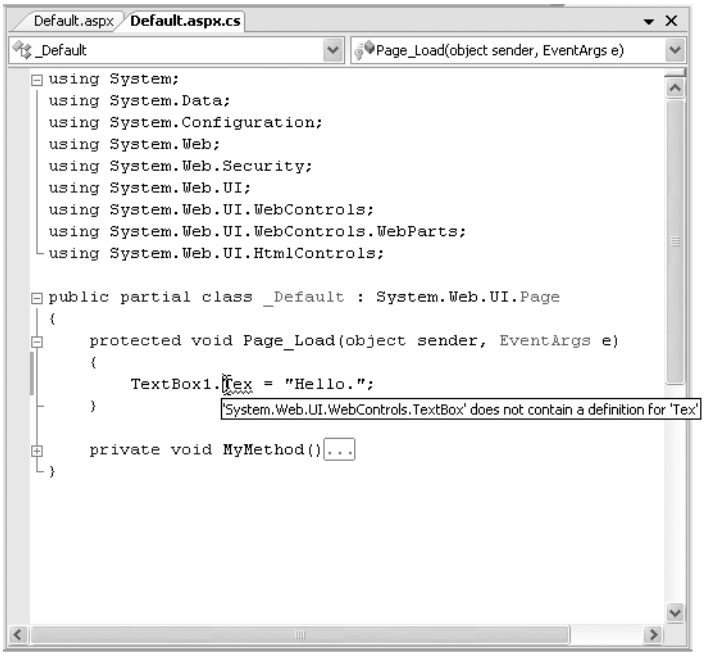

Figure 2-19. *Highlighting errors at design time*

Visual Studio won't flag your errors immediately. Instead, it will quickly scan through your code as soon as you try to compile it and mark all the errors it finds. If your code contains at least one error, Visual Studio will ask you whether it should continue. At this point, you'll almost always decide to cancel the operation and fix the problems Visual Studio has reported. (If you choose to continue, you'll actually wind up using the last compiled version of your application, because the .NET compilers can't build an application that has errors.)

■ **Note** You may find that as you fix errors and rebuild your project you discover more problems. That's because Visual Studio doesn't check for all types of errors at once. When you try to compile your application, Visual Studio scans for basic problems such as unrecognized class names. If these problems exist, they can easily mask other errors. On the other hand, if your code passes this basic level of inspection, Visual Studio checks for more subtle problems such as trying to use an unassigned variable.

The Coding Model

So far, you've learned how to design simple web pages, and you've taken a tour of the Visual Studio interface. But before you get to serious coding, it's important to understand a little more about the underpinnings of the ASP.NET code model. In this section, you'll learn about your options for using code to program a web page and how ASP.NET events wire up to your code.

Visual Studio supports two models for coding web pages and web services:

Inline code: This model is the closest to traditional ASP. All the code and HTML is stored in a single .aspx file. The code is embedded in one or more script blocks. However, even though the code is in a script block, it doesn't lose IntelliSense or debugging support, and it doesn't need to be executed linearly (like classic ASP code). Instead, you'll still react to control events and use subroutines. This model is handy because it keeps everything in one neat package, and it's popular for coding simple web pages.

Code-behind: This model separates each ASP.NET web page into two files: an .aspx markup file with the HTML and control tags, and a .cs code file with the source code for the page. This model provides better organization, and separating the user interface from programmatic logic is keenly important when building complex pages. In Visual Studio 2005, the implementation of the code-behind model has changed, but the overall philosophy is the same.

In .NET 1.0 and 1.1, the design tool you choose determines the model you use. With Visual Studio, you have the freedom to use either approach. When you add a new web page to your website (using Website ➤ Add New Item), the Place Code in a Separate File check box chooses whether you want to use the code-behind model (see Figure 2-20). Visual Studio remembers your previous setting for the next time you add a new page, but it's completely valid (albeit potentially confusing) to mix both styles of pages in the same application.

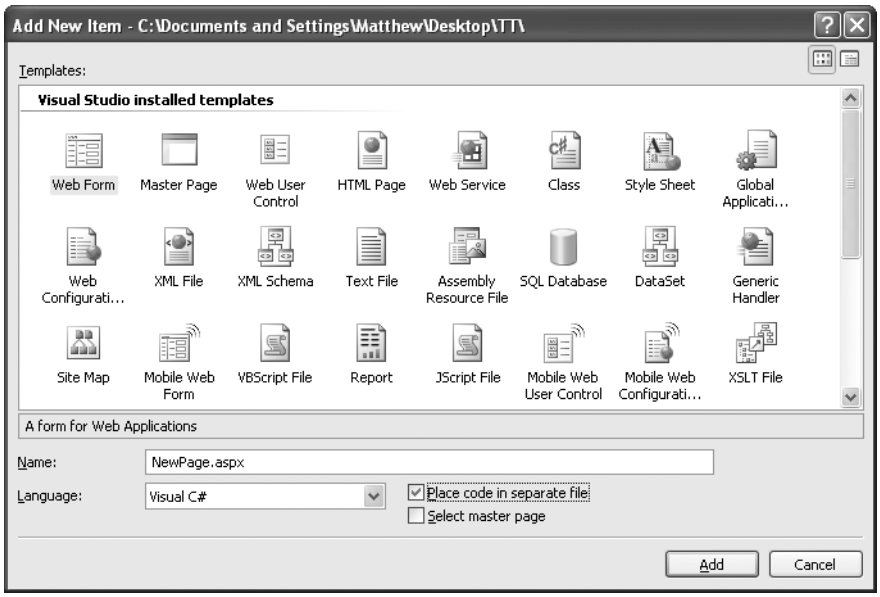

Figure 2-20. *Choosing the coding model*

To understand the difference, it helps to consider a simple page, like the following dynamic time example; this is TestFormInline.aspx, which shows how the page looks with inline code:

```
<%@ Page Language="C#" %>
<script runat="server">
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Current time: " + DateTime.Now.ToLongTimeString();
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Test Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="Click Me!"></asp:Label><br />
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
         Text="Button" /></div>
    </form>
</body>
</html>
```

The following listings, TestFormCodeBehind.aspx and TestFormCodeBehind.aspx.cs, show how the page is broken up into two pieces using the code-behind model. This is TestFormCodeBehind.aspx:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="TestFormCodeBehind.aspx.cs"
    Inherits="TestFormCodeBehind"%>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Test Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="Click Me!"></asp:Label><br />
        <br />
        <br />
        <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
         Text="Button" /></div>
    </form>
</body>
</html>
```

This is TestFormCodeBehind.aspx.cs:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
```

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

public partial class TestFormCodeBehind : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Current time: " + DateTime.Now.ToLongTimeString();
    }
}
```

The only real difference in this code is that the page class is no longer implicit—instead it is declared to contain all the page methods.

Overall, the code-behind model is preferred for complex pages. Although the inline code model is slightly more compact for small pages, as your code and HTML grows it becomes much easier to deal with both portions separately. The code-behind model is also conceptually cleaner, as it explicitly indicates the class you've created and the namespaces you've imported. Finally, the code-behind model introduces the possibility that a web designer may refine the markup in your pages without touching your code. This book uses the code-behind model for all examples.

How Code-Behind Files Are Connected to Pages

Every .aspx page starts with a Page directive. This Page directive specifies the language for the page, and it also tells ASP.NET where to find the associated code (unless you're using inline code, in which case the code is contained in the same file).

You can specify where to find the associated code in several ways. In previous versions of ASP.NET, it was common to use the Src attribute to point to the source code file or the Inherits attribute to indicate a compiled class name. However, both of these options have their idiosyncrasies. For example, with the Inherits attribute, you're forced to always precompile your code, which is tedious (and can cause problems in development teams, because the standard option is to compile every page into a single DLL assembly). But the real problem is that both approaches force you to declare every web control you want to use with a member variable. This adds a lot of boilerplate code.

In ASP.NET 2.0, you can solve the problem using a new language feature called *partial classes*, which let you split a single class into multiple source code files. Essentially, the model is the same as before, but the control declarations are shuffled into a separate file. You, the developer, never need to be distracted by this file—instead you can just access your web-page controls by name. Keen eyes will have spotted the word *partial* in the class declaration for your web-page code:

```
public partial class TestFormCodeBehind : System.Web.UI.Page
{ ... }
```

With this bit of infrastructure in place, the rest is easy. Your .aspx page links to the source code file using the CodeFile attribute, as shown here:

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

Notice that Visual Studio uses a slightly unusual naming syntax for the source code file. It has the full name of the corresponding web page, complete with the .aspx extension, followed by the .cs extension at the end. This is just a matter of convention, and it avoids a problem if you happen to create two different code-behind file types (for example, a web page and a web service) with the same name.

How Control Tags Are Connected to Page Variables

When you request your web page in a browser, ASP.NET starts by finding the associated code file. Then, it generates a variable declaration for each control you have. For example, imagine you have a text box named txtInput:

```
<asp:TextBox id="txtInput" runat="server"/>
```

ASP.NET generates the following member variable declaration and merges it with your page class using the magic of partial classes:

```
protected System.Web.UI.TextBox txtInput;
```

Of course, you won't see this declaration. But you rely on it every time you write a line of code that refers to the txtInput object (either to read or to write a property):

```
txtInput.Text = "Hello.";
```

To make sure this system works, you must keep both the .aspx markup file (with the control tags) and the .cs file (with the source code) synchronized. If you edit control names in one piece using another tool (such as a text editor), you'll break the link, and your code won't compile.

Incidentally, you'll notice that control variables are always declared with the protected accessibility keyword. That's because of the way ASP.NET uses inheritance in the web-page model. The following three layers are at work:

- First, the Page class from the .NET class library defines the basic functionality that allows a web page to host other controls, render itself to HTML, and provide access to the traditional ASP objects such as Request, Response, and Session.
- Second, your code-behind class (for example, TestFormCodeBehind) inherits from the Page class to acquire the basic set of ASP.NET web-page functionality.
- Finally, the .aspx page (for example, HelloWorldPage.aspx) inherits the code from the custom page class you created. This allows it to combine the user interface with the code that supports it.

Protected variables act like private variables with a key difference—they are accessible to derived classes. In other words, using protected variables in your code-behind class ensures that the variables are accessible in the derived page class. This allows ASP.NET to connect your control variables to your control tags at runtime.

How Events Are Connected to Event Handlers

Most of the code in an ASP.NET web page is placed inside event handlers that react to web control events. Using Visual Studio, you can add an event handler to your code in three ways:

Type it in by hand: In this case, you add the method directly to the page class. You must specify the appropriate parameters so that the signature of the event handler exactly matches the signature of the event you want to handle. You'll also need to edit the control tag so that it links the control to the appropriate event handler. (Alternatively, you can use delegates to wire this up programmatically.)

Double-click a control in design view: In this case, Visual Studio will create an event handler for that control's default event (and adjust the control tag accordingly). For example, if you double-click the page, it will create a Page.Load event handler. If you double-click a Button control, Visual Studio will create an event handler for the Click event.

Choose the event from the Properties window: Just select the control, and click the lightning bolt in the Properties window. You'll see a list of all the events provided by that control. Double-click in the box next to the event you want to handle, and Visual Studio will automatically generate the event handler in your page class and adjust the control tag.

The second and third options are the most convenient. The third option is the most flexible, because it allows you to select a method in the page class that you've already created. Just select the event in the Properties window, and click the drop-down arrow at the right. You'll see a list that includes all the methods in your class that match the signature this event requires. You can then choose a method from the list to connect it. Figure 2-21 shows an example where the Button.Click event is connected to the Button_Click() method in your page class. The only limitation of this technique is that it works exclusively with web controls, not server-side HTML controls.

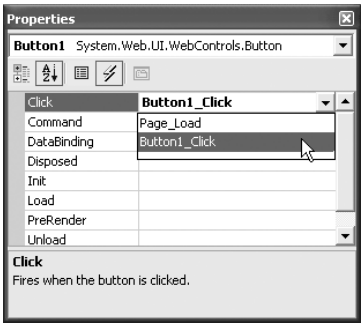

Figure 2-21. *Attaching an event handler*

Visual Studio 2005 uses *automatic event wire-up*, as indicated in the Page directive. Automatic event wire-up has two basic principles:

- All page event handlers are connected automatically based on the name of the event handler. In other words, the Page_Load() method is automatically called when the page loads. Visual Studio adds a comment to your page class to point out the commonly used event methods.
- All control event handlers are connected using attributes in the control tag. The attribute has the same name as the event, prefixed by the word *On*.

For example, if you want to handle the Click event of the Button control, you simply need to set the OnClick attribute in the control tag with the name of the event handler you want to use. Here's the change you need:

```
<asp:Button id="cmdOK" OnClick="txtName_Click" runat="server">
```

ASP.NET controls always use this syntax. Remember, because ASP.NET must connect the event handlers, the derived page class must be able to access the code-behind class. This means your event handlers must be declared with the protected or public keyword. (Protected is preferred, because it prevents other classes from seeing this method.)

Of course, if you're familiar with .NET events, you know there's another approach to connect an event handler. You can do it dynamically through code using delegates. Here's an example:

```
cmdOK.Click += new EventHandler(txtName_Click);
```

This approach is useful if you're creating controls on the fly. You'll see this technique in action in Chapter 3.

Visual Studio Debugging

Visual Studio has always provided robust tools for debugging your web applications. In Visual Studio 2005, these tools remain essentially the same, with some minor enhancements that make it easier to drill into live objects and collections at runtime.

To debug a specific web page in Visual Studio, select that web page in the Solution Explorer, and click the Start Debugging button on the toolbar. (If you are currently editing the web page you want to test, you don't need to select it at all—just click Start Debugging to launch it directly.)

What happens next depends on the location of your project. If your project is stored on a remote web server or a local IIS virtual directory, Visual Studio simply launches your default browser and directs you to the appropriate URL. If you've used a file system application, Visual Studio starts its integrated web server on a dynamically selected port (which prevents it from conflicting with IIS, if it's installed). Then Visual Studio launches the default browser and passes it a URL that points to the local web server. Either way, the real work—compiling the page and creating the page objects—is passed along to the ASP.NET worker process.

■ **Tip** Visual Studio's built-in web server allows you to retrieve a file listing. This means if you create a web application named MyApp, you can make a request in the form of `http://localhost:port/MyApp` to see a list of pages. Then, just click the page you want to test. This process assumes your web application doesn't have a default.aspx page—if it does, any requests for the website root automatically return this page.

The separation between Visual Studio, the web server, and ASP.NET allows for a few interesting tricks. For example, while your browser window is open, you can still make changes to the code and tags of your web pages. Once you've completed your changes, just save the page, and click the Refresh button in your browser to rerequest it. Although you'll always be forced to restart the entire page to see the results of any changes you make, it's still more convenient than rebuilding your whole project.

Fixing and restarting a web page is handy, but what about when you need to track down an elusive error? In these cases, you need Visual Studio's debugging smarts, which are described in the next few sections.

■ **Note** When you use the test web server, it runs all code using your user account. This is different from the much more limited behavior you'll see in IIS, which uses a less-privileged account to ensure security. It's important to understand the difference, because if your application accesses protected resources (such as the file system, a database, the registry, or an event log), you'll need to make sure you explicitly allow the IIS user. For more information about IIS and the hosting model, refer to Chapter 18.

Single-Step Debugging

Single-step debugging allows you to execute your code one line at a time. It's incredibly easy to use. Just follow these steps:

1. Find a location in your code where you want to pause execution, and start single-stepping (you can use any executable line of code but not a variable declaration, comment, or blank line). Click in the margin next to the line code, and a red breakpoint will appear (see Figure 2-22).

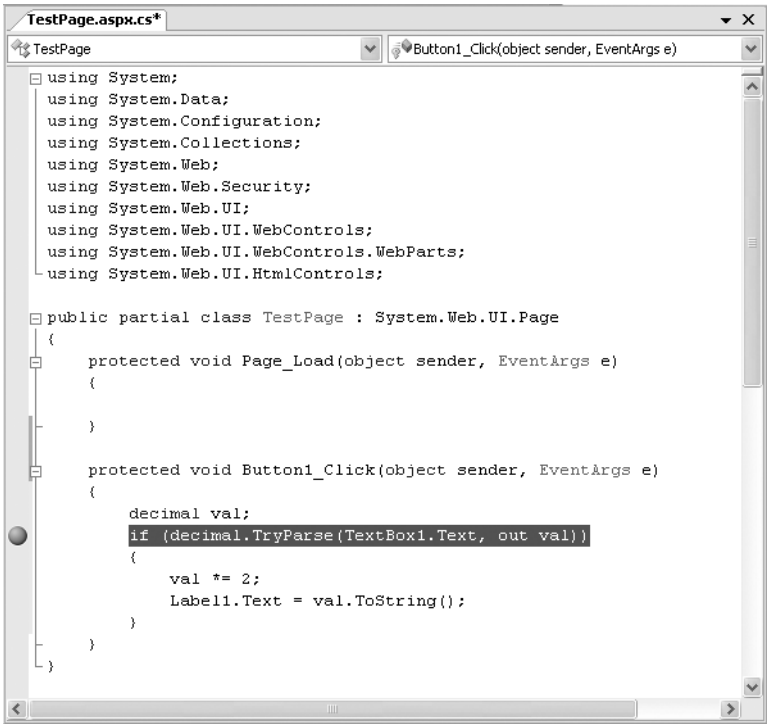

Figure 2-22. *Setting a breakpoint*

2. Now start your program as you would ordinarily. When the program reaches your breakpoint, execution will pause, and you'll be switched back to the Visual Studio code window. The breakpoint statement won't be executed.
3. At this point, you have several options. You can execute the current line by pressing F11. The following line in your code will be highlighted with a yellow arrow, indicating that this is the next line that will be executed. You can continue like this through your program, running one line at a time by pressing F11 and following the code's path of execution. Or, you can exit break mode and resume running your code by pressing F5.

■ **Note** Instead of using shortcut keys such as F11 and F5, you can use the buttons in the Visual Studio Debug toolbar. Alternatively, you can right-click the code window and choose an option from the context menu.

---

4. Whenever the code is in break mode, you can hover over variables to see their current contents. This allows you to verify that variables contain the values you expect (see Figure 2-23). If you hover over an object, you can drill down into all the individual properties by clicking a small plus symbol to expand it (see Figure 2-24).

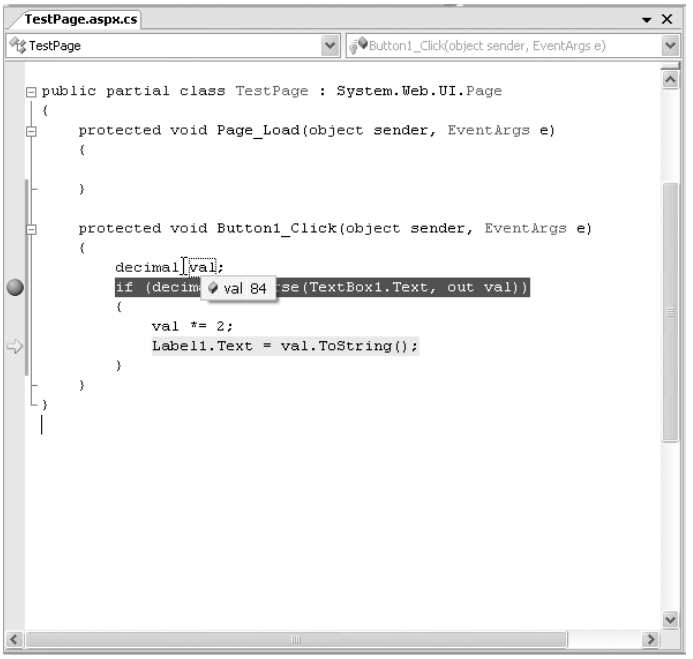

Figure 2-23. *Viewing variable contents in break mode*

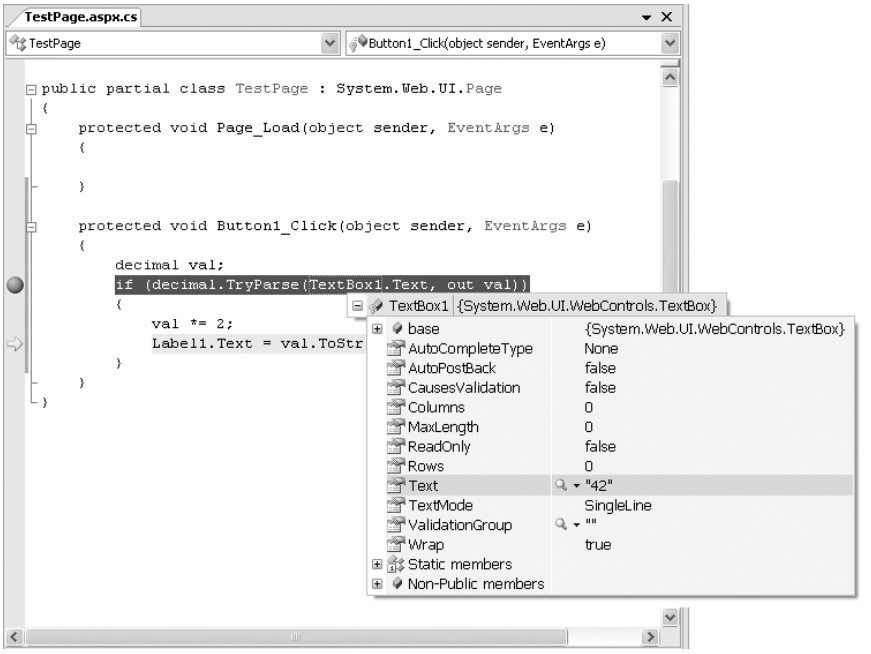

Figure 2-24. *Viewing object properties in break mode*

Tip You can even modify the values in a variable or property directly—just click inside the tooltip, and enter the new value. This allows you to simulate scenarios that are difficult or time-consuming to re-create manually or to test specific error conditions.

5. You can also use any of the commands listed in Table 2-5 while in break mode. These commands are available from the context menu by right-clicking the code window or by using the associated hot key.

Table 2-5. *Commands Available in Break Mode*

Command (Hot Key)	Description
Step Into (F11)	Executes the currently highlighted line and then pauses. If the currently highlighted line calls a procedure, execution will pause at the first executable line inside the method or function (which is why this feature is called stepping *into*).
Step Over (F10)	The same as Step Into, except that it runs procedures as though they are a single line. If you select the Step Over command while a procedure call is highlighted, the entire procedure will be executed. Execution will pause at the next executable statement in the current procedure.
Step Out (Shift+F11)	Executes all the code in the current procedure and then pauses at the statement that immediately follows the one that called this method or function. In other words, this allows you to step "out" of the current procedure in one large jump.
Continue (F5)	Resumes the program and continues to run it normally without pausing until another breakpoint is reached.
Run to Cursor	Allows you to run all the code up to a specific line (where your cursor is currently positioned). You can use this technique to skip a time-consuming loop.
Set Next Statement	Allows you to change your program's path of execution while debugging. It causes your program to mark the current line (where your cursor is positioned) as the current line for execution. When you resume execution, this line will be executed, and the program will continue from that point.
Show Next Statement	Moves focus to the line of code that is marked for execution. This line is marked by a yellow arrow. The Show Next Statement command is useful if you lose your place while editing.

You can switch your program into break mode at any point by clicking the pause button in the toolbar or by selecting Debug ➤ Break All.

Advanced Breakpoints

Choose Debug ➤ Windows ➤ Breakpoints to see a window that lists all the breakpoints in your current project. The Breakpoints window provides a hit count, showing you the number of times a breakpoint has been encountered (see Figure 2-25). You can jump to the corresponding location in code by double-clicking a breakpoint. You can also use the Breakpoints window to disable a breakpoint without removing it. That allows you to keep a breakpoint to use in testing later, without leaving it active. Breakpoints are automatically saved with the hidden solution file described earlier.

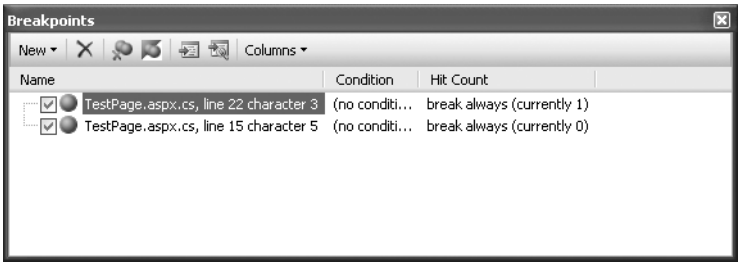

Figure 2-25. *The Breakpoints window*

Visual Studio allows you to customize breakpoints so they occur only if certain conditions are true. To customize a breakpoint, right-click it, and select Breakpoint Properties. In the window that appears, you can take one of the following actions:

- Click the Condition button to set an expression. You can choose to break when this expression is true or when it has changed since the last time the breakpoint was hit.
- Click the Hit Count button to create a breakpoint that pauses only after a breakpoint has been hit a certain number of times (for example, at least 20) or a specific multiple of times (for example, every fifth time).

Variable Watches

In some cases, you might want to track the status of a variable without switching into break mode repeatedly. In this case, it's more useful to use the Locals, Autos, and Watch windows, which allow you track variables across an entire application. Table 2-6 describes these windows.

Table 2-6. *Variable Tracking Windows*

Window	Description
Locals	Automatically displays all the variables that are in scope in the current procedure. This offers a quick summary of important variables.
Autos	Automatically displays variables that Visual Studio determines are important for the current code statement. For example, this might include variables that are accessed or changed in the previous line.
Watch	Displays variables you have added. Watches are saved with your project, so you can continue tracking a variable later. To add a watch, right-click a variable in your code, and select Add Watch; alternatively, double-click the last row in the Watch window, and type in the variable name.

Each row in the Locals, Autos, and Watch windows provides information about the type or class of the variable and its current value. If the variable holds an object instance, you can expand the variable and see its private members and properties. For example, in the Locals window you'll see the this variable, which is a reference to the current page class. If you click the plus symbol next to this, a full list will appear that describes many page properties (and some system values), as shown in Figure 2-26.

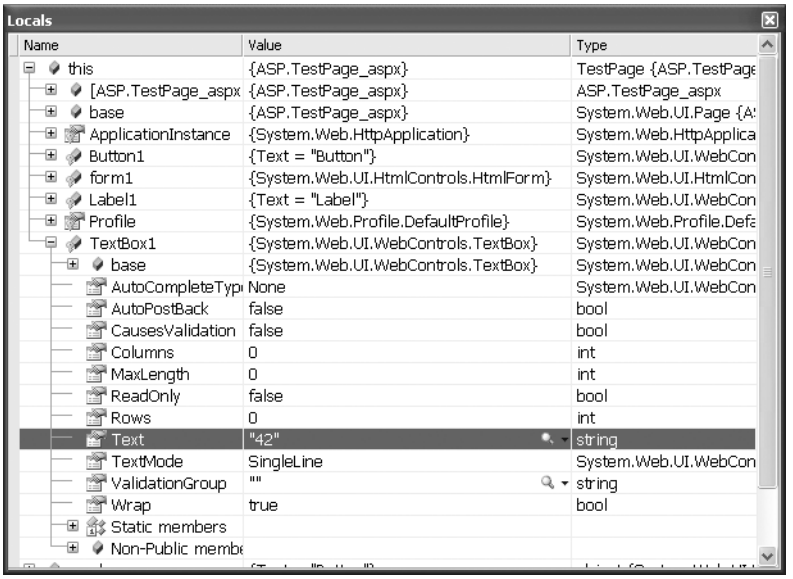

Figure 2-26. *Viewing the current page class in the Locals window*

The Locals, Autos, and Watch windows allow you to change variables or properties while your program is in break mode. Just double-click the current value in the Value column, and type in a new value. If you are missing one of the watch windows, you can show it manually by selecting it from the Debug ➤ Windows submenu.

Visual Studio Macros

One of the most exciting frills of the Visual Studio development environment is its powerful macro and add-in framework (which is largely unchanged from previous versions of Visual Studio .NET). This framework, known as the Visual Studio Automation model, provides almost 200 objects that give you unprecedented control over the IDE, including the ability to access and manipulate the current project hierarchy, the collection of open windows, and the integrated debugger. One of the most convenient and flexible Automation tools is the macro facility.

The simplest macro is a keystroke recording. To create a simple keystroke macro, select Tools ➤ Macros ➤ Record Temporary Macro from the Visual Studio menu, and press the appropriate keystrokes. Once you're finished, click the stop button on the floating macro toolbar. You can now replay the recorded macro (with the Ctrl+Shift+P shortcut key).

■ **Note** Visual Studio allows only one recorded macro, which is overwritten every time you record a new one. To make a temporary macro permanent, you'll need to cut and paste the code into a different subroutine.

A good way to start learning about macros is to use the record facility and then look at the code it generates. Select Tool ➤ Macros ➤ Macro Explorer to see a window that shows a tree of macro modules and the macros they contain (see Figure 2-27). Each macro corresponds to a Visual Basic subroutine. (Unfortunately, C# is not supported.) To edit the macro you just created, right-click the TemporaryMacro subroutine in the RecordingModule, and select Edit.

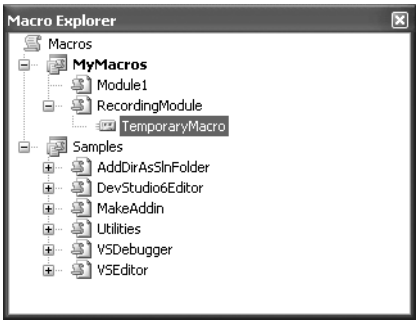

Figure 2-27. *The Macro Explorer*

Macro code uses a special DTE (design-time environment) object model. The DTE hierarchy provides the core features that allow you to interact with every aspect of the IDE. Some of the ingredients at your fingertips include the following:

- Window objects (used to close, rearrange, or otherwise manipulate open windows)
- Document objects (used to edit text)
- Solution and project objects (used to manage the underlying files and project collection)
- Tool-window objects (used to configure the IDE's interface)
- Debugging objects (used for tasks such as creating breakpoints and halting execution)
- Event objects (used to react to IDE events)
- Code-manipulation objects (used to analyze your project's code constructs)

For example, the following macro automatically lists all the files in the project that have been modified but not saved. The list is shown in the Output window.

```
Sub ListModifiedDocuments()
    Dim win As Window = DTE.Windows.Item(Constants.vsWindowKindCommandWindow)
    Dim target As Object

    ' If the current window is an Output window, use it. Otherwise, use a
    ' helper function to find and activate the window.
    If (DTE.ActiveWindow Is win) Then
        target = win.Object
    Else
        target = GetOutputWindowPane("Modified Documents")
        target.clear()
    End If

    ' Loop through all the open documents, and if unsaved changes are detected,
    ' write the document name to the Output window.
    Dim doc As Document
    For Each doc In DTE.Documents
        If Not doc.Saved Then
            target.OutputString(doc.Name & "   " & doc.FullName & _
              Microsoft.VisualBasic.Constants.vbCrLf)
        End If
    Next
End Sub
```

Figure 2-28 shows the result of running this macro.

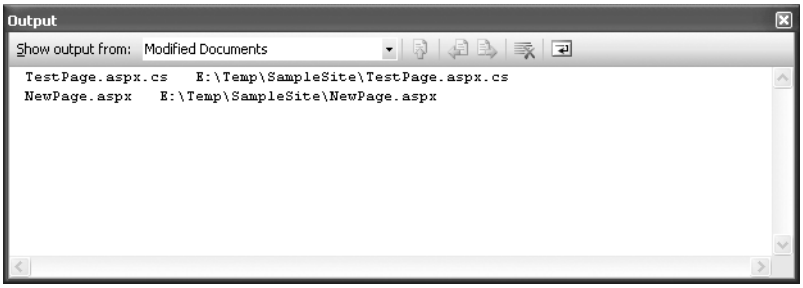

Figure 2-28. *Detecting changed documents*

This is only one of several dozen useful macros that are included in the Samples macro project, which comes with Visual Studio 2005 (and the code download for this chapter). To learn more about Visual Studio macros and add-ins, you can consult a dedicated book on the subject. Several good titles exist for the previous version of Visual Studio .NET, including *Inside Microsoft Visual Studio .NET* (Microsoft Press, 2003).

■ **Tip** Many useful Visual Studio macros are installed by default with Visual Studio 2005. Look under the Samples group in the Macro Explorer, which has macros for adding comments, switching on and off line numbers, inserting dates and times, formatting code, and debugging. You can also download more advanced add-ins from `http://msdn.microsoft.com/vstudio/downloads/samples`. These samples can do everything from automating the build process and integrating with Outlook to spell-checking the text in your user interface.

ASP.NET Development Helper

Another interesting tool that's only begun its development is the ASP.NET Development Helper, a free tool created by Nikhil Kothari from the ASP.NET team. The central goal of the ASP.NET Development Helper is to improve the debugging experience for ASP.NET developers by enhancing the ability of the browser to participate in the debugging process. Currently, the ASP.NET Development Helper is limited to just a few useful features:

- It can report whether a page is in debug or tracing mode.
- It can display the view state information for a page.
- It can display the trace information for a page (and hide it from the page, making sure your layout isn't cluttered).
- It can clear the cache or trigger an application restart.

Many of these features haven't been covered yet, but you'll see a brief example of the ASP.NET Development Helper in the next chapter.

The design of the ASP.NET Development Helper is quite interesting. Essentially, it's built out of two pieces:

- An HTTP module that runs on the web server and makes additional information available to the client browser. (You'll learn about HTTP modules in Chapter 5.)
- An unmanaged browser plug-in that communicates with the HTTP module and displays the important information in a side panel in the browser (see Figure 2-29). The browser plug-in is designed exclusively for Internet Explorer, but at least one other developer has already created a Firefox version that works with the same HTTP module.

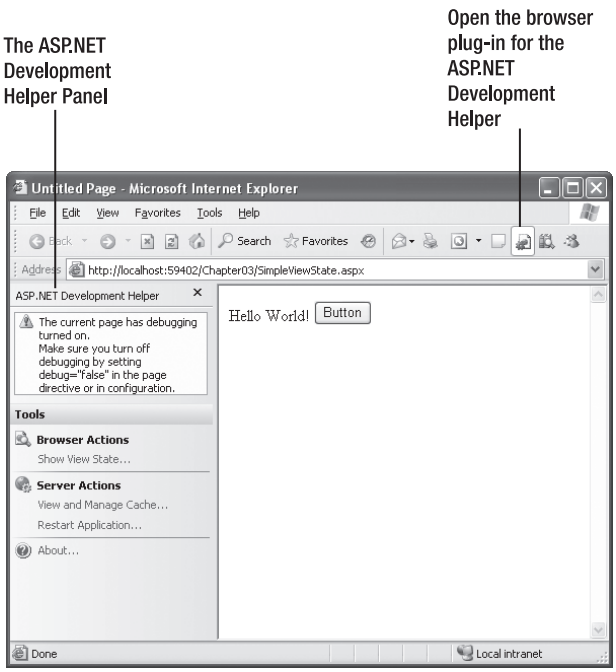

Figure 2-29. *The ASP.NET Development Helper*

To download the ASP.NET Development Helper, surf to `http://www.nikhilk.net/ASPNETDevHelperTool.aspx`. There you can download two DLLs, one for the HTTP module (WebDevInfo.dll) and one for the browser plug-in (WebDevInfo.BHO.dll). Copy these to any directory.

Then, install the browser extension with the following command line:

```
regsvr32 nStuff.WebDevInfo.BHO.dll
```

Next, you need install the assembly for the HTTP module into the GAC. You can do this by dragging and dropping in Windows Explorer, but it's generally easier to use the gacutil.exe utility. Start a Visual Studio command prompt (choose Programs ➤ Visual Studio 2005 ➤ Visual Studio Tools ➤ Visual Studio 2005 Command Prompt from the Start menu), and then run this command:

```
gacutil /i nStuff.WebDevInfo.dll
```

Now, when you want to use this tool with a web application, you need to modify the web.config file so it loads the HTTP module. The content you need depends on the exact version of the tool you're using, but it looks something like this:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <httpModules>
      <add name="DevInfo" type="nStuff.WebDevInfo.DevInfoModule, nStuff.WebDevInfo,
Version=0.5.0.0, Culture=neutral, PublicKeyToken=8fc0e3af5abcb6c4" />
    </httpModules>
    ...
  </system.web>
</configuration>
```

Now, run one of the pages from this application. To actually call up the browser plug-in, look for a button (with a gear icon) in the browser, which will have been added to the end of the Standard toolbar. When you click this icon, you'll see a display like the one shown in Figure 2-29 (assuming you're currently viewing an ASP.NET page from an application that has the matching HTTP module loaded).

You'll see the ASP.NET Developer Helper at work in Chapter 3.

Summary

This chapter considered the role that Visual Studio can play in helping you develop your web applications. At the same time that you explored its rich design-time environment, you also learned about how it works behind the scenes with the code-behind model and how to extend it with time-saving features such as macros. In the next two chapters, you'll jump into full-fledged ASP.NET coding by examining web pages and server controls.