Chapter 2

Creating an ASP.NET Web Application

In this chapter, you will learn how to:

- Create a new Web application with Microsoft Visual Studio .NET.
- Add a Web Forms page to a Web application project.
- Add Server Controls to a Web Forms page and modify their properties.
- Write code in event handlers.
- Build and test a Web application.

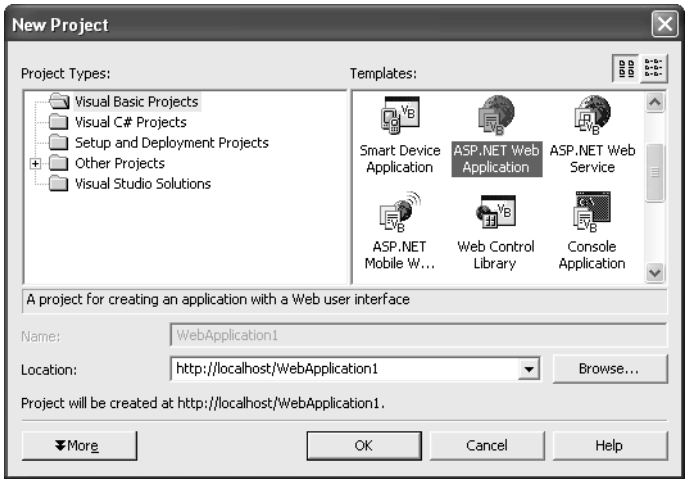
Now that you've learned about some of the features of Visual Studio .NET, the next step is to take advantage of them in your own applications. Conveniently enough, that's precisely what you're going to learn how to do in this chapter.

You'll begin with an overview of the two major project types used for ASP.NET applications. Then you'll look at the file types used in ASP.NET and the purpose of each. Next, you'll learn how to create a new Web application, add a new Web Forms page, and add controls to the page and manipulate their properties. Finally, you'll learn how to add event-handler code to the page, build the project, and test the page.

ASP.NET Project Types

There are three basic types of ASP.NET applications, each with a distinct purpose. *ASP.NET Web Applications* are for a Web application that will provide its own HTML-based user interface. *ASP.NET Web Services* are for Web-based functionality that will be accessed programmatically. *ASP.NET Mobile Web*

Applications, new in Visual Studio .NET 2003, are designed for creating Web applications targeted at Personal Digital Assistants (PDAs), cell phones, and other mobile devices. You can develop all of these application types with or without Visual Studio .NET, although the Visual Studio environment makes developing them significantly easier and faster. The following illustration shows the Visual Studio .NET New Project dialog box displaying the ASP.NET Web Application, ASP.NET Web Service, and ASP.NET Mobile Web Application project templates for Visual Basic .NET (some of the project templates shown might not appear in all editions of Visual Studio .NET).



ASP.NET Web Applications

ASP.NET applications, at their simplest, are much like classic ASP applications. The elements of a simple ASP.NET application are

- A virtual directory in IIS, configured as an application root, to hold the files that make up the application and to control access to the files.
- One or more .aspx files.
- A Global.asax file (analogous to the Global.asa file in classic ASP) to deal with Session and Application startup and clean-up logic. This file is optional.
- A Web.config file used to store configuration settings. This file is optional, and is new for ASP.NET.

For Visual Studio .NET users, the good news is that all of the preceding files are created for you when you create a new Web application project.

ASP.NET Web Forms

Web Forms are an important part of any ASP.NET Web application. Put simply, they are ASP.NET pages that use *ASP.NET Server* Controls. The Web Forms programming model makes it possible (and relatively easy) to develop Web-based applications in much the same way that today's Microsoft Visual Basic programmers develop Microsoft Windows–based applications that have a GUI.

Web Forms in Visual Studio .NET allow you to create rich, interactive applications simply by dragging and dropping controls onto a page and then writing minimal code to handle user interaction, events, and so on. In addition, the Visual Studio .NET environment lets you work on your pages either visually—using the Web Forms Designer—or textually, using the powerful Visual Studio .NET source-code editor.

You can write code in your Web Forms in one of two ways: inline in the .aspx file (as is typical of a classic ASP page), or using a code-behind module. Although it's possible to write your application with code in the actual .aspx file and still take advantage of compiled code and the other improvements of .NET, I recommend that you get in the habit of using code-behind modules. Visual Studio .NET defaults to using code-behind for UI-specific programming logic.

Code-Behind

Code-behind is a new feature in ASP.NET that allows developers to truly separate the HTML and tag-based UI elements from the code that provides user interaction, validation, and so on. Code-behind modules offer developers a number of advantages:

- **Clean separation of HTML and code** Code-behind allows HTML designers and developers to do what they do best independently, thereby minimizing the possibility of messing up one another's work (something that happens all too frequently when developing classic ASP applications).
- **Easier reuse** Code that isn't interspersed with HTML in an .aspx page can be more easily reused in other projects.
- **Simpler maintenance** Because the code is separated from the HTML, your pages will be easier to read and maintain.
- **Deployment without source code** Visual Studio .NET projects using code-behind modules can be deployed as compiled code (in addition to the .aspx pages), allowing you to protect your source code. This can be very useful if you're creating applications for clients but want to retain control of your intellectual property.

All in all, it's worthwhile to get into the habit of using code-behind. You'll see examples of code-behind throughout the book.

ASP.NET Web Services

Although no one would deny that Web applications created with ASP.NET (or even with classic ASP) can be very useful, one feature that has long been missing is an easy way to provide programmatic functionality over the Internet or an intranet without tying the client to a specific UI. This is where ASP.NET Web services come in.

A Web service, at its simplest, is a chunk of programming code that is accessible over the Web. Web services are based on the World Wide Web Consortium's (W3C) SOAP specification. This allows computers on varying platforms, from Windows servers to UNIX workstations, to offer and consume programmatic services over the HTTP protocol.

▶ **Note** SOAP can use other protocols, such as FTP or SMTP, but HTTP is the most common protocol used with SOAP and Web services because most firewalls allow communication via the HTTP protocol.

ASP.NET makes it remarkably easy to implement Web services. In fact, all it takes is adding an appropriate declaration to any method you want to make available as a Web service. Visual Studio .NET makes it even easier by taking care of all the work necessary to make your Web service available to potential clients. Part IV will discuss Web services in greater detail.

ASP.NET Mobile Web Applications

Like standard ASP.NET Web applications, ASP.NET Mobile Web applications contain a number of standard elements:

- A virtual directory in IIS, configured as an application root, to hold the files that make up the application and to control access to the files.
- One or more .aspx files. Visual Studio .NET creates a single Web form by default with the name MobileWebForm1.aspx. The default Web Form contains a special *<mobile:form>* element, which acts as a container for mobile Web server controls (formerly part of the Microsoft Mobile Internet Toolkit). These controls allow you to design a UI for your application that automatically adapts to a wide variety of mobile devices.

- A Global.asax file to deal with Session and Application startup and clean-up logic. This file is optional.
- A Web.config file used to store configuration settings. In a Mobile Web Application project created with Visual Studio .NET, this file also contains a set of filters that help tailor page output to various mobile devices. This file is optional.

As with the ASP.NET Web Application template, Visual Studio .NET creates all of the preceding files when you create a new ASP.NET Mobile Web application project. You'll learn more about developing ASP.NET applications for mobile devices in Chapter 8.

ASP.NET File Types

You'll see a number of new file types in your ASP.NET applications. To avoid any confusion, let's take a minute to go over the ones you'll see most often and discuss how they're used.

- **.aspx** The extension you'll see most often. Analogous to the .asp extension in classic ASP, .aspx is used for Web Forms pages.
- **.ascx** The extension used for Web Forms user controls. User controls provide one of the many ways available in ASP.NET to reuse code. Similar to include files in classic ASP, .ascx files can be as simple as a few HTML tags or can include complex logic that the author might want to reuse in many pages. User controls are added to a Web Forms page using the @ *Register* directive, which is discussed in Part III.
- **.asmx** The extension used for files that implement Web services. Web services can be accessed directly through .asmx files, or the .asmx file can direct the request to a compiled assembly that implements the Web service.
- **.vb** The extension for Visual Basic .NET code modules. All Web Forms pages (.aspx) added to a Visual Studio .NET Web application that are written in Visual Basic .NET will have a corresponding .vb code-behind module with the same name as the Web Form page to which it's related (*pagename*.aspx.vb).
- **.resx** Denotes a resource file. These files are used primarily in Windows Forms applications, but are also available to Web application developers for storing resources such as alternative text strings for internationalization of applications.
- **Global.asax** Used to define Application- and Session-level variables and startup procedures. Global.asax is used the same way as Global.asa is used in classic ASP. Note that while Global.asax can be

structured like Global.asa, with startup procedures such as *Session_OnStart* (*Session_Start* in ASP.NET) coded directly in the Global.asax file in a <*script runat="server"*> block, Visual Studio implements these procedures in a .vb (or .cs) code-behind module (global.asax.vb) rather than in the Global.asax file itself.

In addition to the functionality available in a classic ASP Global.asa file, which was used for handling Application and/or Session start and end events and declaring Application- and/or Session-level variables, ASP.NET also allows you to import namespaces, link to assemblies, and perform other useful tasks. You'll learn more about Global.asax in Chapter 7.

- **Web.config** A new file type in ASP.NET, used to solve one of the major hassles with classic ASP applications: configuration. The Web.config file is a human- and machine-readable XML-based file that stores all of the configuration settings for a given application (or segment of an application). Web.config files are interpreted hierarchically—a Web.config file in a subdirectory of your application will override the settings of the Web.config file (or files) in its parent directories. The advantage is that configuration settings can be inherited where that is desirable, but you also have very granular control over configuration.

Visual Studio .NET

It's certainly possible to create ASP.NET Web applications in Notepad or another text editor, but if you're doing serious ASP.NET or component development, you'll probably want to work within the Visual Studio .NET environment. The advantages of Visual Studio .NET over simple text editors include

- Robust management of project files and multiple projects
- Integration with the Microsoft Visual SourceSafe source-code control environment
- Visual Tools for working with Web services, Web Forms server controls, and database tools
- Packaging and deployment services for Web applications
- Support for multiple languages within a single IDE, including cross-language inheritance and debugging

That's just a brief list. There's much more to the tool than can be covered in a single chapter. So without further ado, let's look at how to create projects and pages in the Visual Studio .NET environment.

Creating Applications

One of the first things you're going to want to do in order to work with ASP.NET in Visual Studio .NET is create a new project, or in Visual Studio .NET parlance, a Web application.

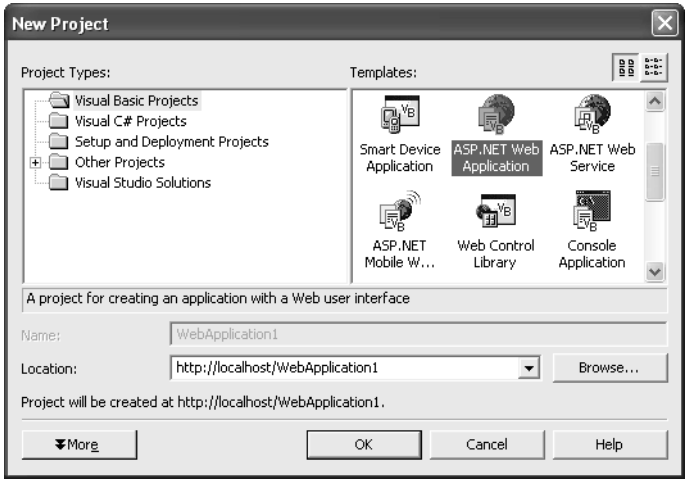
Create an ASP.NET Web application

1. Launch Visual Studio .NET using the techniques you learned in Chapter 1.
2. Open the New Project dialog box using one of the following methods:
   - Click the Create A New Project link on the Visual Studio .NET Start Page (displayed by default when you first open Visual Studio .NET).
   - Click the New Project button (shown in the following illustration), located on the Standard toolbar.

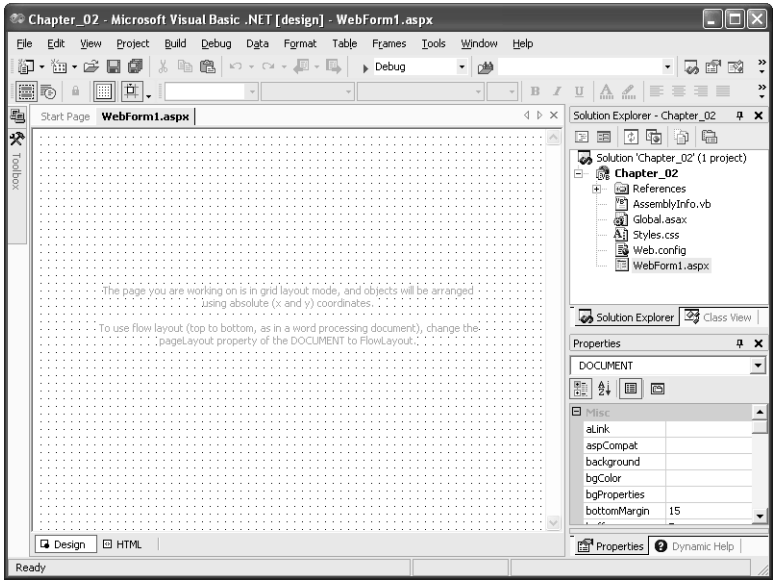
   - From the File menu, select New, and then Project.
3. In the New Project dialog box (see the following illustration), under Project Types, select the Visual Basic Projects folder, then select the appropriate template (ASP.NET Web Application). Type the location as **http://localhost/Chapter_02**, and then click OK.

   Visual Studio .NET will create a new Web application along with physical and virtual directories for the project.



That's it! You've created your first ASP.NET Web application. Note that this application is separate from the Chapter_02 project included with the practice file installation, which is contained in the aspnetsbs solution. Next we'll look at how to add new pages.

In your new Web application, you'll notice that Visual Studio .NET has already added a page named WebForm1.aspx to the project for you and opened it in the editor. Your Visual Studio .NET screen should look similar to the following illustration. However, since one page is rarely enough for most sites, let's look at how to add a new page to your Web application.

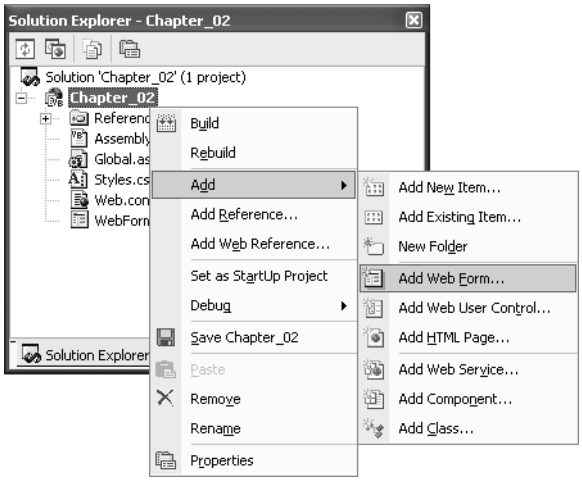


Create a new ASP.NET page (Web Form)

1. Add a new Web Form to your application.

   As with creating a new project, there are several ways to add a new ASP.NET page (Web Form) to your application. The method you choose depends largely on how you like to work. Here are three ways to accomplish this task, although there are others:

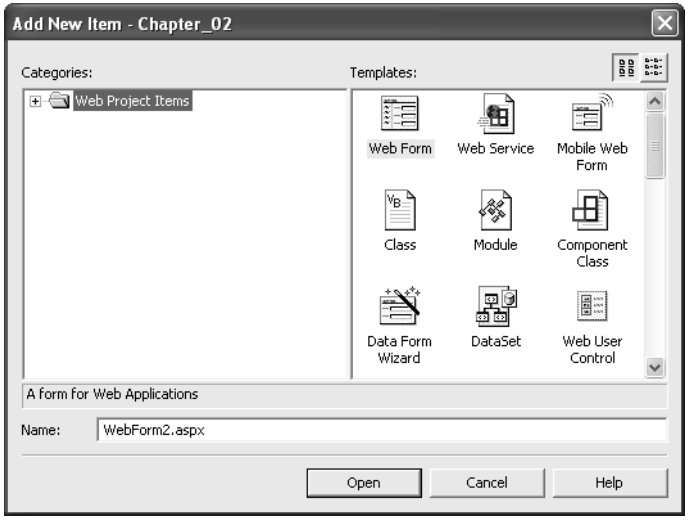
   - In the Solution Explorer window, right-click the application name, then select Add, and then select Add Web Form, as shown in the following illustration.



- On the Visual Studio .NET toolbar, click the Add New Item button (shown in the following illustration) and then select Web Form from the Templates list.

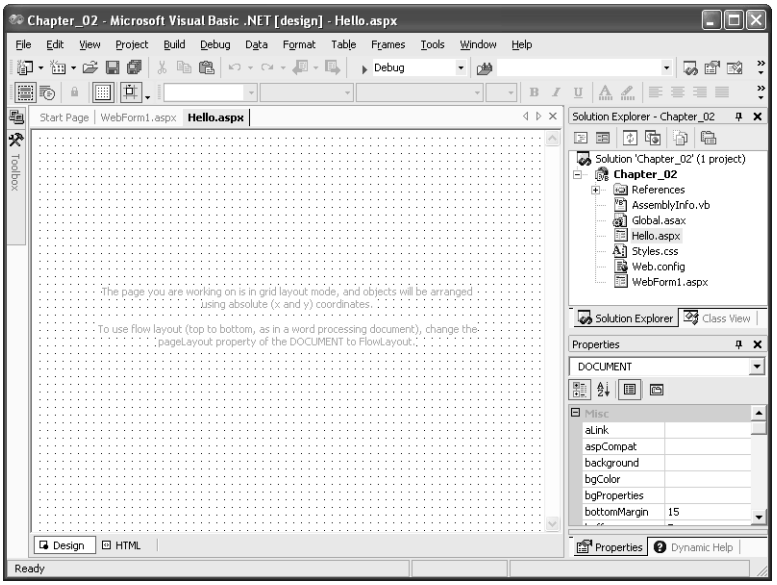
- From the Project menu, select Add Web Form.

Any of these methods will open the Add New Item dialog box, shown in the following illustration.



2 In the Add New Item dialog box, verify that the Web Form template is selected and name the new page Hello.aspx. Click Open.

Visual Studio .NET creates the page, adds it to the project, and opens it in the Web Forms Designer, as shown in the following illustration.



► **Tip** While in the Add New Item dialog box, a brief description of each template is displayed along the bottom of the list window. If you decide to browse the different templates, don't forget to select the proper one (Web Form) before clicking Open.

Adding Server Controls

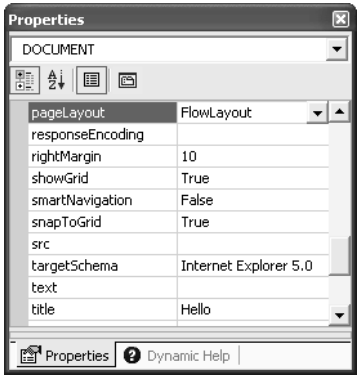
Now that you've created a page for your new application, what can you do with it? Well, let's begin by making it display a "Hello World!" greeting to the client. By default, Web Forms are opened in *GridLayout* mode. Since *GridLayout* relies on cascading style sheets (CSS), which are not supported in all browsers, you might want to change the page layout to *FlowLayout* mode.

Modify Web Form properties

1. With the Web Form open in the Visual Studio .NET designer, click the Web Form page to ensure it is selected.

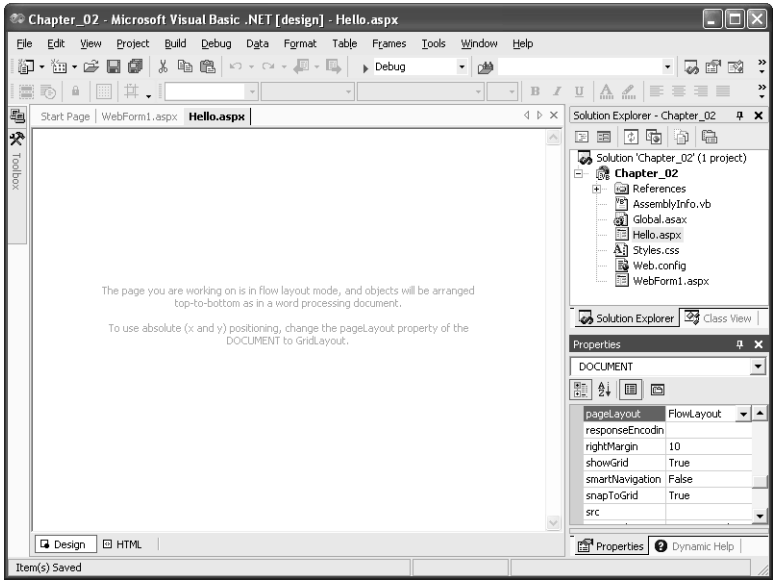
   When the page is selected, the word *DOCUMENT* should appear in the drop-down box at the top of the Properties window.

2 Select the *pageLayout* property from the Properties window, and then use the drop-down list to change its value to *FlowLayout*, as shown in the following illustration.



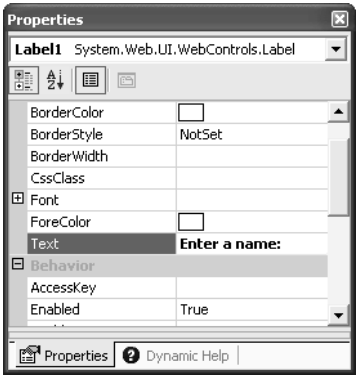
3 Save the page by selecting File, and then Save *filename*, where filename is the name of the file you're currently editing (or by clicking the Save button on the toolbar).

Until you add controls (or HTML elements), the page will display the following message in Design view.



Add controls to a Web Form

1. With the Web Form open in Design mode, place your mouse over the Toolbox tab. (By default, it's found to the left of the code editor/designer window.)
2. When the Toolbox appears, ensure that the Web Forms palette is active. (The title bar of the active palette is shown immediately above the controls displayed in the Toolbox.) If it isn't active, click on its title bar to activate it. Note that the Web Forms palette is available only when a page is in Design mode.
3. With the Web Forms palette active, double-click the *Label* control entry to add an *ASP.NET Label* control to the page. (You might have to let the toolbox hide itself by moving the mouse pointer away from it to see the label on the form.) Once you've added the label, it should be selected by default. If not, click the control to select it.
4. To make the *Label* control display the text you want, you need to change its *Text* property. Select the *Text* property in the Properties window, and then change the text (by default, Label) to **Enter a name:**, as shown in the following illustration.



5. Click the background of the page to place the cursor after the *Label* control you added to the page.
6. Using the Toolbox as in step 3, add a *TextBox* control to the page, and then add a *Button* control to the page.
7. Using the same technique as in step 4, change the *Text* property of the *Button* control to **Submit**.

8 Save the page by clicking the Save button (shown in the following illustration) on the toolbar. You can also save by selecting Save <*filename*> from the File menu.

Add event handler code

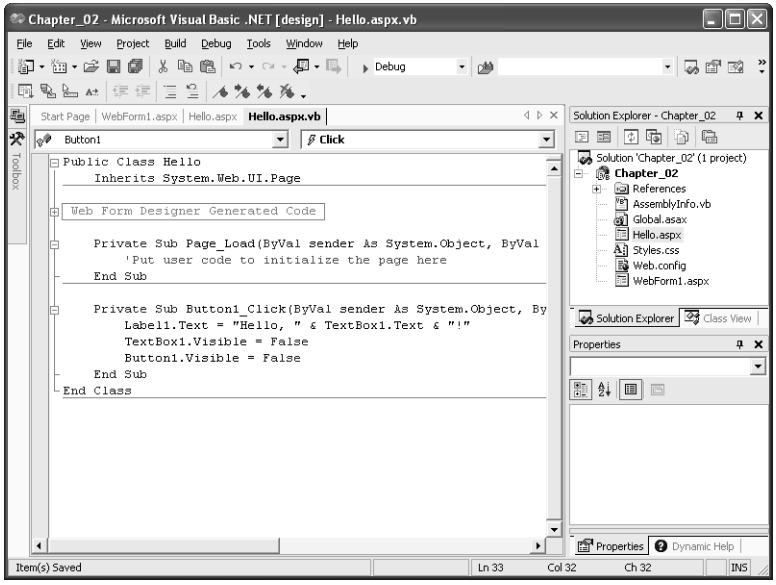
1 To make the page do anything useful, you need to add some code, so double-click the *Button* control.

This will open up the code-behind module for the Web Form and create an event handler procedure called *Button1_Click*.

2 Add the following code to the *Button1_Click* procedure:

```
Label1.Text = "Hello, " & TextBox1.Text & "!"
TextBox1.Visible = False
Button1.Visible = False
```

3 Save the code-behind module, which should now look like the following illustration.



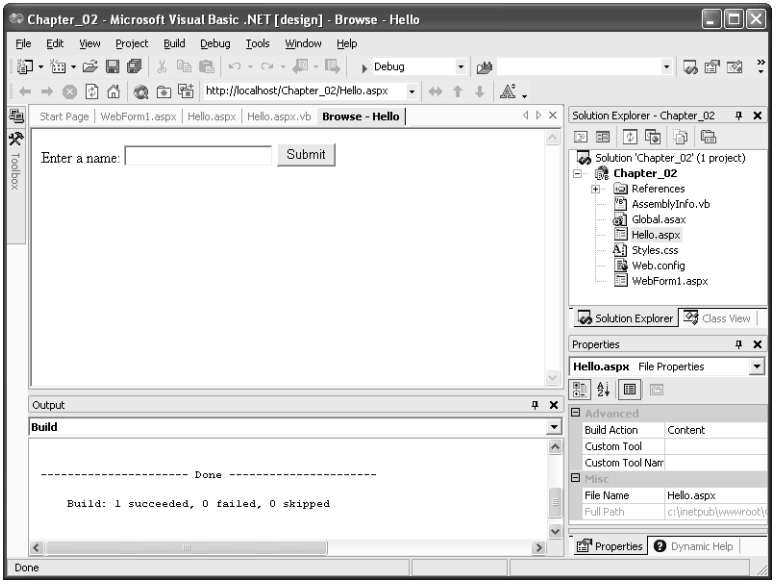
Building and Testing Your Page

Because you modified the code-behind module for the Web Form, you need to build your project before you can browse the page. (You'll learn more about code-behind in later chapters.) *Building* is the process of compiling all of the code modules in the project so they'll be available to the pages and modules that call them. To build the project, from the Build menu, select Build Chapter_02 (or Build Solution, which will build all projects in the solution).

Once you've saved the Web Form page and its code-behind module and built the application, you can test the page.

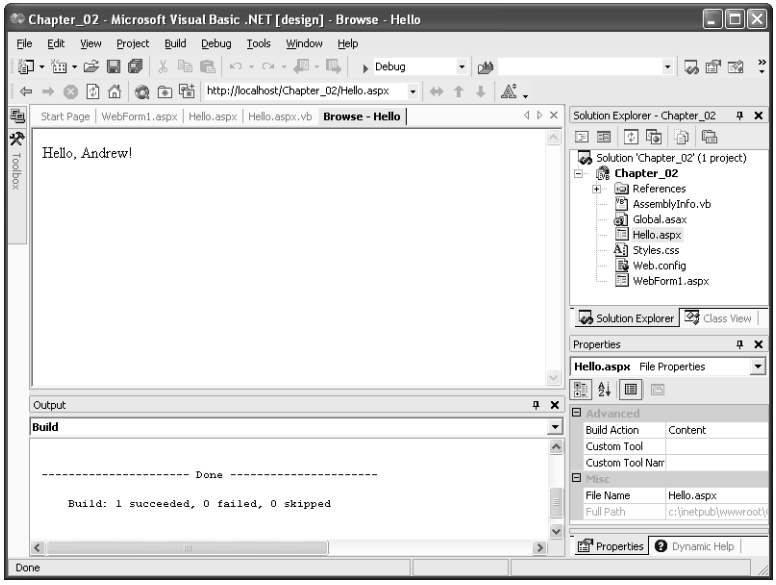
Test your page

1. Right-click the page in Solution Explorer and select View In Browser. The result should look like the following illustration. (You can close the Output window if you want to see more of a page.)



2. Enter your name in the text box and click Submit.

The result should be similar to the following illustration. (Note that the Web toolbar shows the address of the page being browsed. You can enter this address in a browser window on your machine to view the page in a non-embedded browser window.)



Chapter 2 Quick Reference

To	Do this	Button
Create a new project in Visual Studio .NET	Click the New Project button, select the project language and template, and then provide the name and location for the new project.	
Create a new Web Forms page	Click the Add New Item button (or click the arrow to the right and select Add Web Form). Provide a name for the new Web Form and click OK.	
Save a file	Click the Save button, or select Save *<filename>* from the File menu.