ASP.NET 2.0 and ASP.NET Ajax

If any man will draw up his case, and put his name at the foot of the first page, I will give him an immediate reply. Where he compels me to turn over the sheet, he must wait my leisure.
—Lord Sandwich

Rule One: Our client is always right.
Rule Two: If you think our client is wrong, see Rule One.
—Anonymous

A fair question should be followed by a deed in silence.
—Dante Alighieri

You will come here and get books that will open your eyes, and your ears, and your curiosity, and turn you inside out or outside in.
—Ralph Waldo Emerson

OBJECTIVES

In this chapter you will learn:

- Web application development using Active Server Pages .NET (ASP.NET).
- To create Web Forms.
- To create ASP.NET applications consisting of multiple Web Forms.
- To maintain state information about a user with session tracking and cookies.
- To use the **Web Site Administration Tool** to modify web application configuration settings.
- To control user access to web applications using forms authentication and ASP.NET login controls.
- To use databases in ASP.NET applications.
- To design a master page and content pages to create a uniform look-and-feel for a website.

Outline

25.1 Introduction

This chapter introduces **web application development** with Microsoft's **Active Server Pages** .NET (ASP.NET) 2.0 technology. Web-based applications create web content for web-browser clients. This web content includes Extensible HyperText Markup Language (XHTML), client-side scripting, images and binary data. If you are not familiar with XHTML, you should read Chapter 4 before studying this chapter. [*Note:* This chapter assumes that you know Visual Basic and are familiar with the .NET platform version 2.0. To learn more about Visual Basic, check out *Visual Basic 2005 How to Program, Third Edition*, or visit our Visual Basic Resource Center at www.deitel.com/visualbasic.]

We present several examples that demonstrate web application development using **Web Forms**, **web controls** (also called **ASP.NET server controls**) and Visual Basic programming. We also introduce ASP.NET Ajax and use it to enhance one of the earlier examples. Web Form files have the filename extension **.aspx** and contain the web page's GUI. You customize Web Forms by adding web controls including labels, text boxes, images, buttons and other GUI components. The Web Form file generates the web page that is sent to the client browser. From this point onward, we refer to Web Form files as **ASPX files**.

An ASPX file created in Visual Studio is implemented as a class written in a .NET language, such as Visual Basic. This class contains event handlers, initialization code, utility methods and other supporting code. The file that contains this class is called the **code-behind file** and provides the ASPX file's programmatic implementation.

To develop the code and GUIs in this chapter, we used Microsoft Visual Web Developer 2005 Express—an IDE designed for developing ASP.NET web applications. Visual Web Developer and Visual Basic 2005 Express share many common features and visual programming tools that simplify building complex applications, such as those that access a database (Sections 25.5–25.6). The full version of Visual Studio 2005 includes the functionality of Visual Web Developer, so the instructions we present for Visual Web Developer also apply to Visual Studio 2005. Note that you must install either Visual Web Developer 2005 Express (available from `msdn.microsoft.com/vstudio/express/vwd/default.aspx`) or a complete version of Visual Studio 2005 to implement the programs in this chapter.

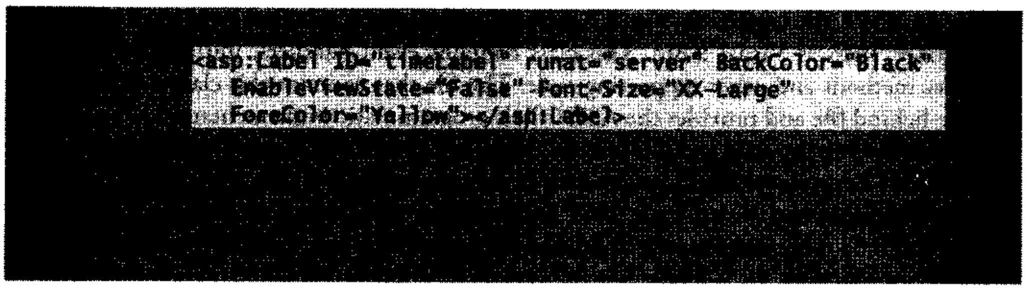
25.2 Creating and Running a Simple Web Form Example

Our first example displays the web server's time of day in a browser window. When run, this program displays the text `A Simple Web Form Example`, followed by the web server's time. As mentioned previously, the program consists of two related files—an ASPX file (Fig. 25.1) and a Visual Basic code-behind file (Fig. 25.2), which we'll discuss in Section 25.2.5. We first display the markup, code and output, then we carefully guide you through the step-by-step process of creating this program. [*Note:* The markup in Fig. 25.1 and other ASPX file listings in this chapter is the same as the markup that appears in Visual Web Developer, but we've reformatted it for presentation purposes to make the code more readable.]

Visual Web Developer generates all the markup shown in Fig. 25.1 when you set the web page's title, type text in the Web Form, drag a `Label` onto the Web Form and set the properties of the page's text and the `Label`. We discuss these steps in Section 25.2.6.

```
<%-- Fig. 25.1: WebTime.aspx --%>
<%-- A page that displays the current time in a Label. --%>
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="WebTime.aspx.vb"
   Inherits="WebTime" EnableSessionState="False" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>A Simple Web Form Example</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h2>
      Current time on the Web server:</h2>
```

Fig. 25.1 | ASPX file that displays the web server's time. (Part 1 of 2.)



```
<asp:Label ID="timeLabel" runat="server" BackColor="Black"
  EnableViewState="False" Font-Size="XX-Large"
  ForeColor="Yellow"></asp:Label>
```

Fig. 25.1 | ASPX file that displays the web server's time. (Part 2 of 2.)

25.2.1 Examining an ASPX File

The ASPX file contains other information in addition to XHTML. Lines 1–2 are ASP.NET **comments** that indicate the figure number, the filename and the purpose of the file. ASP.NET comments begin with `<%--` and terminate with `--%>`. We added these comments to the file. ASP.NET comments are not output as part of the XHTML sent to the client. Lines 3–4 use a **`Page`** directive (in an ASPX file a **directive** is delimited by `<%@` and `%>`) to specify information needed by ASP.NET to process this file. The **`Language`** attribute of the `Page` directive specifies the language of the code-behind file as Visual Basic (`"VB"`); the code-behind file (i.e., the **`CodeFile`**) is `WebTime.aspx.vb`. A code-behind filename usually consists of the full ASPX filename (e.g., `WebTime.aspx`) followed by a filename extension indicating the programming language (`.vb` in this chapter's examples).

The **`AutoEventWireup`** attribute (line 3) determines how Web Form events are handled. When `AutoEventWireup` is set to `true`, ASP.NET determines which methods in the class are called in response to events generated in the `Page`. For example, ASP.NET will call methods `Page_Init` and `Page_Load` in the code-behind file to handle the `Page`'s `Init` and `Load` events, respectively. `AutoEventWireup` requires the event-handling methods to follow specific naming copnventions. (We discuss these events later in the chapter.)

The **`Inherits`** attribute (line 4) specifies the page's class name—in this case, `WebTime`. We say more about `Inherits` momentarily. [*Note:* We explicitly set the **`EnableSessionState`** attribute (line 4) to `False`. We explain the significance of this attribute later in the chapter. The IDE sometimes generates attribute values (e.g., `true` and `false`) and control names (as you will see later in the chapter) that do not adhere to our standard code capitalization conventions (i.e., `True` and `False`). Like Visual Basic, ASP.NET markup is not case sensitive, so using a different case is not problematic. To remain consistent with the code generated by the IDE, we do not modify these values in our code listings or in our accompanying discussions.]

For this first ASPX file, we provide a brief discussion of the XHTML markup. For more information on XHTML, see Chapter 4. Lines 6–7 contain the document type declaration, which specifies the document element name (`HTML`) and the `PUBLIC` Uniform Resource Identifier (URI) for the DTD that defines the XHTML vocabulary.

Lines 9–10 contain the `<html>` and `<head>` start tags, respectively. XHTML documents have the root element `html` and mark up information about the document in the `head` element. Also note that the `html` element specifies the XML namespace of the document using the `xmlns` attribute (see Section 14.4).

Notice the **runat** attribute in line 10, which is set to **"server"**. This attribute indicates that when a client requests this ASPX file, ASP.NET processes the head element and its nested elements on the server and generates the corresponding XHTML, which is then sent to the client. In this case, the XHTML sent to the client will be identical to the markup in the ASPX file. However, as you will see, ASP.NET can generate complex XHTML markup from simple elements in an ASPX file.

Line 11 sets the title of this web page. We demonstrate how to set the title through a property in the IDE shortly. Line 13 contains the <body> start tag, which begins the body of the XHTML document; the body contains the main content that the browser displays. The form that contains our XHTML text and controls is defined in lines 14–24. Again, the runat attribute in the form element indicates that this element executes on the server, which generates equivalent XHTML and sends it to the client. Lines 15–23 contain a div element that groups the elements of the form in a block of markup.

Software Engineering Observation 25.1

Most ASP.NET controls must be placed in a form element in which the <form> tag has the runat="server" attribute.

Lines 16–17 are an XHTML h2 heading element. As we demonstrate shortly, the IDE generates this element in response to typing text directly in the Web Form and selecting the text as a second-level heading.

Lines 18–22 contain a p element to mark up a paragraph of content in the browser. Lines 19–21 mark up a Label web control. The properties that we set in the **Properties** window, such as Font-Size and BackColor (i.e., background color), are attributes here. The **ID** attribute (line 19) assigns a name to the control so that it can be manipulated programmatically in the code-behind file. We set the control's **EnableViewState** attribute (line 20) to False. We explain the significance of this attribute later in the chapter.

The **asp:** tag prefix in the declaration of the **Label** tag (line 19) indicates that the label is an ASP.NET web control, not an XHTML element. Each web control maps to a corresponding XHTML element (or group of elements)—when processing a web control on the server, ASP.NET generates XHTML markup that will be sent to the client to represent that control in a web browser.

Portability Tip 25.1

The same web control can map to different XHTML elements, depending on the client browser and the web control's property settings.

In this example, the asp:Label control maps to the XHTML **span** element (i.e., ASP.NET creates a span element to represent this control in the client's web browser). A span element particular element is used because span elements allow formatting styles to be applied to text. Several of the property values that were applied to our label are represented as part of the style attribute of the span element. You will soon see what the generated span element's markup looks like.

The web control in this example contains the runat="server" attribute–value pair (line 19), because this control must be processed on the server so that the server can translate the control into XHTML that can be rendered in the client browser. If this attribute pair is not present, the asp:Label element is written as text to the client (i.e., the control is not converted into a span element and does not render properly).

25.2.2 Examining a Code-Behind File

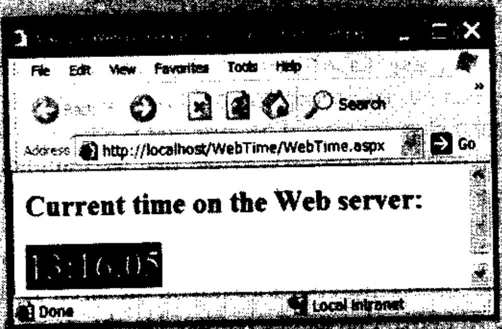
Figure 25.2 presents the code-behind file. Recall that the ASPX file in Fig. 25.1 references `WebTime.aspx.vb` in line 3.

Line 3 (Fig. 25.2) begins the declaration of class `WebTime`. A class declaration can span multiple source-code files, and the separate portions of the class declaration in each file are known as partial classes. The `Partial` modifier in line 3 indicates that the code-behind file is a partial class. We discuss the remainder of this class shortly.

Line 4 indicates that `WebTime` inherits from class **`Page`** in namespace **`System.Web.UI`**. This namespace contains classes and controls that assist in building web-based applications. Class `Page` provides events and objects necessary for creating web-based applications. In addition to class `Page`, `System.Web.UI` also includes class **`Control`**—the base class that provides common functionality for all web controls.

Lines 7–11 define method **`Page_Init`**, which handles the page's **`Init`** event. This event indicates that all the controls on the page have been created and initialized and additional application-specific initialization can now be performed. The only initialization required for this page is setting `timeLabel`'s `Text` property to the time on the server (i.e., the computer on which this code executes). The statement in line 10 retrieves the current time and formats it as *hh*:*mm*:*ss*. For example, 9 AM is formatted as 09:00:00, and 2:30 PM is formatted as 14:30:00. Notice that the code-behind file can access `timeLabel` (the `ID` of the `Label` in the ASPX file) programmatically, even though the file does not contain a declaration for a variable named `timeLabel`. You will learn why momentarily.

```
 1  ' Fig. 25.3: WebTime.aspx.vb
 2  ' Code-behind file for a page that displays the current time.
 3  Partial Class WebTime
 4     Inherits System.Web.UI.Page
 5
 6     ' initializes the contents of the page
 7     Protected Sub Page_Init(ByVal sender As Object, _
 8        ByVal e As System.EventArgs) Handles Me.Init
 9        ' display the server's current time in timeLabel
10        timeLabel.Text = DateTime.Now.ToString("hh:mm:ss")
11     End Sub ' Page_Init
12  End Class ' WebTime
```



http://localhost/WebTime/WebTime.aspx

Current time on the Web server:

13:16:05

Fig. 25.2 | Code-behind file for a page that displays the web server's time.

25.2.3 Relationship Between an ASPX File and a Code-Behind File

How are the ASPX and code-behind files used to create the web page that is sent to the client? First, recall that class WebTime is the base class specified in line 4 of the ASPX file (Fig. 25.1). This class (partially declared in the code-behind file) inherits from **Page**, which defines general web page functionality. Partial class WebTime inherits this functionality and defines some of its own (i.e., displaying the current time). The code in the code-behind file displays the time, whereas the code in the ASPX file defines the GUI.

When a client requests an ASPX file, ASP.NET creates two partial classes behind the scenes. The code-behind file contains one partial class named WebTime and ASP.NET generate another partial class containing the remainder of class WebTime, based on the markup in the ASPX file. For example, WebTime.aspx contains a **Label** web control with ID timeLabel, so the generated partial class would contain a declaration for a variable named timeLabel of type System.Web.UI.WebControls.Label. Class Label represents a web control for displaying text. It is defined in namespace **System.Web.UI.WebControls**, which contains web controls for designing a page's user interface. Web controls in this namespace derive from class **WebControl**. When compiled, the partial class that declares timeLabel combines with the code-behind file's partial class declaration to form the complete WebTime class. This explains why line 10 in Fig. 25.2 can access timeLabel, which is created in lines 19–21 of WebTime.aspx (Fig. 25.1)—method Page_Init and control timeLabel are actually members of the same class, but defined in separate partial classes.

The partial class generated by ASP.NET is based on the ASPX file that defines the page's visual representation. This partial class is combined with the one in Fig. 25.2, which defines the page's logic. The first time the web page is requested, this class is compiled and an instance is created. This instance represents the page and creates the XHTML that is sent to the client. The assembly created from the compiled partial classes is placed in a subdirectory of

```
C:\WINDOWS\Microsoft.NET\Framework\VersionNumber\
   Temporary ASP.NET Files\WebTime
```

where *VersionNumber* is the version number of the .NET Framework (e.g., v2.0.50727) installed on your computer.

Once the web page has been compiled, no recompilation is required on subsequent requeses. New instances of the web page class will be created to serve each request. The project will be recompiled only when you modify the application; changes are detected by the runtime environment, and the application is recompiled to reflect the altered content.

25.2.4 How the Code in an ASP.NET Web Page Executes

Let's look briefly at how the code for our web page executes. When an instance of the page is created, the **PreInit** event occurs first, invoking method **Page_PreInit**, which can be used to set a page's theme and look-and-feel (and perform other tasks that are beyond this chapter's scope). The Init event occurs next, invoking method Page_Init. Method Page_Init is used to initialize objects and other aspects of the page. After Page_Init executes, the **Load** event occurs, and the **Page_Load** event handler executes. Although not present in this example, the PreInit and Load events are inherited from class Page. You will see examples of the Page_Load event handler later in the chapter. After the Load event handler finishes executing, the page processes events that are generated by the page's controls, such as user interactions with the GUI. When the user's request is considered fully processed, an **Unload** event occurs, which calls the **Page_Unload** event handler. This event, too, is inherited from class Page. Page_Unload typically contains code that releases resources used by the page. Other events occur as well, but are typically used only by ASP.NET controls to generate XHTML to render client-side controls. You can learn more about a Page's event life cycle at msdn2.microsoft.com/en-US/library/ms178472.aspx.

25.2.5 Examining the XHTML Generated by an ASP.NET Application

Figure 25.3 shows the XHTML generated by ASP.NET when a client browser requests WebTime.aspx (Fig. 25.1). To view this code, select **View > Source** in Internet Explorer. We added the comments in lines 1–2 and reformatted the XHTML for readability.

The markup in this page is similar to the ASPX file. Lines 7–9 define a document header comparable to that in Fig. 25.1. Lines 10–25 define the document's body. Line 11 begins the form, a mechanism for collecting user information and sending it to the web server. In this particular program, the user does not submit data to the web server for processing; however, processing user data is a crucial part of many applications that is facilitated by forms. We demonstrate how to submit form data to the server in later examples.

XHTML forms can contain visual and nonvisual components. Visual components include buttons and other GUI components with which users interact. Nonvisual components, called **hidden inputs**, store data, such as e-mail addresses, that the document author specifies. A hidden input is defined in lines 13–14. We discuss the precise meaning of this

```
1  [illegible]
2  [illegible] is loaded. -->
3  [illegible]/EN"
4     "http://[illegible]1.dtd">
5
6  [illegible]
7  [illegible]
8     [illegible]
9  [illegible]
10 [illegible]
11    [illegible]WebTime.aspx" id="form1">
12       [illegible]
13          <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value=
14             "/wEPDwUJODExMDE5NzY5ZGSzVbs789nqEeoNueQCnCJQEUgykw==" />
15       [illegible]
16
17       [illegible]
18          [illegible]</h2>
19       [illegible]
20          <span id="timeLabel" style="color:Yellow;
21             background-color:Black;font-size:XX-Large;">13:51:12</span>
22       [illegible]
23       </div>
24    </form>
25 [illegible]
```

Fig. 25.3 | XHTML response when the browser requests WebTime.aspx.

hidden input later in the chapter. Attribute **method** of the form element (line 11) specifies the method by which the web browser submits the form to the server. The **action** attribute identifies the name and location of the resource that will be requested when this form is submitted—in this case, `WebTime.aspx`. Recall that the ASPX file's `form` element contained the `runat="server"` attribute–value pair (line 14 of Fig. 25.1). When the `form` is processed on the server, the `runat` attribute is removed. The `method` and `action` attributes are added, and the resulting XHTML `form` is sent to the client browser.

In the ASPX file, the form's `Label` (i.e., `timeLabel`) is a web control. Here, we are viewing the XHTML created by our application, so the `form` contains a `span` element (lines 20–21 of Fig. 25.3) to represent the text in the label. In this particular case, ASP.NET maps the `Label` web control to an XHTML `span` element. The formatting options that were specified as properties of `timeLabel`, such as the font size and color of the text in the `Label`, are now specified in the `style` attribute of the `span` element.

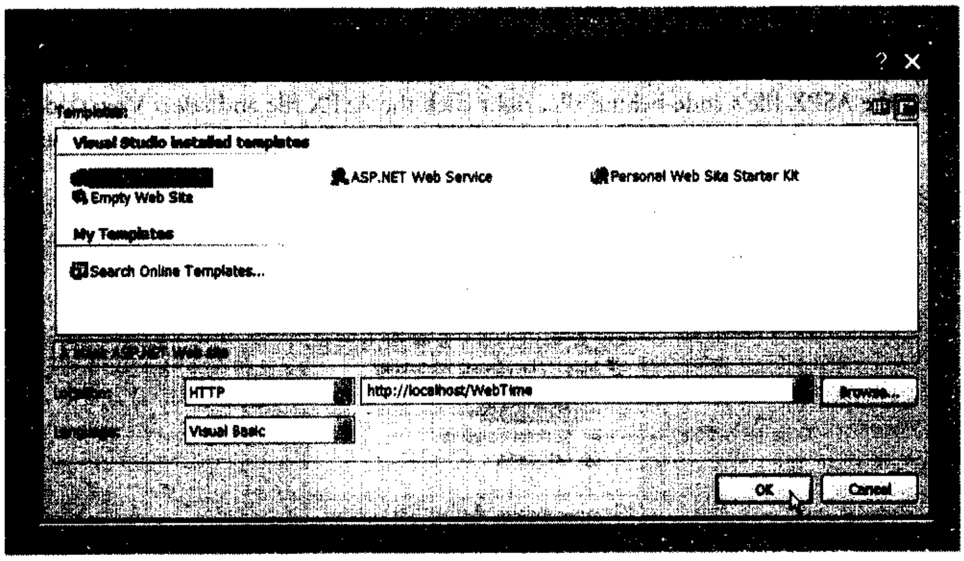
Notice that only those elements in the ASPX file marked with the `runat="server"` attribute–value pair or specified as web controls are modified or replaced when the file is processed by the server. The pure XHTML elements, such as the `h2` in line 18, are sent to the browser as they appear in the ASPX file.

25.2.6 Building an ASP.NET Web Application

Now that we have presented the ASPX file, the code-behind file and the resulting web page sent to the web browser, we show the steps we used to create this application in Visual Web Developer.

Step 1: Creating the Website

In Visual Web Developer, select **File > New Web Site...** to display the **New Web Site** dialog (Fig. 25.4). In this dialog, select **ASP.NET Web Site** in the **Templates** pane. Below this pane, there are two fields in which you can specify the type and location of the web


Templates:
Visual Studio installed templates
ASP.NET Web Service
Personal Web Site Starter Kit
Empty Web Site
My Templates
Search Online Templates...
Browse...
OK
Cancel

HTTP
http://localhost/WebTime
Visual Basic

Fig. 25.4 | Creating an **ASP.NET Web Site** in Visual Web Developer.

application you are creating. If it is not already selected, select **HTTP** from the drop-down list closest to **Location**. This indicates that the web application should be configured to run as an IIS application using HTTP (either on your computer or on a remote computer). We want our project to be located in `http://localhost`, which is the URL for IIS's root dir- ectory (this URL normally corresponds to the `C:\InetPub\wwwroot` directory on your machine). The name **`localhost`** indicates that the server resides on local computer. If the web server were located on a different computer, `localhost` would be replaced with the appropriate IP address or hostname. By default, Visual Web Developer sets the location where the website will be created to `http://localhost/WebSite`, which we change to `http://localhost/WebTime`.

If you do not have IIS on your computer or do not have permission to access it, you can select **File System** from the drop-down list next to **Location** to create the web application in a folder on your computer. You will be able to test the application using Visual Web Developer's internal ASP.NET Development Server, but you will not be able to access the application remotely over the Internet.

The **Language** drop-down list in the **New Web Site** dialog allows you to specify the language (i.e., Visual Basic, Visual C# or Visual J#) in which you will write the code-behind file(s) for the web application. Change the setting to Visual Basic. Click **OK** to create the website. This creates the directory `C:\Inetpub\wwwroot\WebTime` (in IIS) and makes it accessible through the URL `http://localhost/WebTime`. This action also creates a `WebTime` directory in the directory `My Documents\Visual Studio 2005\Projects` in which the project's solution files (e.g., `WebTime.sln`) are stored.

Step 2: Examining the **Solution Explorer** *of the Newly Created Project*

The next several figures describe the new project's content, beginning with the **Solution Explorer** shown in Fig. 25.5. Like Visual Basic 2005 Express, Visual Web Developer creates several files when you create a new project. It creates an ASPX file (i.e., Web Form) named `Default.aspx` for each new **ASP.NET Web Site** project. This file is open by default in the Web Forms Designer in **Source** mode when the project first loads (we discuss this momentarily). As mentioned previously, a code-behind file is included as part of the project. Visual Web Developer creates a code-behind file named **`Default.aspx.vb`**. To open the ASPX file's code-behind file, right click the ASPX file and select **View Code** or click the **View Code** button (▤) at the top of the **Solution Explorer**. Alternatively, you can

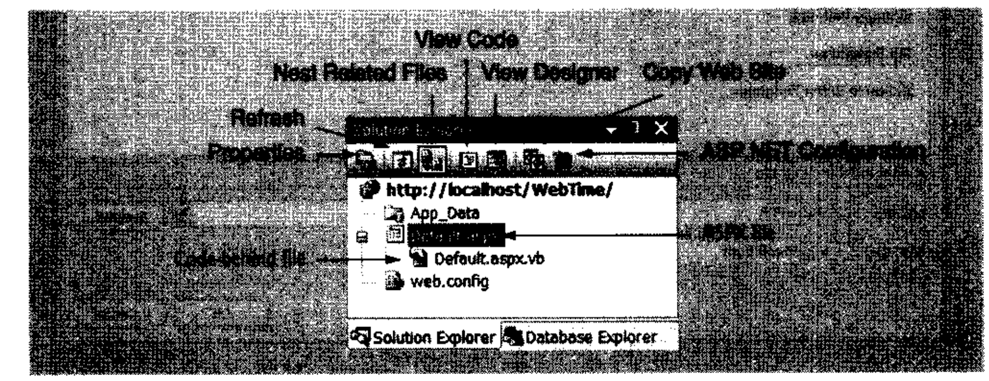


Fig. 25.5 | **Solution Explorer** window for project `WebTime`.

expand the node for the ASPX file to reveal the node for the code-behind file (see Fig. 25.5). You can also choose to list all the files in the project individually (instead of nested) by clicking the **Nest Related Files** button—this option is turned on by default, so clicking the button toggles the option off.

The **Properties** and **Refresh** buttons in Visual Web Developer's **Solution Explorer** behave like those in Visual Basic 2005 Express. Visual Web Developer's **Solution Explorer** also contains the buttons **View Designer**, **Copy Web Site** and **ASP.NET Configuration**. The **View Designer** button allows you to open the Web Form in **Design** mode, which we discuss shortly. The **Copy Web Site** button opens a dialog that allows you to move the files in this project to another location, such as a remote web server. This is useful if you are developing the application on your local computer, but want to make it available to the public from a different location. Finally, the **ASP.NET Configuration** button takes you to a web page called the **Web Site Administration Tool**, where you can manipulate various settings and security options for your application. We discuss this tool in greater detail in Section 25.6.

Step 3: Examining the Toolbox *in Visual Web Developer*

Figure 25.6 shows the **Toolbox** displayed in the IDE when the project loads. Figure 25.6(a) displays the beginning of the **Standard** list of web controls, and Fig. 25.6(b) displays the remaining web controls, and the list of **Data** controls used in ASP.NET. We discuss specific controls in Fig. 25.6 as they are used throughout the chapter. Notice that some controls in the **Toolbox** are similar to Windows controls.

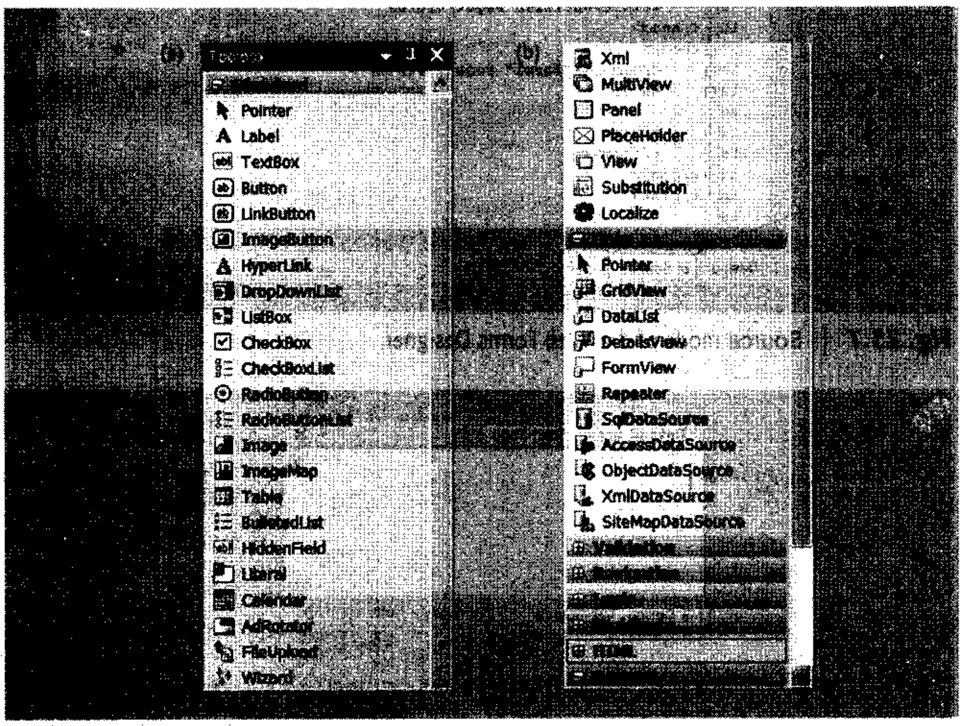
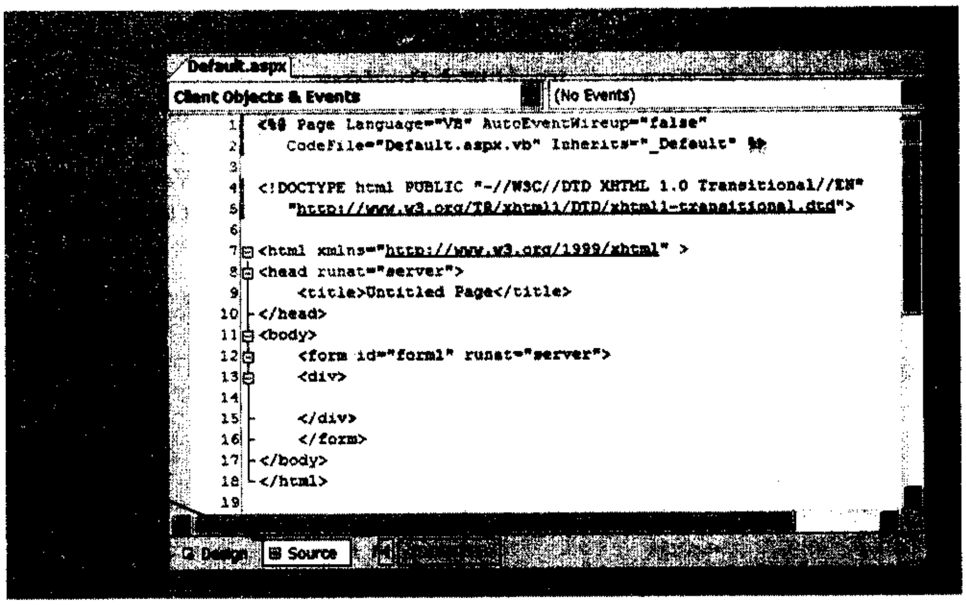


Fig. 25.6 | **Toolbox** in Visual Web Developer.

Step 4: Examining the Web Forms Designer

Figure 25.7 shows the Web Forms Designer in **Source** mode, which appears in the center of the IDE. When the project loads for the first time, the Web Forms Designer displays the autogenerated ASPX file (i.e., `Default.aspx`) in **Source** mode, which allows you to view and edit the markup that comprises the web page. The markup listed in Fig. 25.7 was created by the IDE and serves as a template that we will modify shortly. Clicking the **Design** button in the lower-left corner of the Web Forms Designer switches to **Design** mode (Fig. 25.8), which allows you to drag and drop controls from the **Toolbox** onto the Web Form and see the controls. You can also type at the current cursor location to add text to the web page. We demonstrate this shortly. In response to such actions, the IDE generates the appropriate markup in the ASPX file. Notice that **Design** mode indicates the XHTML element where the cursor is currently located. Clicking the **Source** button returns the Web Forms Designer to **Source** mode, where you can see the generated markup.



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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

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

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

Fig. 25.7 | **Source** mode of the Web Forms Designer.

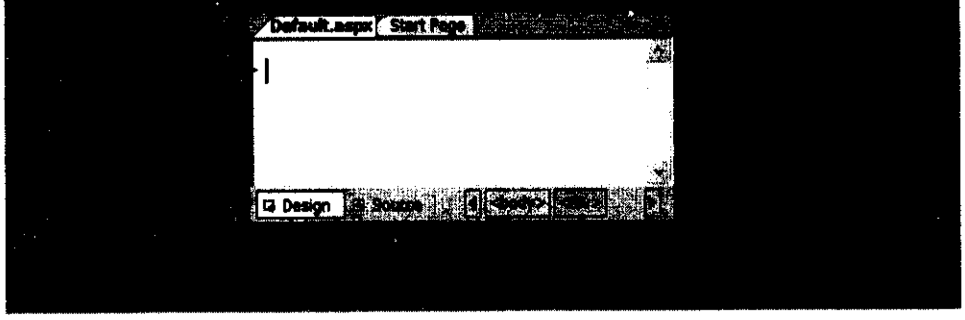
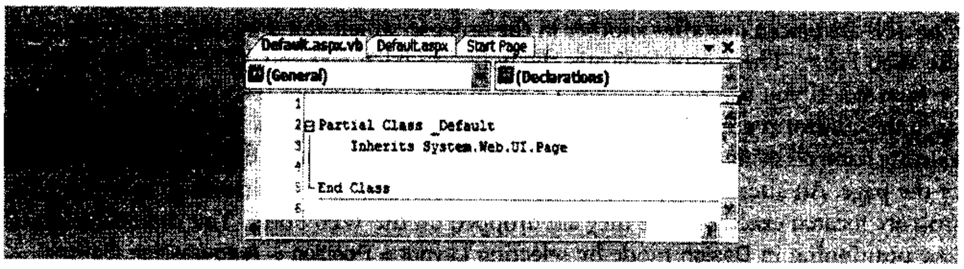


Fig. 25.8 | **Design** mode of the Web Forms Designer.

Step 5: Examining the Code-Behind File in the IDE
The next figure (Fig. 25.9) displays `Default.aspx.vb`—the code-behind file generated by Visual Web Developer for `Default.aspx`. Right click the ASPX file in the **Solution Explorer** and select **View Code** to open the code-behind file. When it is first created, this file contains nothing more than a partial class declaration. We will add the `Page_Init` event handler to this code momentarily.



```
1
2  Partial Class _Default
3      Inherits System.Web.UI.Page
4
5  End Class
6
```

Fig. 25.9 | Code-behind file for `Default.aspx` generated by Visual Web Developer.

Step 6: Renaming the ASPX File
Now that you've seen the contents of the default ASPX and code-behind files, let's rename these files. Right click the ASPX file in the **Solution Explorer** and select **Rename**. Enter the new filename `WebTime.aspx` and press *Enter*. This updates the name of both the ASPX file and the code-behind file. The IDE also updates the `Page` directive's `CodeFile` attribute in `WebTime.aspx`.

Step 7: Renaming the Class in the Code-Behind File and Updating the ASPX File
Although renaming the ASPX file causes the name of the code-behind file to change, this action does not affect the name of the partial class declared in the code-behind file. Open the code-behind file and change the class name from `_Default` (line 2 in Fig. 25.9) to `WebTime`, so the partial class declaration appears as in line 3 of Fig. 25.2. Recall that this class is also referenced by the `Page` directive in the ASPX file. Using the Web Forms Designer's **Source** mode, modify the `Inherits` attribute of the `Page` directive in `WebTime.aspx`, so it appears as in line 4 of Fig. 25.1. The value of the `Inherits` attribute and the class name in the code-behind file must be identical; otherwise, you'll get errors when you build the web application.

Step 8: Changing the Title of the Page
Before designing the content of the Web Form, we change its title from the default `Untitled Page` (line 9 of Fig. 25.7) to `A Simple Web Form Example`. To do so, open the ASPX file in **Source** mode and modify the text in the `title` element—i.e., the text between the tags `<title>` and `</title>`. Alternatively, you can open the ASPX file in **Design** mode and modify the Web Form's `Title` property in the **Properties** window. To view the Web Form's properties, select `DOCUMENT` from the drop-down list in the **Properties** window; **DOCUMENT** represents the Web Form in the **Properties** window.

Step 9: Designing the Page
Designing a Web Form is as simple as designing a Windows Form. To add controls to the page, drag-and-drop them from the **Toolbox** onto the Web Form in **Design** mode. Like the

Web Form itself, each control is an object that has properties, methods and events. You can set these properties and events visually using the **Properties** window or programmatically in the code-behind file. However, unlike working with a Windows Form, you can type text directly on a Web Form at the cursor location or insert XHTML elements from the **Toolbox**.

Controls and other elements are placed sequentially on a Web Form, much as text and images are placed in a document using word-processing software like Microsoft Word. Controls are placed one after another in the order in which you drag-and-drop them onto the Web Form. The cursor indicates the point at which text and XHTML elements will be inserted. If you want to position a control between existing text or controls, you can drop the control at a specific position within the existing elements. You can also rearrange existing controls using drag-and-drop actions. By default, controls flow based on the width of the page. An alternate type of layout is known as **absolute positioning**, in which controls are located exactly where they are dropped on the Web Form. You can enable absolute positioning in **Design** mode by selecting **Layout > Position > Auto-position Options...**, clicking the first checkbox in the **Positioning options** pane of the **Options** dialog that appears, then selecting the appropriate positioning option from the drop-down menu.

Portability Tip 25.2

Absolute positioning is discouraged, because pages designed in this manner may not render correctly on computers with different screen resolutions and font sizes. This could cause absolutely positioned elements to overlap each other or display off-screen, requiring the client to scroll to see the full page content.

In this example, we use one piece of text and one `Label`. To add the text to the Web Form, click the blank Web Form in **Design** mode and type `Current time on the Web server:`. Visual Web Developer is a WYSIWYG (What You See Is What You Get) editor—whenever you make a change to a Web Form in **Design** mode, the IDE creates the markup (visible in **Source** mode) necessary to achieve the desired visual effects seen in **Design** mode. After adding the text to the Web Form, switch to **Source** mode. You should see that the IDE added this text to the `div` element that appears in the ASPX file by default. Back in **Design** mode, highlight the text you added. From the **Block Format** drop-down list (see Fig. 25.10), choose **Heading 2** to format this text as a heading that will appear bold in a font slightly larger than the default. This action encloses the text in an `h2` element. Finally, click to the right of the text and press the *Enter* key to start a new paragraph. This action generates a `p` (paragraph) element in the ASPX file's markup. The IDE should now look like Fig. 25.10.

You can place a `Label` on a Web Form either by dragging-and-dropping or by double clicking the **Toolbox**'s **Label** control. Ensure that the cursor is in the new paragraph, then add a `Label` that will be used to display the time. Using the **Properties** window, set the `(ID)` property of the `Label` to `timeLabel`. In the `Text` property, delete `timeLabel`'s text—this text will be set programmatically in the code-behind file. When a `Label` does not contain text, its name is displayed in square brackets in the Web Forms Designer (Fig. 25.11) as a placeholder for design and layout purposes. This text is not displayed at execution time. We set `timeLabel`'s `BackColor`, `ForeColor` and `Font-Size` properties to `Black`, `Yellow` and `XX-Large`, respectively. To change the `Label`'s font properties, select the `Label`, expand the `Font` node in the **Properties** window and change each relevant property.

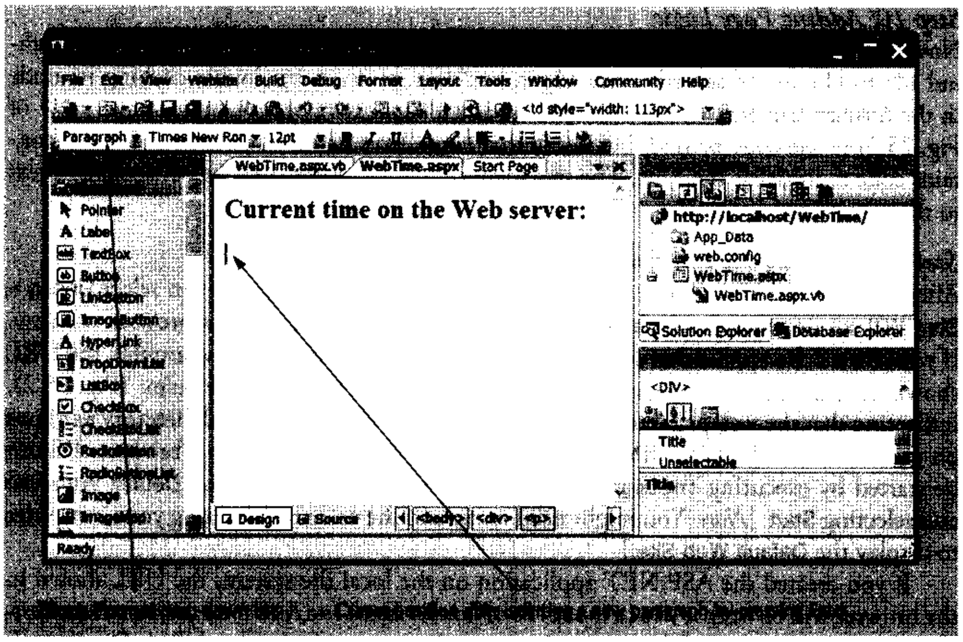


Fig. 25.10 | WebTime.aspx after inserting text and a new paragraph.

As the Label's properties are set, Visual Web Developer updates the ASPX file's contents. Figure 25.11 shows the IDE after setting these properties.

Next, set the Label's EnableViewState property to False. Finally, select DOCUMENT from the drop-down list in the **Properties** window and set the Web Form's EnableSessionState property to False. We discuss both of these properties later in the chapter.

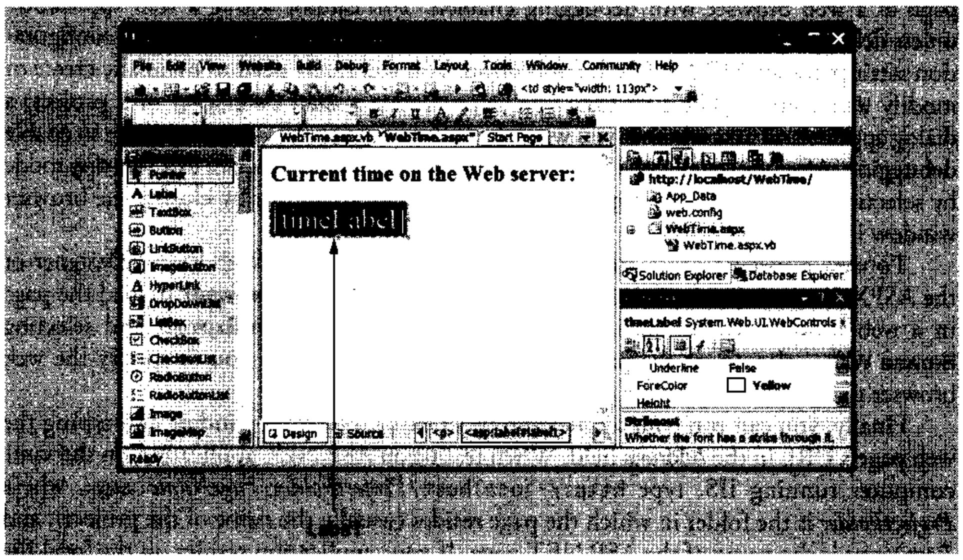


Fig. 25.11 | WebTime.aspx after adding a Label and setting its properties.

Step 10: Adding Page Logic
Now that you've designed the user interface, you'll add Visual Basic code to the code-behind file to obtain the server's time. Open `WebTime.aspx.vb` by double clicking its node in the **Solution Explorer**. In this example, we add a `Page_Init` event handler (lines 7–11 of Fig. 25.2) to the code-behind file. Recall that `Page_Init` handles the `Init` event and contains code to initialize the page. The statement in line 10 of Fig. 25.2 sets `timeLabel`'s text to the server's current time.

Step 11: Running the Program
After creating the Web Form, you can view it several ways. First, you can select **Debug > Start Without Debugging**, which runs the application by opening it in a browser window. If you created the application on your local IIS server (as we did in this example), the URL shown in the browser will be `http://localhost/WebTime/WebTime.aspx` (Fig. 25.2), indicating that the web page (the ASPX file) is located within the virtual directory `WebTime` on the local IIS web server. IIS must be running to test the website in a browser. IIS can be started by executing `inetmgr.exe` from **Start > Run...**, right clicking **Default Web Site** and selecting **Start**. [*Note:* You might need to expand the node representing your computer to display the **Default Web Site**.]

If you created the ASP.NET application on the local file system, the URL shown in the browser will be `http://localhost:`*PortNumber*`/WebTime/WebTime.aspx`, where *PortNumber* is the number of the randomly assigned port on which Visual Web Developer's built-in test server runs. The IDE assigns the port number on a per-solution basis. This URL indicates that the `WebTime` project folder is being accessed through the root directory of the test server running at `localhost:`*PortNumber*. When you select **Debug > Start Without Debugging**, a tray icon appears near the bottom-right of your screen next to the computer's date and time to show that the **ASP.NET Development Server** is running. The test server stops when you exit Visual Web Developer.

To debug your application, you can select **Debug > Start Debugging** to view the web page in a web browser with debugging enabled. You cannot debug a web application unless debugging is explicitly enabled by the **`Web.config`** file—a file that stores configuration settings for an ASP.NET web application. You will rarely need to manually create or modify `Web.config`. The first time you select **Debug > Start Debugging** in a project, a dialog appears and asks whether you want the IDE to modify the `Web.config` file to enable debugging. After you click **OK**, the IDE enters **Running** mode. You can exit **Running** mode by selecting **Debug > Stop Debugging** in Visual Web Developer or by closing the browser window in which the ASPX file is displayed.

To view a specific ASPX file, you can right click either the Web Forms Designer or the ASPX filename (in the **Solution Explorer**) and select **View In Browser** to load the page in a web browser. Right clicking the ASPX file in the **Solution Explorer** and selecting **Browse With...** also opens the page in a browser, but first allows you to specify the web browser that should display the page and its screen resolution.

Finally, you can run your application by opening a browser window and typing the web page's URL in the **Address** field. When testing an ASP.NET application on the same computer running IIS, type `http://localhost/`*ProjectFolder*`/`*PageName*`.aspx`, where *ProjectFolder* is the folder in which the page resides (usually the name of the project), and *PageName* is the name of the ASP.NET page. If your application resides on the local file system, you must first start the **ASP.NET Development Server** by running the application using one of the methods described above. Then you can type the URL (including the *PortNumber* found in the test server's tray icon) in the browser to execute the application.

Note that all of these methods of running the application compile the project for you. In fact, ASP.NET compiles your web page whenever it changes between HTTP requests. For example, suppose you browse the page, then modify the ASPX file or add code to the code-behind file. When you reload the page, ASP.NET recompiles the page on the server before returning the HTTP response to the browser. This important new behavior of ASP.NET 2.0 ensures that clients always see the latest version of the page. You can manually compile a web page or an entire website by selecting **Build Page** or **Build Site**, respectively, from the **Build** menu in Visual Web Developer.

Windows Firewall Settings
If you would like to test your web application over a network, you may need to change your Windows Firewall settings. For security reasons, Windows Firewall does not allow remote access to a web server on your local computer by default. To change this, open the Windows Firewall utility in the Windows Control Panel. In Windows XP, Click the **Advanced** tab and select your network connection from the **Network Connection Settings** list, then click **Settings...**. On the **Services** tab of the **Advanced Settings** dialog, ensure that **Web Server (HTTP)** is checked. In Windows Vista click the **Change settings** link, then click **Continue** in dialog that appears. Select the **Exceptions** tab and place a check next to **World Wide Web Services (HTTP)**.

25.3 Web Controls

This section introduces some of the web controls located in the **Standard** section of the **Toolbox** (Fig. 25.6). Figure 25.12 summarizes some of the web controls used in the chapter examples.

`Label`	Displays text that the user cannot edit.
`TextBox`	Gathers user input and displays text.
`Button`	Triggers an event when clicked.
`HyperLink`	Displays a hyperlink.
`DropDownList`	Displays a drop-down list of choices from which you can select an item.
`RadioButtonList`	Groups radio buttons.
`Image`	Displays images (e.g., GIF and JPG).

Fig. 25.12 | Commonly used web controls.

25.3.1 Text and Graphics Controls

Figure 25.13 depicts a simple form for gathering user input. This example uses all the controls listed in Fig. 25.12, except `Label`, which you used in Section 25.2. The code in

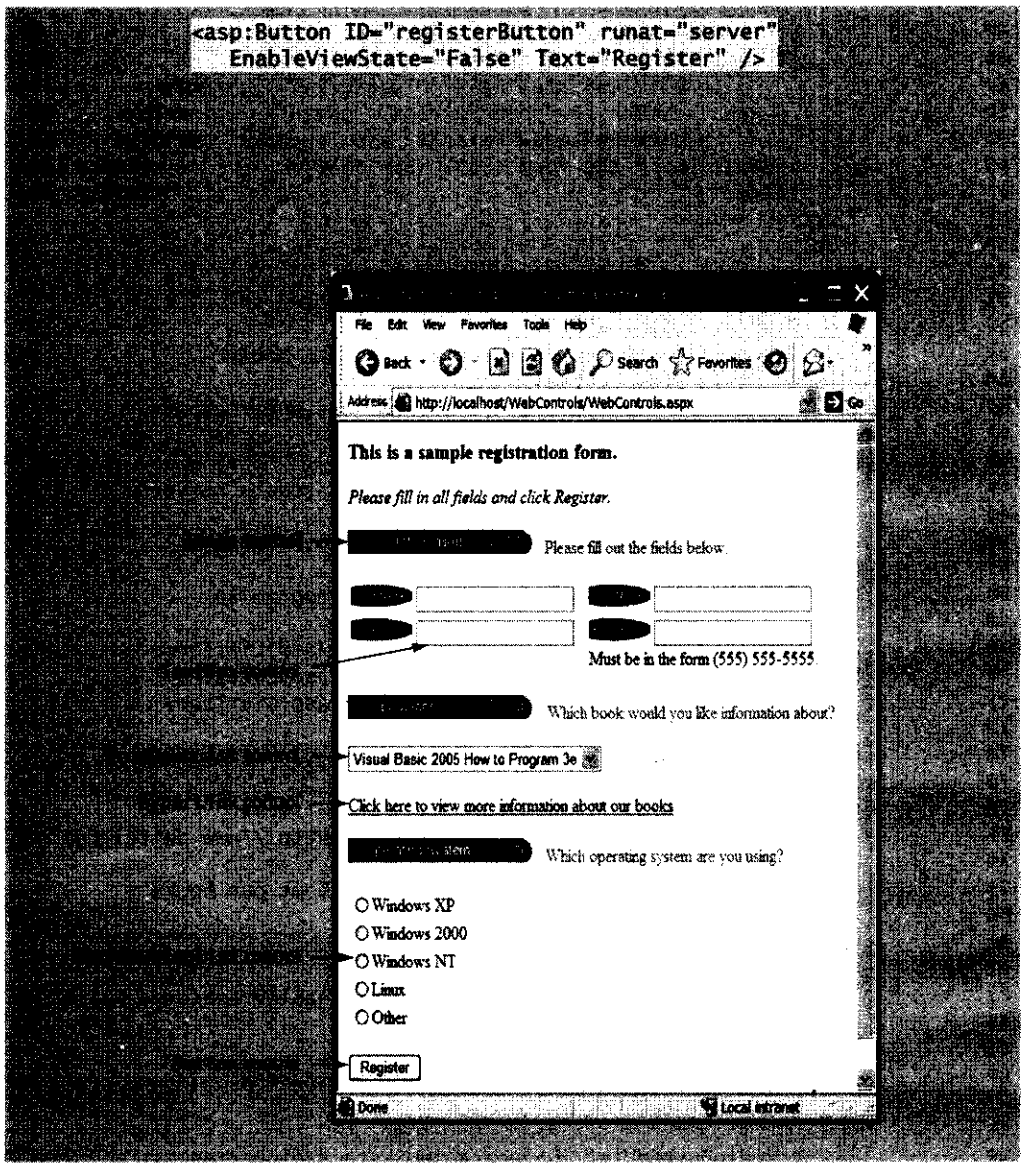
Fig. 25.13 was generated by Visual Web Developer in response to dragging controls onto the page in **Design** mode. To begin, create an ASP.NET website named `WebControls`. [*Note:* This example does not contain any functionality—i.e., no action occurs when the user clicks **Register**. We ask you to provide the functionality as an exercise. In subsequent examples, we demonstrate how to add functionality to many of these web controls.]

```
<%-- Fig. 25.13: WebControls.aspx --%>
<%-- Registration form that demonstrates web controls. --%>
<%@ Page Language="VB" AutoEventWireup="false"
   CodeFile="WebControls.aspx.vb" Inherits="WebControls" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Web Controls Demonstration</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h3>This is a sample registration form.</h3>
      <p>
         <em>Please fill in all fields and click Register.</em></p>
      <p>
         <asp:Image ID="userInformationImage" runat="server"
            EnableViewState="False" ImageUrl="~/Images/user.png" /> &nbsp;
         <span style="color: teal">
            Please fill out the fields below.</span>
      </p>
      <table id="TABLE1">
         <tr>
            [illegible]
               [illegible]
                  EnableViewState="False" ImageUrl="~/Images/[illegible]
               <asp:TextBox ID="firstNameTextBox" runat="server"
                  EnableViewState="False"></asp:TextBox>
            </td>
            <td style="width: 231px; height: 21px" valign="top">
               <asp:Image ID="lastNameImage" runat="server"
                  EnableViewState="False" ImageUrl="~/Images/lname.png" />
               <asp:TextBox ID="lastNameTextBox" runat="server"
                  EnableViewState="False"></asp:TextBox>
            </td>
         </tr>
         <tr>
            <td style="width: 230px" valign="top">
               <asp:Image ID="emailImage" runat="server"
                  EnableViewState="False" ImageUrl="~/Images/email.png" />
               <asp:TextBox ID="emailTextBox" runat="server"
                  EnableViewState="False"></asp:TextBox>
            </td>
```

Fig. 25.13 | Web Form that demonstrates web controls. (Part 1 of 3.)

```
[illegible]
<span style="color: teal">
   Which book would you like information about?</span>
</p>
[illegible]
<asp:DropDownList ID="booksDropDownList" runat="server"
   EnableViewState="False">
   <asp:ListItem>Visual Basic 2005 How to Program 3e
       </asp:ListItem>
   <asp:ListItem>Visual C# 2005 How to Program 2e
       </asp:ListItem>
   <asp:ListItem>Java How to Program 6e</asp:ListItem>
   <asp:ListItem>C++ How to Program 5e</asp:ListItem>
   <asp:ListItem>XML How to Program 1e</asp:ListItem>
</asp:DropDownList>
[illegible]
<asp:HyperLink ID="booksHyperLink" runat="server"
   EnableViewState="False" NavigateUrl="http://www.deitel.com"
   Target="_blank">
   Click here to view more information about our books
</asp:HyperLink>
</p>
[illegible]
<asp:Image ID="osImage" runat="server" [illegible]
   ImageUrl="~/Images/os.png" /> &nbsp;
<span style="color: teal">
   Which operating system are you using?</span>
</p>
[illegible]
<asp:RadioButtonList ID="operatingSystemRadioButtonList"
   runat="server" EnableViewState="False">
   <asp:ListItem>Windows XP</asp:ListItem>
   <asp:ListItem>Windows 2000</asp:ListItem>
   <asp:ListItem>Windows NT</asp:ListItem>
   <asp:ListItem>Linux</asp:ListItem>
   <asp:ListItem>Other</asp:ListItem>
</asp:RadioButtonList>
</p>
<p>
```

Fig. 25.13 | Web Form that demonstrates web controls. (Part 2 of 3.)



```
<asp:Button ID="registerButton" runat="server"
   EnableViewState="False" Text="Register" />
```

http://localhost/WebControls/WebControls.aspx

This is a sample registration form.

Please fill in all fields and click Register.

Please fill out the fields below.

Must be in the form (555) 555-5555

Which book would you like information about?

Visual Basic 2005 How to Program 3e

Click here to view more information about our books

Which operating system are you using?

- O Windows XP
- O Windows 2000
- O Windows NT
- O Linux
- O Other

Register

Fig. 25.13 | Web Form that demonstrates web controls. (Part 3 of 3.)

Before discussing the web controls used in this ASPX file, we explain the XHTML that creates the layout seen in Fig. 25.13. The page contains an h3 heading element (line 16), followed by a series of additional XHTML blocks. We place most of the web controls inside p elements (i.e., paragraphs), but we use an XHTML table element (lines 25–55) to organize the Image and TextBox controls in the user information section of the page. In the preceding section, we described how to add heading elements and paragraphs visually without manipulating any XHTML in the ASPX file directly. Visual Web Developer allows you to add a table in a similar manner.

Adding an XHTML Table to a Web Form

To create a table with two rows and two columns in **Design** mode, select the **Insert Table** command from the **Layout** menu. In the **Insert Table** dialog that appears, select the **Custom** radio button. In the **Layout** group box, change the values of **Rows** and **Columns** to 2. By default, the contents of a table cell are aligned vertically in the middle of the cell. We changed the vertical alignment of all cells in the table by clicking the **Cell Properties...** button, then selecting **top** from the **Vertical align** combo box in the resulting dialog. This causes the content of each table cell to align with the top of the cell. Click **OK** to close the **Cell Properties** dialog, then click **OK** to close the **Insert Table** dialog and create the table. Once a table is created, controls and text can be added to particular cells to create a neatly organized layout.

Setting the Color of Text on a Web Form

Notice that some of the instructions to the user on the form appear in a teal color. To set the color of a specific piece of text, highlight the text and select **Format > Foreground color...**. In the **Color Picker** dialog, click the **Named Colors** tab and choose a color. Click **OK** to apply the color. Note that the IDE places the colored text in an XHTML `span` element (e.g., lines 22–23) and applies the color using the `span`'s `style` attribute.

Examining Web Controls on a Sample Registration Form

Lines 20–21 of Fig. 25.13 define an **`Image`** control, which inserts an image into a web page. The images used in this example are located in the chapter's examples directory. You can download the examples from `www.deitel.com/books/iw3htp4`. Before an image can be displayed on a web page using an `Image` web control, the image must first be added to the project. We added an `Images` folder to this project (and to each example project in the chapter that uses images) by right clicking the location of the project in the **Solution Explorer**, selecting **New Folder** and entering the folder name `Images`. We then added each of the images used in the example to this folder by right clicking the folder, selecting **Add Existing Item...** and browsing for the files to add. You can also drag a folder full of images onto the project's location in the **Solution Explorer** to add the folder and all the images to the project.

The **`ImageUrl`** property (line 21) specifies the location of the image to display in the `Image` control. To select an image, click the ellipsis next to the `ImageUrl` property in the **Properties** window and use the **Select Image** dialog to browse for the desired image in the project's `Images` folder. When the IDE fills in the `ImageUrl` property based on your selection, it includes a tilde and forward slash (`~/`) at the beginning of the `ImageUrl`—this indicates that the `Images` folder is in the root directory of the project.

Lines 25–55 contain the `table` element created by the steps discussed previously. Each `td` element contains an `Image` control and a **`TextBox`** control, which allows you to obtain text from the user and display text to the user. For example, lines 30–31 define a `TextBox` control used to collect the user's first name.

Lines 64–73 define a **`DropDownList`**. This control is similar to the XHTML `select` control. When a user clicks the drop-down list, it expands and displays a list from which the user can make a selection. Each item in the drop-down list is defined by a **`ListItem`** element (lines 66–72). After dragging a `DropDownList` control onto a Web Form, you can add items to it using the **ListItem Collection Editor**. This process is similar to customizing a `ListBox` in a Windows application. In Visual Web Developer, you can access the **ListItem Collection Editor** by clicking the ellipsis next to the `Items` property of the `DropDownList`, or by using the **DropDownList Tasks** menu. You can open this menu by clicking the small arrowhead that appears in the upper-right corner of the control in **Design** mode (Fig. 25.14). This menu is called a **smart tag menu**. Visual Web Developer displays smart tag menus for many ASP.NET controls to facilitate common tasks. Clicking **Edit Items...** in the **DropDownList Tasks** menu opens the **ListItem Collection Editor**, which allows you to add `ListItem` elements to the `DropDownList`.

The **`HyperLink`** control (lines 76–80 of Fig. 25.13) adds a hyperlink to a web page. The **`NavigateUrl`** property (line 77) of this control specifies the resource (i.e., `http://www.deitel.com`) that is requested when a user clicks the hyperlink. Setting the **`Target`** property to `_blank` specifies that the requested web page should open in a new browser window. By default, `HyperLink` controls cause pages to open in the same browser window.

Lines 89–96 define a **`RadioButtonList`** control, which provides a series of radio buttons from which the user can select only one. Like options in a `DropDownList`, individual radio buttons are defined by `ListItem` elements. Note that, like the **DropDownList Tasks** smart tag menu, the **RadioButtonList Tasks** smart tag menu also provides an **Edit Items...** link to open the **ListItem Collection Editor.**

The final web control in Fig. 25.13 is a **`Button`** (lines 99–100). A `Button` web control represents a button that triggers an action when clicked. This control typically maps to an XHTML `input` element with attribute `type` set to `"submit"`. As stated earlier, clicking the **Register** button in this example does not do anything.

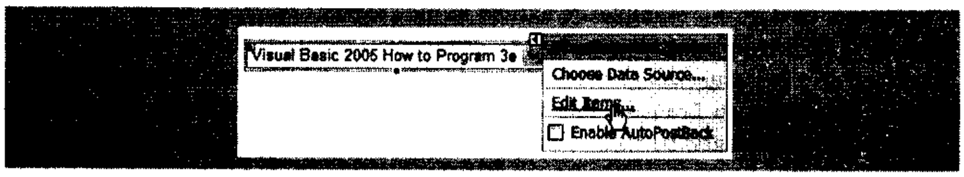


Fig. 25.14 | **DropDownList Tasks** smart tag menu.

25.3.2 AdRotator Control

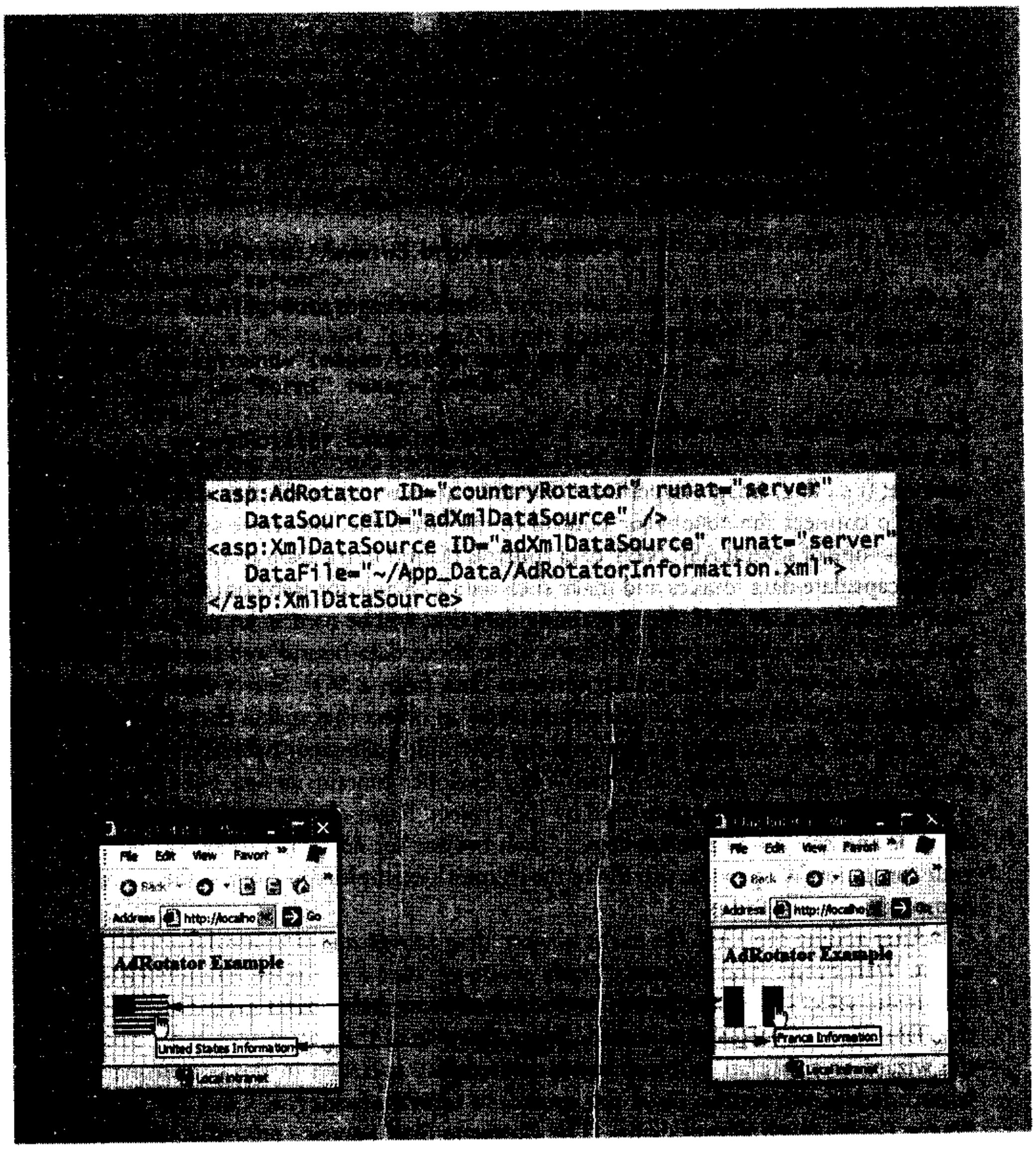
Web pages often contain product or service advertisements, which usually consist of images. Although website authors want to include as many sponsors as possible, web pages can display only a limited number of advertisements. To address this problem, ASP.NET provides the **`AdRotator`** web control for displaying advertisements. Using advertisement data located in an XML file, an `AdRotator` randomly selects an image to display and generates a hyperlink to the web page associated with that image. Browsers that do not support images display alternate text that is specified in the XML document. If a user clicks the image or substituted text, the browser loads the web page associated with that image.

Demonstrating the `AdRotator` Web Control

Figure 25.15 demonstrates the `AdRotator` web control. In this example, the "advertisements" that we rotate are the flags of 10 countries. When a user clicks the displayed flag image, the browser is redirected to a web page containing information about the country that the flag represents. If a user refreshes the browser or requests the page again, one of the 10 flags is again chosen at random and displayed.

The ASPX file in Fig. 25.15 is similar to that in Fig. 25.1. However, instead of XHTML text and a `Label`, this page contains XHTML text (the `h3` element in line 16) and an AdRotator control named countryRotator (lines 18–19). This page also contains an XmlDataSource control (lines 20–22), which supplies the data to the AdRotator control. The background attribute of the page's body element (line 13) is set to the image background.png, located in the project's Images folder. To specify this file, click the ellipsis provided next to the Background property of DOCUMENT in the **Properties** window and use the resulting dialog to select background.png from the Images folder. The images and XML file used in this example are both located in the chapter's examples directory.

You do not need to add any code to the code-behind file, because the AdRotator control "does all the work." The output depicts two different requests. Figure 25.15(a) shows



```
<asp:AdRotator ID="countryRotator" runat="server"
   DataSourceID="adXmlDataSource" />
<asp:XmlDataSource ID="adXmlDataSource" runat="server"
   DataFile="~/App_Data/AdRotatorInformation.xml">
</asp:XmlDataSource>
```

Fig. 25.15 | Web Form that demonstrates the AdRotator web control. (Part 1 of 2.)

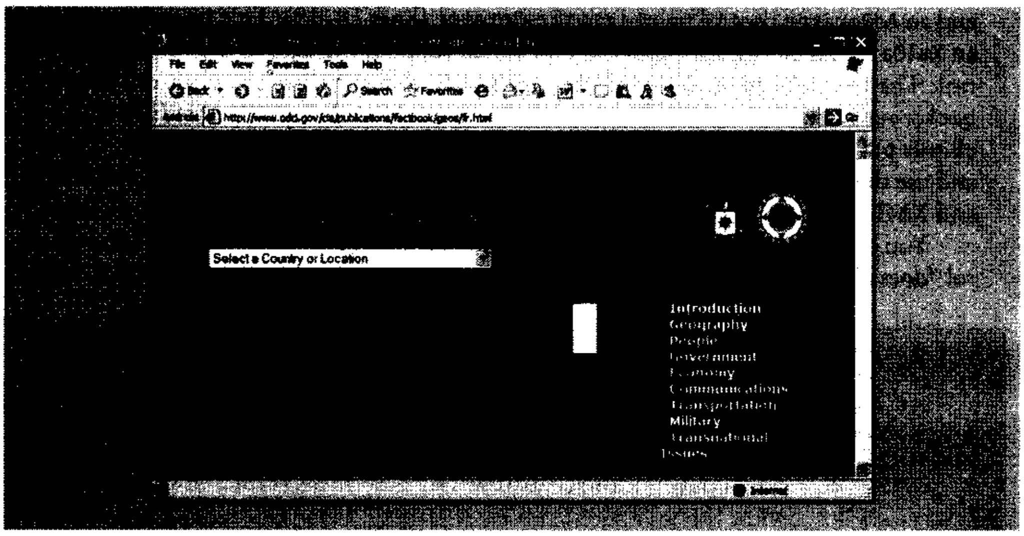


Fig. 25.15 | Web Form that demonstrates the `AdRotator` web control. (Part 2 of 2.)

the first time the page is requested, when the American flag is shown. In the second request, as shown in Fig. 25.15(b), the French flag is displayed. Figure 25.15(c) depicts the web page that loads when the French flag is clicked.

Connecting Data to an ***`AdRotator`*** *Control*
An `AdRotator` control accesses an XML file (presented shortly) to determine what advertisement (i.e., flag) image, hyperlink URL and alternate text to display and include in the page. To connect the `AdRotator` control to the XML file, we create an **`XmlDataSource`** control—one of several ASP.NET data controls (found in the **Data** section of the **Toolbox**) that encapsulate data sources and make such data available for web controls. An `XmlDataSource` references an XML file containing data that will be used in an ASP.NET application. Later in the chapter, you will learn more about data-bound web controls, as well as the `SqlDataSource` control, which retrieves data from a SQL Server database, and the `ObjectDataSource` control, which encapsulates an object that makes data available.

To build this example, we first add the XML file `AdRotatorInformation.xml` to the project. Each project created in Visual Web Developer contains an `App_Data` folder, which is intended to store all the data used by the project. Right click this folder in the **Solution Explorer** and select **Add Existing Item...**, then browse for `AdRotatorInformation.xml` on your computer. We provide this file in the chapter's examples directory in the subdirectory named `exampleXMLFiles`.

After adding the XML file to the project, drag an `AdRotator` control from the **Toolbox** to the Web Form. The **AdRotator Tasks** smart tag menu will open automatically. From this menu, select **<New Data Source...>** from the **Choose Data Source** drop-down list to start the **Data Source Configuration Wizard**. Select **XML File** as the data-source type. This causes the wizard to create an `XmlDataSource` with the `ID` specified in the bottom half of the wizard dialog. We set the `ID` of the control to `adXmlDataSource`. Click **OK** in the **Data Source Configuration Wizard** dialog. The **Configure Data Source - adXmlDataSource** dialog appears next. In this dialog's **Data File** section, click **Browse...** and, in the **Select XML File** dialog, locate and select the XML file you added to the `App_Data` folder. Click **OK** to exit this dialog, then click **OK** to exit the **Configure Data Source - adXmlDataSource** dialog. After completing these steps, the `AdRotator` is configured to use the XML file to determine which advertisements to display.

Examining an XML File Containing Advertisement Information
XML document `AdRotatorInformation.xml` (Fig. 25.16)—or any XML document used with an `AdRotator` control—must contain one **`Advertisements`** root element (lines 4–94). Within that element can be several **`Ad`** elements (e.g., lines 5–12), each of which provides information about a different advertisement. Element **`ImageUrl`** (line 6) specifies the path (location) of the advertisement's image, and element **`NavigateUrl`** (lines 7–9) specifies the URL for the web page that loads when a user clicks the advertisement. Note that we reformatted this file for presentation purposes. The actual XML file cannot con-tain any whitespace before or after the URL in the `NavigateUrl` element, or the whitespace will be considered part of the URL, and the page will not load properly.

The **`AlternateText`** element (line 10) nested in each `Ad` element contains text that displays in place of the image when the browser cannot locate or render the image for some reason (i.e., the file is missing, or the browser is not capable of displaying it), or to assist the visually impaired. The `AlternateText` element's text is also a tooltip that Internet Explorer displays when a user places the mouse pointer over the image (Fig. 25.15). The **`Impressions`** element (line 11) specifies how often a particular image appears, relative to the other images. An advertisement that has a higher `Impressions` value displays more frequently than an advertisement with a lower value. In our example, the advertisements display with equal probability, because the value of each `Impressions` element is set to `1`.

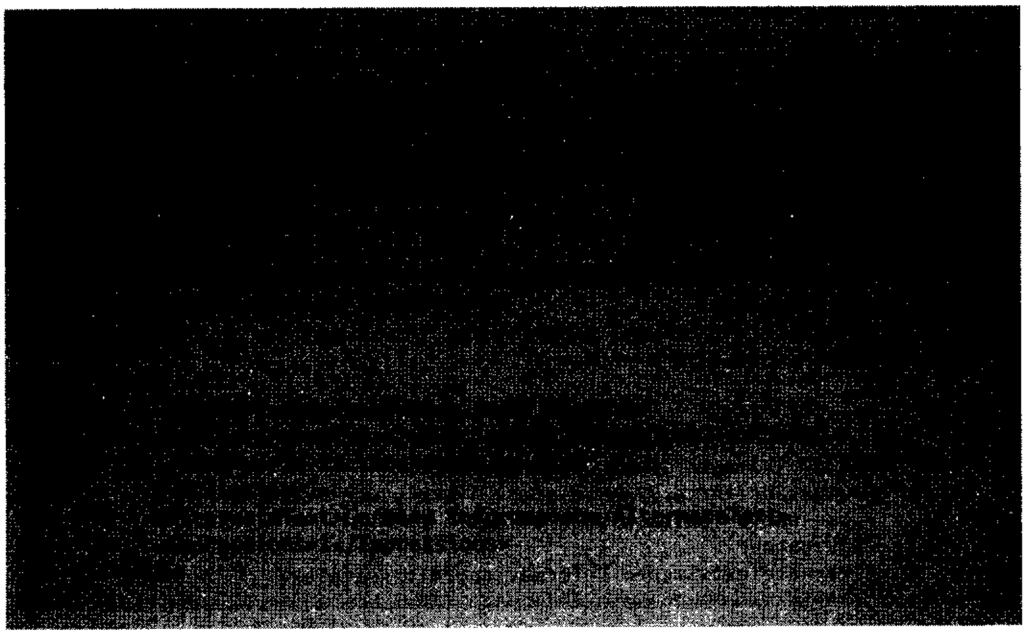


Fig. 25.16 | XML file containing advertisement information used in `AdRotator` example. (Part 1 of 3.)

```
   <Ad>
      <ImageUrl>Images/italy.png</ImageUrl>
      <NavigateUrl>http://www.cia.gov/library/publications/
         the-world-factbook/geos/it.html
      </NavigateUrl>
      <AlternateText>Italy Information</AlternateText>
      <Impressions>1</Impressions>
   </Ad>

   <Ad>
      <ImageUrl>Images/spain.png</ImageUrl>
      <NavigateUrl>http://www.cia.gov/library/publications/
         the-world-factbook/geos/sp.html
      </NavigateUrl>
      <AlternateText>Spain Information</AlternateText>
      <Impressions>1</Impressions>
   </Ad>

   <Ad>
      <ImageUrl>Images/latvia.png</ImageUrl>
      <NavigateUrl>http://www.cia.gov/library/publications/
         the-world-factbook/geos/lg.html
      </NavigateUrl>
      <AlternateText>Latvia Information</AlternateText>
      <Impressions>1</Impressions>
   </Ad>

   <Ad>
      <ImageUrl>Images/peru.png</ImageUrl>
      <NavigateUrl>http://www.cia.gov/library/publications/
         the-world-factbook/geos/pe.html
      </NavigateUrl>
      <AlternateText>Peru Information</AlternateText>
      <Impressions>1</Impressions>
   </Ad>

   <Ad>
      <ImageUrl>Images/senegal.png</ImageUrl>
      <NavigateUrl>http://www.cia.gov/library/publications/
         the-world-factbook/geos/sg.html
      </NavigateUrl>
      <AlternateText>Senegal Information</AlternateText>
      <Impressions>1</Impressions>
   </Ad>

   <Ad>
      <ImageUrl>Images/sweden.png</ImageUrl>
      <NavigateUrl>http://www.cia.gov/library/publications/
         the-world-factbook/geos/sw.html
      </NavigateUrl>
      <AlternateText>Sweden Information</AlternateText>
```

Fig. 25.16 | XML file containing advertisement information used in AdRotator example. (Part 2 of 3.)

```
      <Impressions>1</Impressions>
   </Ad>

   <Ad>
      <ImageUrl>Images/thailand.png</ImageUrl>
      <NavigateUrl>http://www.cia.gov/library/publications/
         the-world-factbook/geos/th.html
      </NavigateUrl>
      <AlternateText>Thailand Information</AlternateText>
      <Impressions>1</Impressions>
   </Ad>

   <Ad>
      <ImageUrl>Images/unitedstates.png</ImageUrl>
      <NavigateUrl>http://www.cia.gov/library/publications/
         the-world-factbook/geos/us.html
      </NavigateUrl>
      <AlternateText>United States Information</AlternateText>
      <Impressions>1</Impressions>
   </Ad>
</Advertisements>
```

Fig. 25.16 | XML file containing advertisement information used in `AdRotator` example. (Part 3 of 3.)

25.3.3 Validation Controls

This section introduces a different type of web control, called a **validation control** (or **validator**), which determines whether the data in another web control is in the proper format. For example, validators could determine whether a user has provided information in a required field or whether a zip-code field contains exactly five digits. Validators provide a mechanism for validating user input on the client. When the XHTML for our page is created, the validator is converted into JavaScript that performs the validation. However, some clients do not support scripting or disable scripting. So, for security reasons, validation is always performed on the server too—whether or not scripting is enabled on the client. For this example, we assume the client has JavaScript enabled.

Validating Input in a Web Form

The example in this section prompts the user to enter a name, e-mail address and phone number. A website could use a form like this to collect contact information from site visitors. After the user enters any data, but before the data is sent to the web server, validators ensure that the user entered a value in each field and that the e-mail address and phone number values are in an acceptable format. In this example, (555) 123-4567, 555-123-4567 and 123-4567 are all considered valid phone numbers. Once the data is submitted, the web server responds by displaying an appropriate message and an XHTML table repeating the submitted information. Note that a real business application would typically store the submitted data in a database or in a file on the server. We simply send the data back to the form to demonstrate that the server received the data.

Figure 25.17 presents the ASPX file. Like the Web Form in Fig. 25.13, this Web Form uses a `table` to organize the page's contents. Lines 24–25, 36–37 and 58–59 define

TextBoxes for retrieving the user's name, e-mail address and phone number, respectively, and line 78 defines a **Submit** button. Lines 80–82 create a Label named outputLabel that displays the response from the server when the user successfully submits the form. Notice that outputLabel's **Visible** property is initially set to False (line 81), so the Label does not appear in the client's browser when the page loads for the first time.

Using RequiredFieldValidator *Controls*

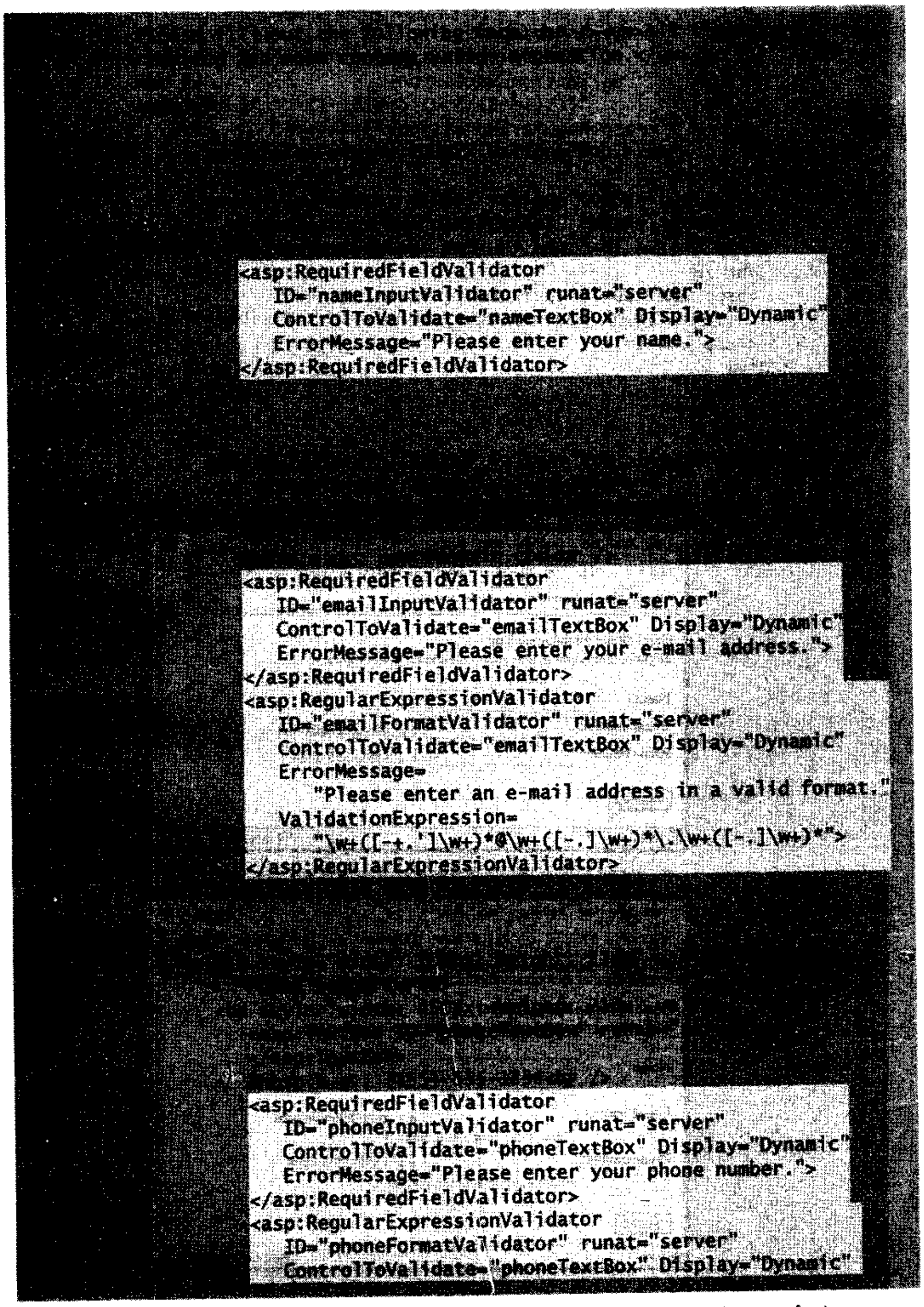
In this example, we use three **RequiredFieldValidator** controls (found in the **Validation** section of the **Toolbox**) to ensure that the name, e-mail address and phone number TextBoxes are not empty when the form is submitted. A RequiredFieldValidator makes an input control a required field. If such a field is empty, validation fails. For example, lines 26–30 define RequiredFieldValidator nameInputValidator, which confirms that nameTextBox is not empty. Line 28 associates nameTextBox with nameInputValidator by setting the validator's **ControlToValidate** property to nameTextBox. This indicates that nameInputValidator verifies the nameTextBox's contents. We set the value of this property (and the validator's other properties) by selecting the validator in **Design** mode and using the **Properties** window to specify property values. Property **ErrorMessage**'s text (line 29) is displayed on the Web Form if the validation fails. If the user does not input any data in nameTextBox and attempts to submit the form, the ErrorMessage text is displayed in red. Because we set the validator's **Display** property to Dynamic (line 28), the validator is displayed on the Web Form only when validation fails. Space is allocated dynamically when validation fails, causing the controls below the validator to shift downward to accommodate the ErrorMessage, as seen in Fig. 25.17(a)–(c).

Using RegularExpressionValidator *Controls*

This example also uses **RegularExpressionValidator** controls to match the e-mail address and phone number entered by the user against regular expressions. These controls determine whether the e-mail address and phone number were each entered in a valid format. For example, lines 44–51 create a RegularExpressionValidator named emailFormatValidator. Line 46 sets property ControlToValidate to emailTextBox to indicate that emailFormatValidator verifies the emailTextBox's contents.

```
 1  [illegible]
 2  [illegible]
 3  [illegible]
 4  [illegible]
 5  [illegible]
 6  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 7     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
 8
 9  <html xmlns="http://www.w3.org/1999/xhtml" >
10  <head runat="server">
11     <title>Demonstrating Validation Controls</title>
12  </head>
13  <body>
14     <form id="form1" runat="server">
15     <div>
```

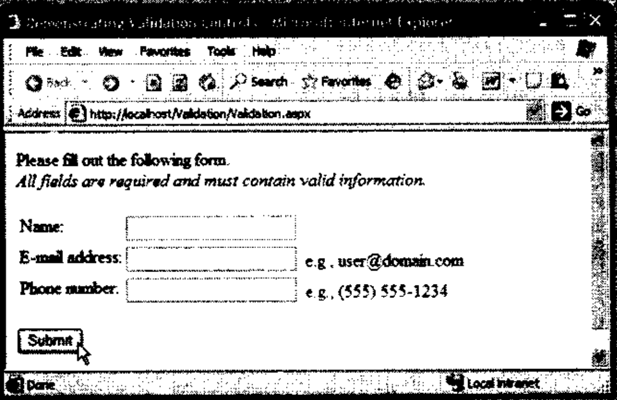
Fig. 25.17 | Form that demonstrates using validators to validate user input. (Part 1 of 4.)



```
[illegible]
<asp:RequiredFieldValidator
   ID="nameInputValidator" runat="server"
   ControlToValidate="nameTextBox" Display="Dynamic"
   ErrorMessage="Please enter your name.">
</asp:RequiredFieldValidator>
[illegible]
<asp:RequiredFieldValidator
   ID="emailInputValidator" runat="server"
   ControlToValidate="emailTextBox" Display="Dynamic"
   ErrorMessage="Please enter your e-mail address.">
</asp:RequiredFieldValidator>
<asp:RegularExpressionValidator
   ID="emailFormatValidator" runat="server"
   ControlToValidate="emailTextBox" Display="Dynamic"
   ErrorMessage=
      "Please enter an e-mail address in a valid format."
   ValidationExpression=
      "\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
[illegible]
<asp:RequiredFieldValidator
   ID="phoneInputValidator" runat="server"
   ControlToValidate="phoneTextBox" Display="Dynamic"
   ErrorMessage="Please enter your phone number.">
</asp:RequiredFieldValidator>
<asp:RegularExpressionValidator
   ID="phoneFormatValidator" runat="server"
   ControlToValidate="phoneTextBox" Display="Dynamic"
```

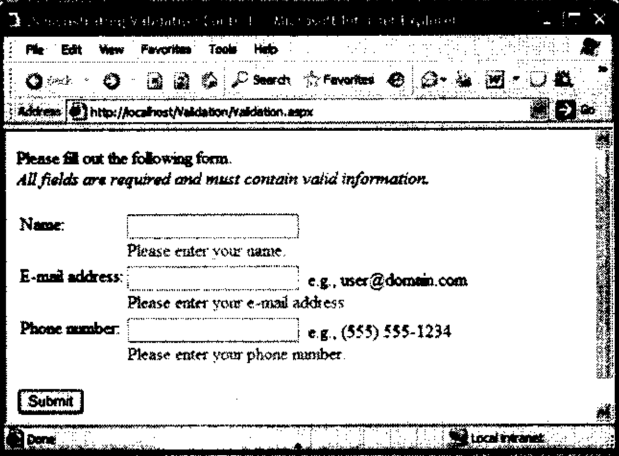
Fig. 25.17 | Form that demonstrates using validators to validate user input. (Part 2 of 4.)

```
   ErrorMessage=
      "Please enter a phone number in a valid format."
   ValidationExpression=
      "((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}">
</asp:RegularExpressionValidator>
[illegible]
[illegible] ID="submitButton" runat="server" Text="Submit" [illegible]
[illegible]
<asp:Label ID="outputLabel" runat="server"
   Text="Thank you for your submission." Visible=[illegible]
</asp:Label>
[illegible]
```



Please fill out the following form.
All fields are required and must contain valid information.

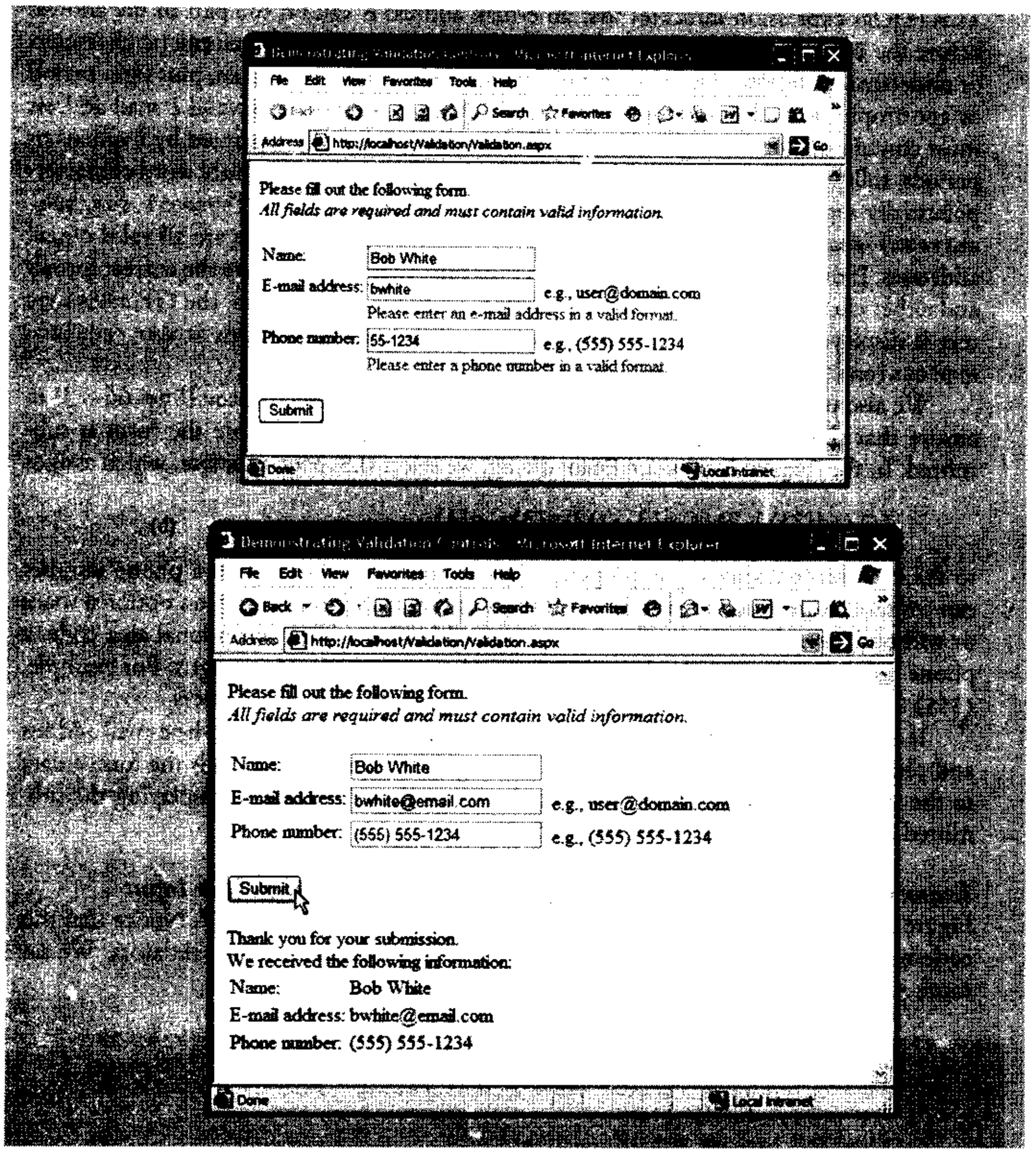
Name:
E-mail address: e.g., user@domain.com
Phone number: e.g., (555) 555-1234



Please fill out the following form.
All fields are required and must contain valid information.

Name:
Please enter your name.
E-mail address: e.g., user@domain.com
Please enter your e-mail address
Phone number: e.g., (555) 555-1234
Please enter your phone number.

Fig. 25.17 | Form that demonstrates using validators to validate user input. (Part 3 of 4.)


Demonstrating Validation Controls - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/Validation/Validation.aspx
Submit
Done
Local intranet

Please fill out the following form.
All fields are required and must contain valid information.

Name: Bob White
E-mail address: bwhite e.g., user@domain.com
Please enter an e-mail address in a valid format.
Phone number: 55-1234 e.g., (555) 555-1234
Please enter a phone number in a valid format.

Please fill out the following form.
All fields are required and must contain valid information.

Name: Bob White
E-mail address: bwhite@email.com e.g., user@domain.com
Phone number: (555) 555-1234 e.g., (555) 555-1234

Thank you for your submission.
We received the following information:
Name: Bob White
E-mail address: bwhite@email.com
Phone number: (555) 555-1234

Fig. 25.17 | Form that demonstrates using validators to validate user input. (Part 4 of 4.)

A `RegularExpressionValidator`'s **`ValidationExpression`** property specifies the regular expression that validates the `ControlToValidate`'s contents. Clicking the ellipsis next to property `ValidationExpression` in the **Properties** window displays the **Regular Expression Editor** dialog, which contains a list of **Standard expressions** for phone numbers, zip codes and other formatted information. You can also write your own custom expression. For the `emailFormatValidator`, we selected the standard expression **Internet e-mail address**, which uses the validation expression

```
\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*
```

This regular expression indicates that an e-mail address is valid if the part of the address before the `@` symbol contains one or more word characters (i.e., alphanumeric characters or underscores), followed by zero or more strings comprised of a hyphen, plus sign, period or apostrophe and additional word characters. After the `@` symbol, a valid e-mail address must contain one or more groups of word characters potentially separated by hyphens or periods, followed by a required period and another group of one or more word characters potentially separated by hyphens or periods. For example, `bob.white@email.com`, `bob-white@my-email.com` and `bob's-personal.email@white.email.com` are all valid e-mail addresses. If the user enters text in the `emailTextBox` that does not have the correct format and either clicks in a different text box or attempts to submit the form, the `ErrorMessage` text is displayed in red. You can learn more about regular expressions at `www.regular-expressions.info`.

We also use `RegularExpressionValidator` `phoneFormatValidator` (lines 66–73) to ensure that the `phoneTextBox` contains a valid phone number before the form is submitted. In the **Regular Expression Editor** dialog, we select **U.S. phone number**, which assigns

```
((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}
```

to the `ValidationExpression` property. This expression indicates that a phone number can contain a three-digit area code either in parentheses and followed by an optional space or without parentheses and followed by required hyphen. After an optional area code, a phone number must contain three digits, a hyphen and another four digits. For example, `(555) 123-4567`, `555-123-4567` and `123-4567` are all valid phone numbers.

If all five validators are successful (i.e., each `TextBox` is filled in, and the e-mail address and phone number provided are valid), clicking the **Submit** button sends the form's data to the server. As shown in Fig. 25.17(d), the server then responds by displaying the submitted data in the `outputLabel` (lines 80–82).

Examining the Code-Behind File for a Web Form That Receives User Input

Figure 25.18 depicts the code-behind file for the ASPX file in Fig. 25.17. Notice that this code-behind file does not contain any implementation related to the validators. We say more about this soon.

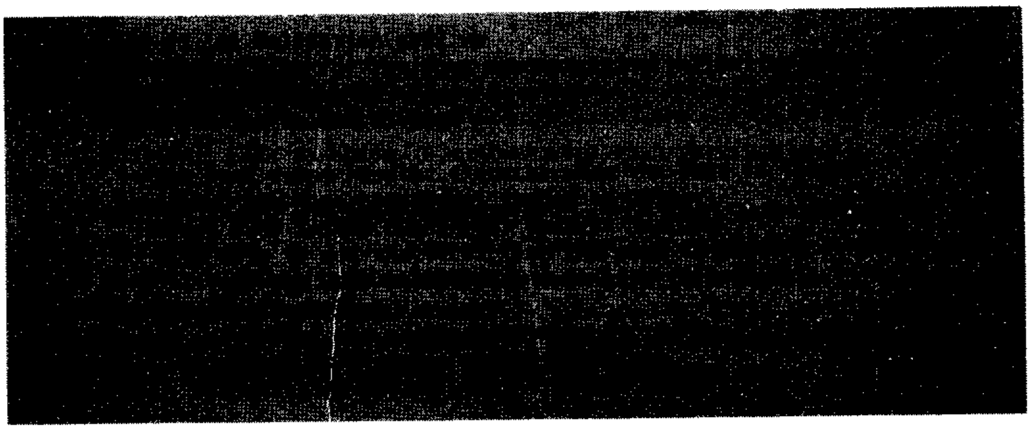


Fig. 25.18 | Code-behind file for the form demonstrating validation controls. (Part 1 of 2.)

[illegible]

Fig. 25.18 | Code-behind file for the form demonstrating validation controls. (Part 2 of 2.)

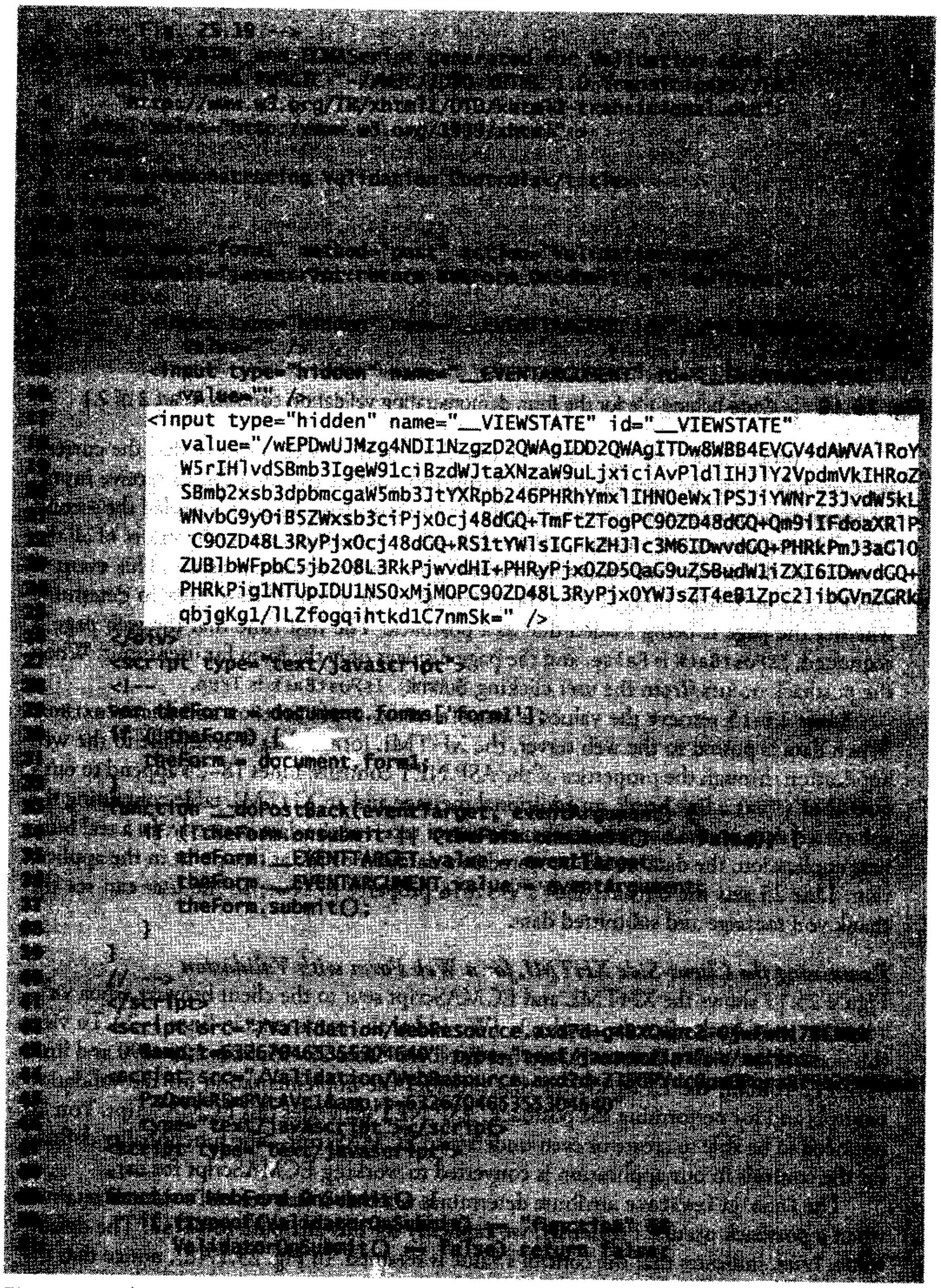
Web programmers using ASP.NET often design their web pages so that the current page reloads when the user submits the form; this enables the program to receive input, process it as necessary and display the results in the same page when it is loaded the second time. These pages usually contain a form that, when submitted, sends the values of all the controls to the server and causes the current page to be requested again. This event is known as a **postback**. Line 11 uses the **`IsPostBack`** property of class `Page` to determine whether the page is being loaded due to a postback. The first time that the web page is requested, `IsPostBack` is `False`, and the page displays only the form for user input. When the postback occurs (from the user clicking **Submit**), `IsPostBack` is `True`.

Lines 13–15 retrieve the values of `nameTextBox`, `emailTextBox` and `phoneTextBox`. When data is posted to the web server, the XHTML form's data is accessible to the web application through the properties of the ASP.NET controls. Lines 18–24 append to `outputLabel`'s `Text` a line break, an additional message and an XHTML table containing the submitted data, so the user knows that the server received the data correctly. In a real business application, the data would be stored in a database or file at this point in the application. Line 25 sets the `outputLabel`'s `Visible` property to `True`, so the user can see the thank you message and submitted data.

Examining the Client-Side XHTML for a Web Form with Validation

Figure 25.19 shows the XHTML and ECMAScript sent to the client browser when `Validation.aspx` loads after the postback. (We added the comments in lines 1–2.) To view this code, select **View > Source** in Internet Explorer. Lines 27–55, lines 126–190 and lines 196–212 contain the ECMAScript that provides the implementation for the validation controls and for performing the postback. ASP.NET generates this ECMAScript. You do not need to be able to create or even understand ECMAScript—the functionality defined for the controls in our application is converted to working ECMAScript for us.

The `EnableViewState` attribute determines whether a web control's value is retained when a postback occurs. Previously, we explicitly set this attribute to `False`. The default value, `True`, indicates that the control's value is retained. In Fig. 25.17(d), notice that the user input is retained after the postback occurs. A `hidden` input in the XHTML document (lines 17–25 of Fig. 25.19) contains the data of the controls on this page. This element is always named `__VIEWSTATE` and stores the controls' data as an encoded string.



```
<!-- Fig. 25.19 -->
[illegible]
[illegible]
   http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
[illegible]
   <title>Demonstrating Validation Controls</title>
[illegible]
[illegible]
   [illegible]
   [illegible]
   [illegible]
      [illegible]
         value="" />
      <input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT"
         value="" />
      <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
         value="/wEPDwUJMzg4NDI1NzgzD2QWAgIDD2QWAgITDw8WBB4EVGV4dAWVAlRoY
         W5rIHlvdSBmb3IgeW91ciBzdWJtaXNzaW9uLjxiciAvPldlIHJlY2VpdmVkIHRoZ
         SBmb2xsb3dpbmcgaW5mb3JtYXRpb246PHRhYmxlIHN0eWxlPSJiYWNrZ3JvdW5kL
         WNvbG9yOiB5ZWxsb3ciPjx0cj48dGQ+TmFtZTogPC90ZD48dGQ+Qm9iIFdoaXRlP
          C90ZD48L3RyPjx0cj48dGQ+RS1tYWlsIGFkZHJlc3M6IDwvdGQ+PHRkPmJ3aGl0
         ZUBlbWFpbC5jb208L3RkPjwvdHI+PHRyPjx0ZD5QaG9uZSBudW1iZXI6IDwvdGQ+
         PHRkPig1NTUpIDU1NS0xMjM0PC90ZD48L3RyPjx0YWJsZT4eB1Zpc2libGVnZGRk
         qbjgKg1/lLZfogqihtkd1C7nmSk=" />
   </div>
   <script type="text/javascript">
   <!--
   var theForm = document.forms['form1'];
   if (!theForm) {
      theForm = document.form1;
   }
   function __doPostBack(eventTarget, eventArgument) {
      if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
         theForm.__EVENTTARGET.value = eventTarget;
         theForm.__EVENTARGUMENT.value = eventArgument;
         theForm.submit();
      }
   }
   // -->
   </script>
   <script src="/Validation/WebResource.axd?d=g4BXOwpt2-0jwFwNi7BCNQ
      &amp;t=632670465355304640" type="text/javascript"></script>
   <script src="/Validation/WebResource.axd?d=[illegible]
      [illegible]&amp;t=632670465355304640"
      type="text/javascript"></script>
   <script type="text/javascript">
   <!--
   function WebForm_OnSubmit() {
      if (typeof(ValidatorOnSubmit) == "function" &&
         ValidatorOnSubmit() == false) return false;
```

Fig. 25.19 | XHTML and ECMAScript generated by ASP.NET and sent to the browser when Validation.aspx is requested. (Part 1 of 5.)

```
      return true;
      [illegible]
   }
   </script>
   [illegible]
      Please fill out the following form.<br />
      <em>All fields are required and must contain valid information</em>
      [illegible]
      <table>
         <tr>
            <td style="width: 100px" valign="top">Name:</td>
            <td style="width: 450px; valign="top">
               <input name="nameTextBox" type="text" [illegible]
                  id="nameTextBox" />
               <br />
               <span id="nameInputValidator"
                  style="color:Red;display:none;">
                  Please enter your name.</span> </td>
         </tr>
         <tr>
            <td style="width: 100px" valign="top">E-mail address:</td>
            <td style="width: 450px" valign="top">
               <input name="emailTextBox" type="text"
                  [illegible] id="emailTextBox" />
               [illegible]@[illegible].com<br />
               <span id="emailInputValidator"
                  style="color:Red;display:none;">
                  Please enter your e-mail address.</span>
               <span id="emailFormatValidator"
                  style="color:Red;display:none;">Please enter an e-mail
                  address in a valid format.</span> </td>
         </tr>
         <tr>
            <td style="width: 100px; height: 21px" valign="top">
               Phone number:</td>
            <td style="width: 450px; height: 21px" valign="top">
               <input name="phoneTextBox" type="text"
                  [illegible] (555) 555-1234 [illegible] id="phoneTextBox" />
               [illegible]
               <span id="phoneInputValidator"
                  style="color:Red;display:none;">
                  Please enter your phone number.</span>
               <span id="phoneFormatValidator"
                  [illegible]
                  [illegible]
         [illegible]
      [illegible]
      [illegible]
   [illegible]
```

Fig. 25.19 | XHTML and ECMAScript generated by ASP.NET and sent to the browser when Validation.aspx is requested. (Part 2 of 5.)

```
[illegible]
```

Fig. 25.19 | XHTML and ECMAScript generated by ASP.NET and sent to the browser when `Validation.aspx` is requested. (Part 3 of 5.)

```
[illegible]
   var phoneInputValidator = document.all ?
      document.all["phoneInputValidator"] :
      document.getElementById("phoneInputValidator");
[illegible]
         [illegible]OOKxcCCSN" />
[illegible]
   var Page_ValidationActive = false;
   [illegible](ValidatorOnLoad) [illegible]
      ValidatorOnLoad();
[illegible]
   function ValidatorOnSubmit() {
      if (Page_ValidationActive) {
```

Fig. 25.19 | XHTML and ECMAScript generated by ASP.NET and sent to the browser when `Validation.aspx` is requested. (Part 4 of 5.)

```
            return ValidatorCommonOnSubmit();
         }
         else {
            return true;
         }
      }
   // -->
   </script>
[illegible]
```

Fig. 25.19 | XHTML and ECMAScript generated by ASP.NET and sent to the browser when `Validation.aspx` is requested. (Part 5 of 5.)

Performance Tip 25.1

Setting `EnableViewState` *to* `False` *reduces the amount of data passed to the web server with each request.*

Software Engineering Observation 25.2

Client-side validation cannot be trusted by the server because there are too many ways top circumvent client-side validation. For this reason, all important validation should be peformed on the server.

25.4 Session Tracking

Originally, critics accused the Internet and e-businesses of failing to provide the kind of customized service typically experienced in "brick-and-mortar" stores. To address this problem, e-businesses began to establish mechanisms by which they could personalize users' browsing experiences, tailoring content to individual users while enabling them to bypass irrelevant information. Businesses achieve this level of service by tracking each customer's movement through the Internet and combining the collected data with information provided by the consumer, including billing information, personal preferences, interests and hobbies.

Personalization

Personalization makes it possible for e-businesses to communicate effectively with their customers and also improves users' ability to locate desired products and services. Companies that provide content of particular interest to users can establish relationships with customers and build on those relationships over time. Furthermore, by targeting consumers with personal offers, recommendations, advertisements, promotions and services, e-businesses create customer loyalty. Websites can use sophisticated technology to allow visitors to customize home pages to suit their individual needs and preferences. Similarly, online shopping sites often store personal information for customers, tailoring notifications and special offers to their interests. Such services encourage customers to visit sites more frequently and make purchases more regularly.

Privacy
A trade-off exists, however, between personalized e-business service and protection of privacy. Some consumers embrace the idea of tailored content, but others fear the possible adverse consequences if the info they provide to e-businesses is released or collected by tracking technologies. Consumers and privacy advocates ask: What if the e-business to which we give personal data sells or gives that information to another organization without our knowledge? What if we do not want our actions on the Internet—a supposedly anonymous medium—to be tracked and recorded by unknown parties? What if unauthorized parties gain access to sensitive private data, such as credit card numbers or medical history? All of these are questions that must be debated and addressed by programmers, consumers, e-businesses and lawmakers alike.

Recognizing Clients
To provide personalized services to consumers, e-businesses must be able to recognize clients when they request information from a site. As we have discussed, the request/response system on which the web operates is facilitated by HTTP. Unfortunately, HTTP is a stateless protocol. This means that web servers cannot determine whether a request comes from a particular client or whether the same or different clients generate a series of requests.

To circumvent this problem, sites can use the concept of a "session" to identify individual clients. A session represents a unique client on a website. If the client leaves a site and then returns later, the client will still be recognized as the same user. To help the server distinguish among clients, each client must identify itself to the server. Tracking individual clients, known as **session tracking**, can be achieved in a number of ways. One popular technique uses cookies (Section 25.4.1); another uses ASP.NET's `HttpSessionState` object (Section 25.4.2). Additional session-tracking techniques include the use of `input` `form` elements of type `"hidden"` and URL rewriting. Using `"hidden"` form elements, a Web Form can write session-tracking data into a `form` in the web page that it returns to the client in response to a prior request. When the user submits the form in the new web page, all the form data, including the `"hidden"` fields, is sent to the form handler on the web server. When a website performs URL rewriting, the Web Form embeds session-tracking information directly in the URLs of hyperlinks that the user clicks to send subsequent requests to the web server.

Note that our previous examples set the Web Form's `EnableSessionState` property to `False`. However, because we wish to use session tracking in the following examples, we keep this property's default setting—`True`.

25.4.1 Cookies

Cookies provide web developers with a tool for identifying and tracking web users. A cookie is a piece of data stored in a small text file on the user's computer. A cookie maintains information about the client during and between browser sessions. The first time a user visits the website, the user's computer might receive a cookie; this cookie is then retrieved each time the user revisits that site. The collected information is intended to be an anonymous record containing data that is used to personalize the user's future visits to the site. For example, cookies in a shopping application might store unique identifiers for users. When a user adds items to an online shopping cart or performs another task resulting in a request to the web server, the server receives a cookie containing the user's unique identifier. The server then uses the unique identifier to locate the shopping cart and perform any necessary processing.

In addition to identifying users, cookies also can indicate users' shopping preferences. When a web server receives a request from a client, the server can examine the cookie(s) it sent to the client during previous communications, identify the users's preferences and immediately return products of interest to the client.

Every HTTP-based interaction between a client and a server includes a header containing information either about the request (when the communication is from the client to the server) or about the response (when the communication is from the server to the client). When a web server receives a request, the header includes information such as the request type (e.g., `Get`) and any cookies that have been sent previously from the server to be stored on the client machine. When the server formulates its response, the header information contains any cookies the server wants to store on the client computer and other information, such as the MIME type of the response.

The **expiration date** of a cookie determines how long the cookie remains on the client's computer. If you do not set an expiration date for a cookie, the web browser maintains the cookie for the duration of the browsing session. Otherwise, the Web browser maintains the cookie until the expiration date occurs. When the browser requests a resource from a web server, cookies previously sent to the client by that Web server are returned to the web server as part of the request formulated by the browser. Cookies are deleted when they **expire**.

Portability Tip 25.3

Users may disable cookies in their browsers to ensure that their privacy is protected. Such users will experience difficulty using sites that depend on cookies to maintain state information.

Using Cookies to Provide Book Recommendations

The next web application demonstrates the use of cookies. The example contains two pages. In the first page (Figs. 25.20–25.21), users select a favorite programming language from a group of radio buttons and submit the XHTML form to the web server for processing. The web server responds by creating a cookie that stores a record of the chosen language, as well as the ISBN number for a book on that topic. The server then returns an XHTML document to the browser, allowing the user either to select another favorite programming language or to view the second page in our application (Figs. 25.22–25.23), which lists recommended books pertaining to the programming language that the user selected previously. When the user clicks the hyperlink, the cookies previously stored on the client are read and used to form the list of book recommendations.

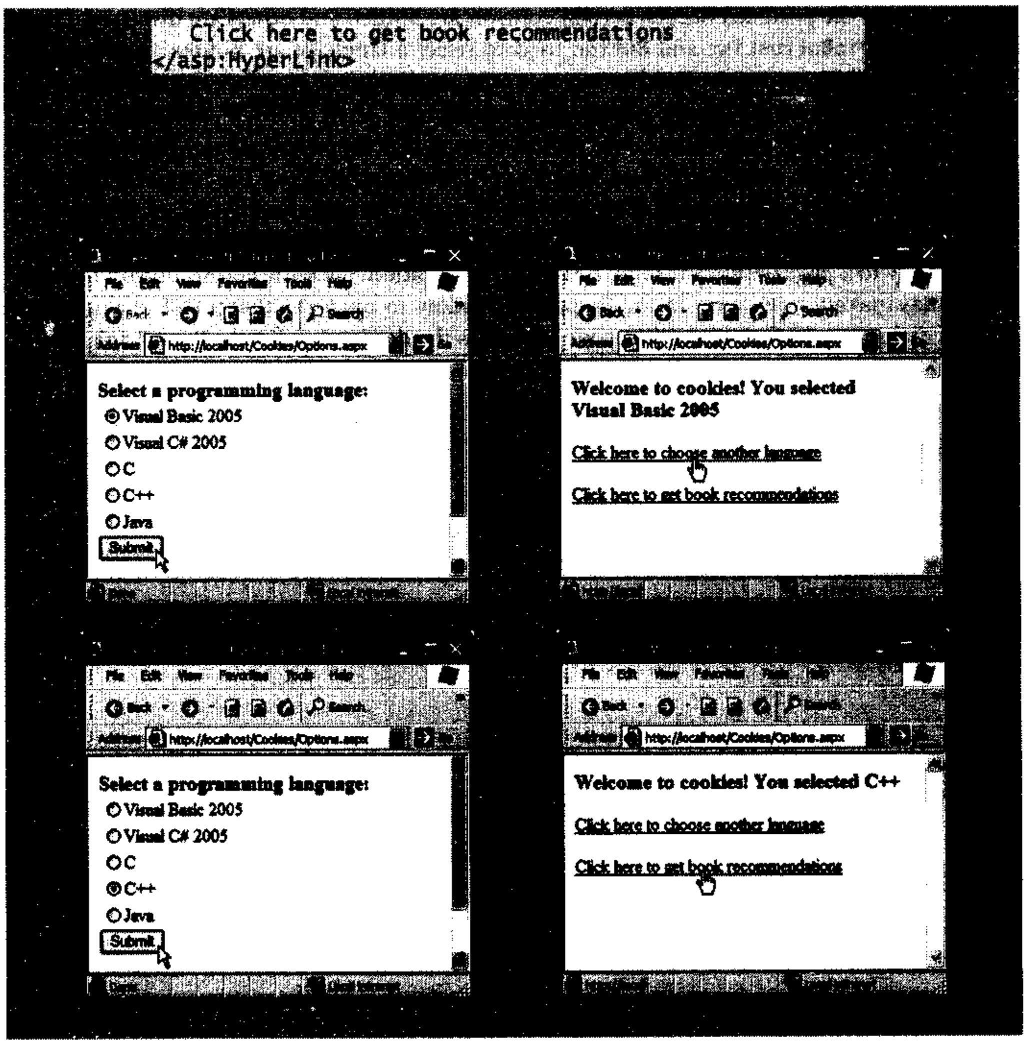
The ASPX file in Fig. 25.20 contains five radio buttons (lines 20–26) with the values **Visual Basic 2005**, **Visual C# 2005**, **C**, **C++**, and **Java**. Recall that you can set the values of radio buttons via the **ListItem Collection Editor**, which you open either by clicking the `RadioButtonList`'s `Items` property in the **Properties** window or by clicking the **Edit Items...** link in the **RadioButtonList Tasks** smart tag menu. The user selects a programming language by clicking one of the radio buttons. When the user clicks **Submit**, we'll create a cookie containing the selected language. Then, we'll add this cookie to the HTTP response header, so the cookie will be stored on the user's computer. Each time the user chooses a language and clicks **Submit**, a cookie is written to the client. Each time the client requests information from our web application, the cookies are sent back to the server.

When the postback occurs, certain controls are hidden and others are displayed. The `Label`, `RadioButtonList` and `Button` used to select a language are hidden. Toward the bottom of the page, a `Label` and two `HyperLink`s are displayed. One link requests this page (lines 32–35), and the other requests `Recommendations.aspx` (lines 37–40). Clicking the first hyperlink (the one that requests the current page) does not cause a postback to occur. The file `Options.aspx` is specified in the `NavigateUrl` property of the hyperlink. When the hyperlink is clicked, a new request for this page occurs. Recall that earlier in the chapter, we set `NavigateUrl` to a remote website (`http://www.deitel.com`). To set this property to a page within the same ASP.NET application, click the ellipsis button next to the `NavigateUrl` property in the **Properties** window to open the **Select URL** dialog. Use this dialog to select a page within your project as the destination for the `HyperLink`.

```
 1  <%-- Fig. 25.20: Options.aspx --%>
 2  <%-- Allows client to select programming languages and access --%>
 3  <%-- book recommendations. --%>
 4  <%@ Page Language="VB" AutoEventWireup="false"
 5     CodeFile="Options.aspx.vb" Inherits="Options" %>
 6
 7  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 8     [illegible]
 9
10  [illegible]="http://www.w3.org/1999/xhtml" >
11  <head runat="server">
12     <title>Cookies</title>
13  </head>
14  <body>
15     <form id="form1" runat="server">
16     <div>
17        <asp:Label ID="promptLabel" runat="server" Font-Bold="True"
18           Font-Size="Large" Text="Select a programming language:">
19        </asp:Label>
20        <asp:RadioButtonList ID="languageList" runat="server">
21           <asp:ListItem>Visual Basic 2005</asp:ListItem>
22           <asp:ListItem>Visual C# 2005</asp:ListItem>
23           <asp:ListItem>C</asp:ListItem>
24           <asp:ListItem>C++</asp:ListItem>
25           <asp:ListItem>Java</asp:ListItem>
26        </asp:RadioButtonList>
27        <asp:Button ID="submitButton" runat="server" Text="Submit" />
28        <asp:Label ID="responseLabel" runat="server" Font-Bold="True"
29           Font-Size="Large" Text="Welcome to cookies!" Visible="False">
30        </asp:Label><br />
31        <br />
32        <asp:HyperLink ID="languageLink" runat="server"
33           NavigateUrl="~/Options.aspx" Visible="False">
34           Click here to choose another language
35        </asp:HyperLink><br />
36        <br />
37        <asp:HyperLink ID="recommendationsLink" runat="server"
38           NavigateUrl="~/Recommendations.aspx" Visible="False">
```

Fig. 25.20 | ASPX file that presents a list of programming languages. (Part 1 of 2.)

```
        Click here to get book recommendations
</asp:HyperLink>
```



Select a programming language:
- (● Visual Basic 2005)
- Visual C# 2005
- C
- C++
- Java

Submit

Welcome to cookies! You selected Visual Basic 2005

Click here to choose another language

Click here to get book recommendations

Select a programming language:
- Visual Basic 2005
- Visual C# 2005
- C
- (● C++)
- Java

Submit

Welcome to cookies! You selected C++

Click here to choose another language

Click here to get book recommendations

File Edit View Favorites Tools Help
Back
Search
Address http://localhost/Cookies/Options.aspx
Go

Fig. 25.20 | ASPX file that presents a list of programming languages. (Part 2 of 2.)

Adding and Linking to a New Web Form

Setting the `NavigateUrl` property to a page in the current application requires that the destination page exist already. Thus, to set the `NavigateUrl` property of the second link (the one that requests the page with book recommendations) to `Recommendations.aspx`, you must first create this file by right clicking the project location in the **Solution Explorer** and selecting **Add New Item...** from the menu that appears. In the **Add New Item** dialog, select **Web Form** from the **Templates** pane and change the name of the file to `Recommendations.aspx`. Finally, check the box labeled **Place code in separate file** to indicate that the IDE should create a code-behind file for this ASPX file. Click **Add** to create the file. (We discuss the contents of this ASPX file and code-behind file shortly.) Once the `Recommendations.aspx` file exists, you can select it as the `NavigateUrl` value for a `HyperLink` in the **Select URL** dialog.

Writing Cookies in a Code-Behind File

Figure 25.21 presents the code-behind file for `Options.aspx` (Fig. 25.20). This file contains the code that writes a cookie to the client machine when the user selects a programming language. The code-behind file also modifies the appearance of the page in response to a postback.

[redacted]

```
' stores values to represent books as cookies
Private books As New System.Collections.Hashtable()
```

[redacted]

```
   books.Add("Visual Basic 2005", "0-13-186900-0")
   books.Add("Visual C# 2005", "0-13-152523-9")
   books.Add("C", "0-13-142644-3")
   books.Add("C++", "0-13-185757-6")
   books.Add("Java", "0-13-148398-6")
```

[redacted]

```
      ' if the user made a selection, display it in responseLabel
      If languageList.SelectedItem IsNot Nothing Then
         responseLabel.Text &= " You selected " & _
            languageList.SelectedItem.Text.ToString()
      Else
         responseLabel.Text &= " You did not select a language."
      End If
```

[redacted]

Fig. 25.21 | Code-behind file that writes a cookie to the client. (Part 1 of 2.)

```
46     ' write a cookie to record the user's selection
47     Protected Sub submitButton_Click(ByVal sender As Object, _
48        ByVal e As System.EventArgs) Handles submitButton.Click
49        ' if the user made a selection
50        If languageList.SelectedItem IsNot Nothing Then
51           Dim language As String = languageList.SelectedItem.ToString()
52
53           ' get ISBN number of book for the given language
54           Dim ISBN As String = books(language).ToString()
55
56           ' create cookie using language-ISBN name-value pair
57           Dim cookie As New HttpCookie(language, ISBN)
58
59           ' add cookie to response to place it on the user's machine
60           Response.Cookies.Add(cookie)
61        End If
62     End Sub ' submitButton_Click
63  End Class ' Options
```

Fig. 25.21 | Code-behind file that writes a cookie to the client. (Part 2 of 2.)

Line 7 creates variable books as a **Hashtable** (namespace System.Collections)—a data structure that stores **key–value pairs**. A program uses the key to store and retrieve the associated value in the Hashtable. In this example, the keys are strings containing the programming languages' names, and the values are strings containing the ISBN numbers for the recommended books. Class Hashtable provides method **Add**, which takes as arguments a key and a value. A value that is added via method Add is placed in the Hashtable at a location determined by the key. The value for a specific Hashtable entry can be determined by indexing the Hashtable with that value's key. The expression

HashtableName(*keyName*)

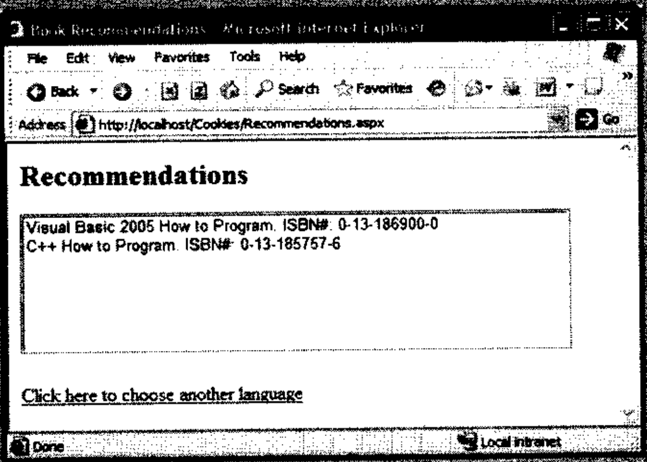
returns the value in the key–value pair in which *keyName* is the key. For example, the expression books(language) in line 54 returns the value that corresponds to the key contained in language.

Clicking the **Submit** button creates a cookie if a language is selected and causes a postback to occur. In the submitButton_Click event handler (lines 47–62), a new cookie object (of type **HttpCookie**) is created to store the language and its corresponding ISBN number (line 57). This cookie is then Added to the **Cookies** collection sent as part of the HTTP response header (line 60). The postback causes the condition in the If statement of Page_Load (line 24) to evaluate to True, and lines 27–42 execute. Lines 27–29 reveal the initially hidden controls responseLabel, languageLink and recommendationsLink. Lines 32–34 hide the controls used to obtain the user's language selection. Line 37 determines whether the user selected a language. If so, that language is displayed in responseLabel (lines 38–39). Otherwise, text indicating that a language was not selected is displayed in responseLabel (line 41).

Displaying Book Recommendations Based on Cookie Values

After the postback of Options.aspx, the user may request a book recommendation. The book recommendation hyperlink forwards the user to Recommendations.aspx (Fig. 25.22) to display the recommendations based on the user's language selections.

```
1  <%-- Fig. 25.22: Recommendations.aspx --%>
2  <%-- Displays book recommendations using cookies. --%>
3  <%@ Page Language="VB" AutoEventWireup="false"
4     CodeFile="Recommendations.aspx.vb" Inherits="Recommendations" %>
5
6  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
7     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
8
9  <html xmlns="http://www.w3.org/1999/xhtml">
10 <head runat="server">
11     <title>Book Recommendations</title>
12 </head>
13 <body>
14     <form id="form1" runat="server">
15     <div>
16        <asp:Label ID="recommendationsLabel" runat="server"
17           Font-Bold="True" Font-Size="X-Large" Text="Recommendations">
18        </asp:Label><br />
19        <br />
20        <asp:ListBox ID="booksListBox" runat="server" Height="125px"
21           Width="450px"></asp:ListBox><br />
22        <br />
23        <asp:HyperLink ID="languageLink" runat="server"
24           [illegible]
25           [illegible] here to choose another language
26        </asp:HyperLink>[illegible]
27     </form>
28 </body>
29 [illegible]
```


Book Recommendations - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/Cookies/Recommendations.aspx
Go
Done
Local intranet

Recommendations

Visual Basic 2005 How to Program. ISBN#: 0-13-186900-0
C++ How to Program. ISBN#: 0-13-185757-6

Click here to choose another language

Fig. 25.22 | ASPX file that displays book recommendations based on cookies.

Recommendations.aspx contains a Label (lines 16–18), a ListBox (lines 20–21) and a HyperLink (lines 23–26). The Label displays the text **Recommendations** if the user selects one or more languages; otherwise, it displays **No Recommendations**. The ListBox displays the recommendations specified by the code-behind file (Fig. 25.23). The `HyperLink` allows the user to return to `Options.aspx` to select additional languages.

Code-Behind File That Creates Book Recommendations from Cookies

In the code-behind file (Fig. 25.23), method `Page_Init` (lines 7–28) retrieves the cookies from the client, using the `Request` object's **`Cookies`** property (line 10). This returns a collection of type **`HttpCookieCollection`**, containing cookies that have previously been written to the client. Cookies can be read by an application only if they were created in the domain in which the application is running—a web server can never access cookies created outside the domain associated with that server. For example, a cookie created by a web server in the `deitel.com` domain cannot be read by a web server in any other domain. [*Note:* Depending on the settings in `web.config` and whether other pages store cookies, other cookie values may be displayed by this web application.]

Line 13 determines whether at least one cookie exists. Lines 14–17 add the information in the cookie(s) to the `booksListBox`. The `loop` retrieves the name and value of each cookie using `i`, the loop's control variable, to determine the current value in the cookie collection. The **`Name`** and **`Value`** properties of class `HttpCookie`, which contain the language and corresponding ISBN, respectively, are concatenated with `" How to Program.`

```
[illegible]
   [illegible] and populate ListBox with any book recommendations
   [illegible] Sub Page_Init(ByVal sender As Object, _
      [illegible] Handles MyBase.Init
      ' retrieve client's cookies
      Dim cookies As HttpCookieCollection = Request.Cookies

      ' if there are cookies, list the appropriate books and [illegible]
      If cookies.Count <> 0 Then
         For i As Integer = 0 To cookies.Count - 1
            booksListBox.Items.Add(cookies(i).Name & _
               " How to Program. ISBN#: " & cookies(i).Value)
         Next
      Else
         ' if there are no cookies, then no language was chosen, so
         ' display appropriate message and clear and hide booksListBox
         recommendationsLabel.Text = "No Recommendations"
         booksListBox.Items.Clear()
         booksListBox.Visible = False

         ' modify languageLink because no language was selected
         languageLink.Text = "Click here to choose a language"
27    End If
28 End Sub ' Page_Init
29 End Class ' Recommendations
```

Fig. 25.23 | Reading cookies from a client to determine book recommendations.

ISBN# " and added to the ListBox. Lines 21–26 execute if no language was selected. We summarize some commonly used HttpCookie properties in Fig. 25.24.

Domain	Returns a String containing the cookie's domain (i.e., the domain of the web server running the application that wrote the cookie). This determines which web servers can receive the cookie. By default, cookies are sent to the web server that originally sent the cookie to the client. Changing the Domain property causes the cookie to be returned to a web server other than the one that originally wrote it.
Expires	Returns a DateTime object indicating when the browser can delete the cookie.
Name	Returns a String containing the cookie's name.
Path	Returns a String containing the path to a directory on the server (i.e., the Domain) to which the cookie applies. Cookies can be "targeted" to specific directories on the web server. By default, a cookie is returned only to applications operating in the same directory as the application that sent the cookie or a subdirectory of that directory. Changing the Path property causes the cookie to be returned to a directory other than the one from which it was originally written.
Secure	Returns a Boolean value indicating whether the cookie should be transmitted through a secure protocol. The value True causes a secure protocol to be used.
Value	Returns a String containing the cookie's value.

Fig. 25.24 | HttpCookie properties.

25.4.2 Session Tracking with HttpSessionState

Session-tracking capabilities are provided by the FCL class **HttpSessionState**. To demonstrate basic session-tracking techniques, we modified the example of Figs. 25.20–25.23 to use HttpSessionState objects. Figures 25.25–25.26 present the ASPX file and code-behind file for Options.aspx. Figures 25.28–25.29 present the ASPX file and code-behind file for Recommendations.aspx. Options.aspx is similar to the version presented in Fig. 25.20, but Fig. 25.25 contains two additional Labels (lines 32–33 and lines 35–36), which we discuss shortly.

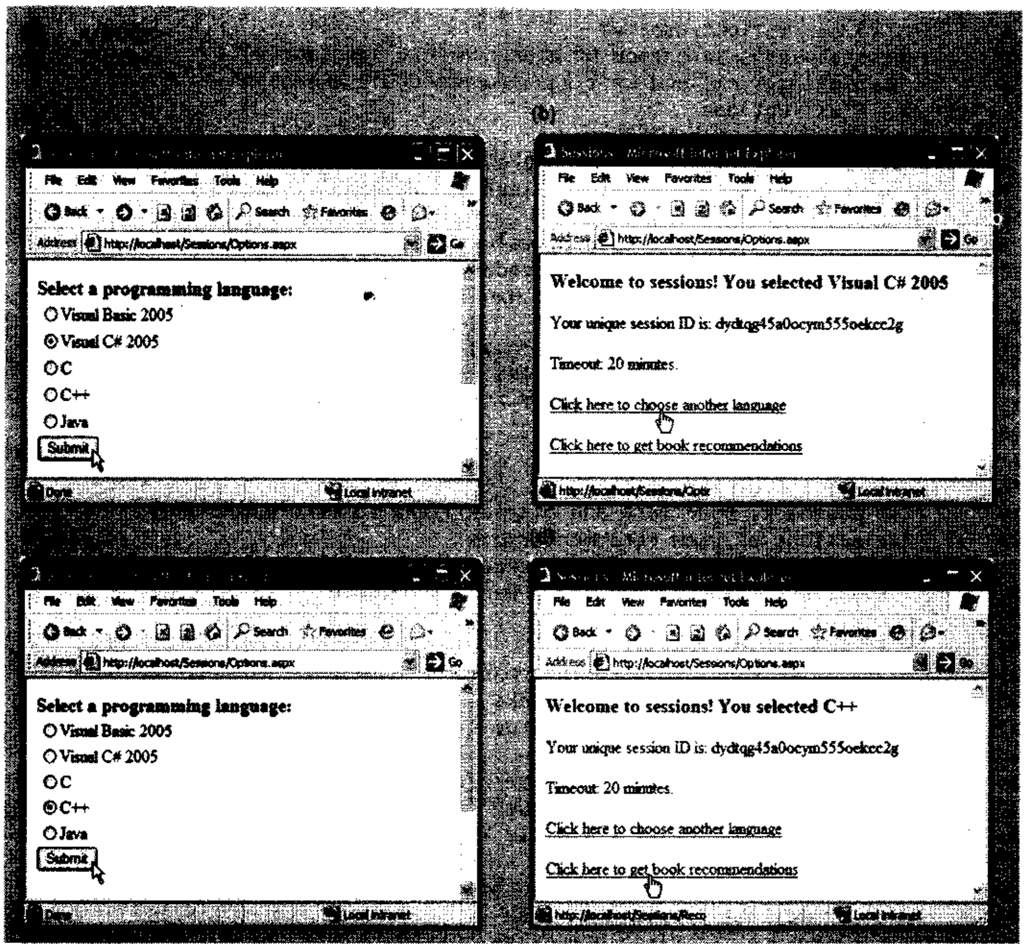
Every Web Form includes an HttpSessionState object, which is accessible through property **Session** of class Page. Throughout this section, we use property Session to manipulate our page's HttpSessionState object. When the web page is requested, an HttpSessionState object is created and assigned to the Page's Session property. As a result, we often refer to property Session as the Session object.

Adding Session Items

When the user presses **Submit** on the Web Form, submitButton_Click is invoked in the code-behind file (Fig. 25.26, lines 55–66). Method submitButton_Click responds by adding a key–value pair to our Session object, specifying the language chosen and the ISBN number for a book on that language. These key–value pairs are often referred to as session items. Next, a postback occurs. Each time the user clicks **Submit**, `submitButton_Click` adds a new session item to the `HttpSessionState` object. Because much of this example is identical to the last example, we concentrate on the new features.

```
<%-- Fig. 25.25: Options.aspx --%>
<%-- Allows client to select programming languages and access --%>
<%-- book recommendations. --%>
<%@ Page Language="VB" AutoEventWireup="false"
   CodeFile="Options.aspx.vb" Inherits="Options" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
   <title>Sessions</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Label ID="promptLabel" runat="server" Font-Bold="True"
         Font-Size="Large" Text="Select a programming language:">
      </asp:Label>
      <asp:RadioButtonList ID="languageList" runat="server">
         <asp:ListItem>Visual Basic 2005</asp:ListItem>
         <asp:ListItem>Visual C# 2005</asp:ListItem>
         <asp:ListItem>C</asp:ListItem>
         <asp:ListItem>C++</asp:ListItem>
         <asp:ListItem>Java</asp:ListItem>
      </asp:RadioButtonList>
      <asp:Button ID="submitButton" runat="server" Text="Submit" />
      <asp:Label ID="responseLabel" runat="server" Font-Bold="True"
         Font-Size="Large" Text="Welcome to sessions!" Visible="False">
      </asp:Label><br />
      <br />
      <asp:Label ID="idLabel" runat="server" Visible="False">
      </asp:Label><br />
      <br />
      <asp:Label ID="timeoutLabel" runat="server" Visible="False">
      </asp:Label><br />
      <br />
      <asp:HyperLink ID="languageLink" runat="server"
         NavigateUrl="~/Options.aspx" Visible="False">
         Click here to choose another language
      </asp:HyperLink><br />
      <br />
      <asp:HyperLink ID="recommendationsLink" runat="server"
         NavigateUrl="~/Recommendations.aspx" Visible="False">
         Click here to get book recommendations
      </asp:HyperLink>
   </div>
   </form>
```

Fig. 25.25 | ASPX file that presents a list of programming languages. (Part 1 of 2.)



Select a programming language:
○ Visual Basic 2005
◉ Visual C# 2005
○ C
○ C++
○ Java
Submit

Welcome to sessions! You selected Visual C# 2005

Your unique session ID is: dydtqg45a0ocym555oekcc2g

Timeout: 20 minutes.

Click here to choose another language

Click here to get book recommendations

Select a programming language:
○ Visual Basic 2005
○ Visual C# 2005
○ C
◉ C++
○ Java
Submit

Welcome to sessions! You selected C++

Your unique session ID is: dydtqg45a0ocym555oekcc2g

Timeout: 20 minutes.

Click here to choose another language

Click here to get book recommendations

Fig. 25.25 | ASPX file that presents a list of programming languages. (Part 2 of 2.)

Software Engineering Observation 25.3

A Web Form should not use instance variables to maintain client state information, because each new request or postback is handled by a new instance of the page. Instead, maintain client state information in `HttpSessionState` *objects, because such objects are specific to each client.*

Like a cookie, an `HttpSessionState` object can store name–value pairs. These session items are placed in an `HttpSessionState` object by calling method **Add**. Line 64 calls `Add` to place the language and its corresponding recommended book's ISBN number in the `HttpSessionState` object. If the application calls method `Add` to add an attribute that has the same name as an attribute previously stored in a session, the object associated with that attribute is replaced.

Software Engineering Observation 25.4

A benefit of using `HttpSessionState` *objects (rather than cookies) is that* `HttpSessionState` *objects can store any type of object (not just* `Strings`*) as attribute values. This provides you with increased flexibility in determining the type of state information to maintain for clients.*

```
[illegible]
        Dim books As New System.Collections.[illegible]

        ' initializes the Hashtable of [illegible]
        Protected Sub Page_Init(ByVal [illegible] As Object, _
            ByVal e As System.EventArgs) [illegible]
            [illegible] "0-13-[illegible]")
            books.Add("Visual C# 2005", "[illegible]")
            books.Add("C", "0-13-[illegible]")
            books.Add([illegible], "[illegible]")
            books.Add("Java", "0-13-[illegible]")
        End Sub ' Page_Init

        [illegible]
        [illegible]
        [illegible]
            [illegible]

            If IsPostBack Then
                [illegible]
                [illegible]
                responseLabel.Visible = [illegible]
                [illegible].Visible = True
                timeoutLabel.Visible = [illegible]
                languageLink.Visible = [illegible]
                [illegible] = True

                ' hide other controls [illegible]
                [illegible]
                [illegible]
                submitButton.Visible = False

                ' if the user made a selection [illegible]
                If languageList.Selected[illegible]
                    responseLabel.Text [illegible]
                        languageList.Selected[illegible]
                Else
                    responseLabel.Text &= " You did not select a [illegible]
                End If

                ' display session ID
                idLabel.Text = "Your unique session ID is: " & Session.SessionID

                ' display the timeout
                timeoutLabel.Text = "Timeout: " & Session.Timeout & " minutes."
            End If
        End Sub ' Page_Load
```

Fig. 25.26 | Processes user's selection of a programming language by displaying links and writing information in a `Session` object. (Part 1 of 2.)

```
   ' [illegible] to the Session
   [illegible]_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles submitButton.Click
      ' if the user made a selection
      If languageList.SelectedItem IsNot Nothing Then
         Dim language As String = languageList.SelectedItem.ToString()

         ' get ISBN number of book for the given language
         Dim ISBN As String = books(language).ToString()

         Session.Add(language, ISBN) ' add name/value pair to Session
      End If
   End Sub ' submitButton_Click
End Class ' Options
```

Fig. 25.26 | Processes user's selection of a programming language by displaying links and writing information in a Session object. (Part 2 of 2.)

The application handles the postback event (lines 24–51) in method Page_Load. Here, we retrieve information about the current client's session from the Session object's properties and display this information in the web page. The ASP.NET application contains information about the HttpSessionState object for the current client. Property **SessionID** (line 47) contains the **unique session ID**—a sequence of random letters and numbers. The first time a client connects to the web server, a unique session ID is created for that client and a temporary cookie is written to the client so the server can identify the client on subsequent requests. When the client makes additional requests, the client's session ID from that temporary cookie is compared with the session IDs stored in the web server's memory to retrieve the client's HttpSessionState object. Recall that clients may disable cookies in their web browsers to ensure that their privacy is protected. Such clients will experience difficulty using web applications that depend on HttpSessionState objects and cookies to maintain state information. The HttpSessionStrate property IsCookieless indicates whether URL rewriting or cookies are used for session tracking. Property **Timeout** (line 50) specifies the maximum amount of time that an HttpSessionState object can be inactive before it is discarded. Figure 25.27 lists some common HttpSessionState properties.

Count	Specifies the number of key–value pairs in the Session object.
IsNewSession	Indicates whether this is a new session (i.e., whether the session was created during loading of this page).
IsReadOnly	Indicates whether the Session object is read-only.

Fig. 25.27 | HttpSessionState properties. (Part 1 of 2.)

`Keys`	Returns a collection containing the `Session` object's keys.
`SessionID`	Returns the session's unique ID.
`Timeout`	Specifies the maximum number of minutes during which a session can be inactive (i.e., no requests are made) before the session expires. By default, this property is set to 20 minutes.

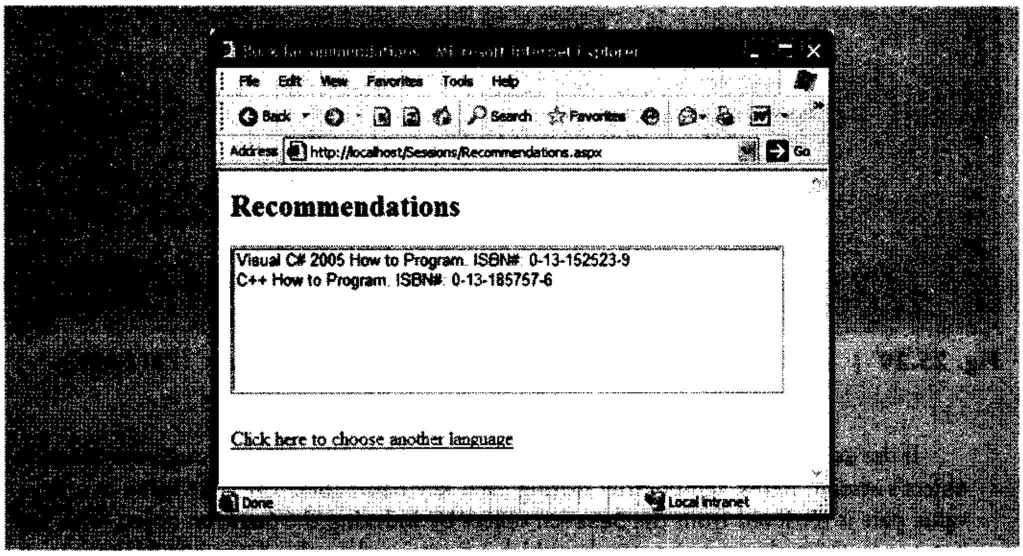
Fig. 25.27 | `HttpSessionState` properties. (Part 2 of 2.)

Displaying Recommendations Based on Session Values

As in the cookies example, this application provides a link to `Recommendations.aspx` (Fig. 25.28), which displays a list of book recommendations based on the user's language selections. Lines 20–21 define a `ListBox` web control that is used to present the recommendations to the user.

```
 1  <%-- Fig. 25.28: Recommendations.aspx --%>
 2  <%-- Displays book recommendations [illegible] --%>
 3  <%@ Page Language="VB" [illegible]
 4     CodeFile="Recommendations.aspx.vb" [illegible] %>
 5
 6  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 [illegible]
 7     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-[illegible]
 8
 9  <html xmlns="http://www.w3.org/1999/xhtml" >
10  <head id="Head1" runat="server">
11     <title>Book Recommendations</title>
12  </head>
13  <body>
14     <form id="form1" runat="server">
15     <div>
16        <asp:Label ID="recommendationsLabel" [illegible]
17           Font-Bold="True" Font-Size="X-Large" [illegible]
18        </asp:Label><br />
19        <br />
20        <asp:ListBox ID="booksListBox" runat="server" [illegible]
21           [illegible]
22        <br />
23        <asp:HyperLink ID="languageLink" runat="server"
24           NavigateUrl="~/Options.aspx">
25           Click here to choose another language
26        </asp:HyperLink></div>
27     </form>
28  </body>
29  </html>
```

Fig. 25.28 | Session-based book recommendations displayed in a `ListBox`. (Part 1 of 2.)


File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/Sessions/Recommendations.aspx
Go
Done
Local intranet

Recommendations

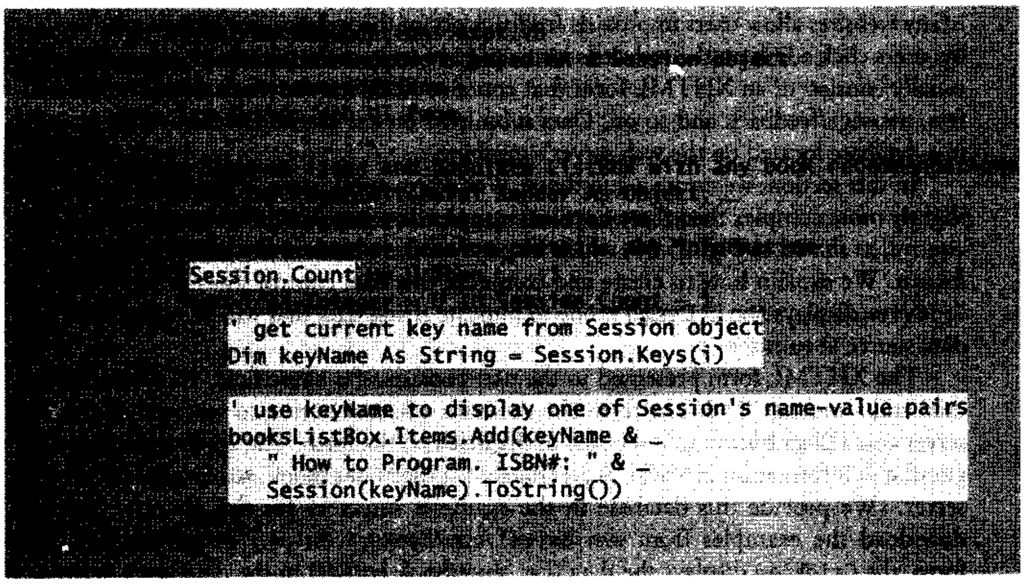
```
Visual C# 2005 How to Program. ISBN#: 0-13-152523-9
C++ How to Program. ISBN#: 0-13-185757-6
```

Click here to choose another language

Fig. 25.28 | Session-based book recommendations displayed in a `ListBox`. (Part 2 of 2.)

Code-Behind File That Creates Book Recommendations from a Session

Figure 25.29 presents the code-behind file for `Recommendations.aspx`. Event handler `Page_Init` (lines 7–30) retrieves the session information. If a user has not selected a language on `Options.aspx`, our `Session` object's **`Count`** property will be `0`. This property provides the number of session items contained in a `Session` object. If `Session` object's `Count` property is `0` (i.e., no language was selected), then we display the text **No Recommendations** and update the `Text` of the `HyperLink` back to `Options.aspx`.



```
Session.Count

   ' get current key name from Session object
   Dim keyName As String = Session.Keys(i)

   ' use keyName to display one of Session's name-value pairs
   booksListBox.Items.Add(keyName & _
      " How to Program. ISBN#: " & _
      Session(keyName).ToString())
```

Fig. 25.29 | Session data used to provide book recommendations to the user. (Part 1 of 2.)

```
         ' [illegible]
         ' [illegible]
         recommendationsLabel.Text = [illegible]
         booksListBox.Items.Clear()
         booksListBox.Visible = False

         ' modify languageLink because no language [illegible]
         languageLink.Text = "Click here to [illegible]
      End If
   End Sub ' Page_Init
End Class ' Recommendations
```

Fig. 25.29 | Session data used to provide book recommendations to the user. (Part 2 of 2.)

If the user has chosen a language, the loop in lines 11–19 iterates through our `Session` object's session items, temporarily storing each key name (line 13). The value in a key–value pair is retrieved from the `Session` object by indexing the `Session` object with the key name, using the same process by which we retrieved a value from our `Hashtable` in the preceding section.

Line 13 accesses the **Keys** property of class `HttpSessionState`, which returns a collection containing all the keys in the session. Line 13 indexes this collection to retrieve the current key. Lines 16–18 concatenate `keyName`'s value to the `String " How to Program. ISBN#: "` and the value from the `Session` object for which `keyName` is the key. This `String` is the recommendation that appears in the `ListBox`.

25.5 Case Study: Connecting to a Database in ASP.NET

Many websites allow users to provide feedback about the website in a **guestbook**. Typically, users click a link on the website's home page to request the guestbook page. This page usually consists of an XHTML form that contains fields for the user's name, e-mail address, message/feedback and so on. Data submitted on the guestbook form is then stored in a database located on the web server's machine.

In this section, we create a guestbook Web Form application. This example's GUI is slightly more complex than that of earlier examples. It contains a **`GridView`** ASP.NET data control, as shown in Fig. 25.30, which displays all the entries in the guestbook in tabular format. We explain how to create and configure this data control shortly. Note that the `GridView` displays **abc** in **Design** mode to indicate string data that will be retrieved from a data source at runtime.

The XHTML form presented to the user consists of a name field, an e-mail address field and a message field. The form also contains a **Submit** button to send the data to the server and a **Clear** button to reset each of the fields on the form. The application stores the guestbook information in a SQL Server database called `Guestbook.mdf` located on the web server. (We provide this database in the examples directory for this chapter. You can download the examples from `www.deitel.com/books/iw3htp4`.) Below the XHTML form, the `GridView` displays the data (i.e., guestbook entries) in the database's `Messages` table.

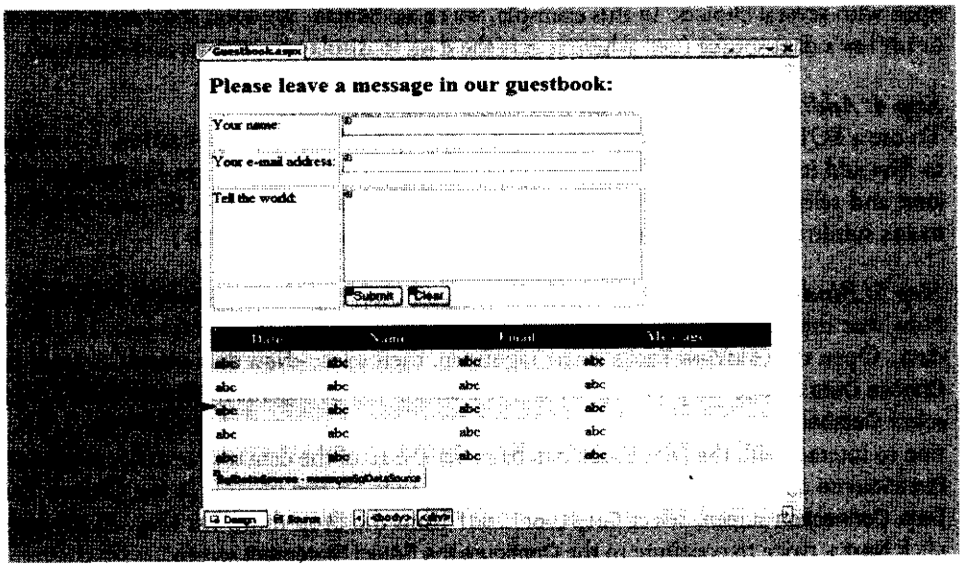


Fig. 25.30 | Guestbook application GUI in **Design** mode.

25.5.1 Building a Web Form That Displays Data from a Database

We now explain how to build this GUI and set up the data binding between the `GridView` control and the database. We present the ASPX file generated from the GUI later in the section, and we discuss the related code-behind file in the next section. To build the guestbook application, perform the following steps:

Step 1: Creating the Project
Create an **ASP.NET Web Site** named `Guestbook` and name the ASPX file `Guestbook.aspx`. Rename the class in the code-behind file `Guestbook`, and update the `Page` directive in the ASPX file accordingly.

Step 2: Creating the Form for User Input
In **Design** mode for the ASPX file, add the text `Please leave a message in our guestbook:` formatted as an `h2` header. As discussed in Section 25.3.1, insert an XHTML table with two columns and four rows, configured so that the text in each cell aligns with the top of the cell. Place the appropriate text (see Fig. 25.30) in the top three cells in the table's left column. Then place `TextBox`es named `nameTextBox`, `emailTextBox` and `messageTextBox` in the top three table cells in the right column. Set `messageTextBox` to be a multiline `TextBox`. Finally, add `Button`s named `submitButton` and `clearButton` to the bottom-right table cell. Set the buttons' `Text` properties to `Submit` and `Clear`, respectively. We discuss the buttons' event handlers when we present the code-behind file.

Step 3: Adding a `GridView` Control to the Web Form
Add a `GridView` named `messagesGridView` that will display the guestbook entries. This control appears in the **Data** section of the **Toolbox**. The colors for the `GridView` are specified through the **Auto Format...** link in the **GridView Tasks** smart tag menu that opens when you place the `GridView` on the page. Clicking this link causes an **Auto Format** dialog to

open with several choices. In this example, we chose **Simple**. We soon show how to set the GridView's data source (i.e., where it gets the data to display in its rows and columns).

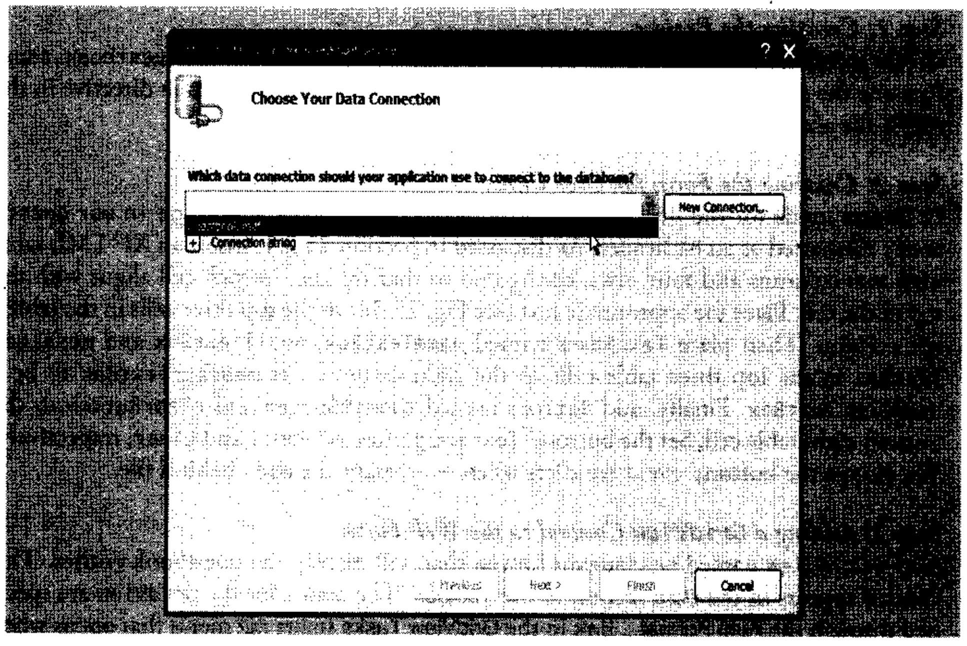
Step 4: Adding a Database to an ASP.NET Web Application

To use a SQL Server 2005 Express database in an ASP.NET web application, it is easiest to first add it to the project's App_Data folder. Right click this folder in the **Solution Explorer** and select **Add Existing Item....** Locate the Guestbook.mdf file in the exampleDatabases subdirectory of the chapter's examples directory, then click **Add**.

Step 5: Binding the GridView to the Messages Table of the Guestbook Database

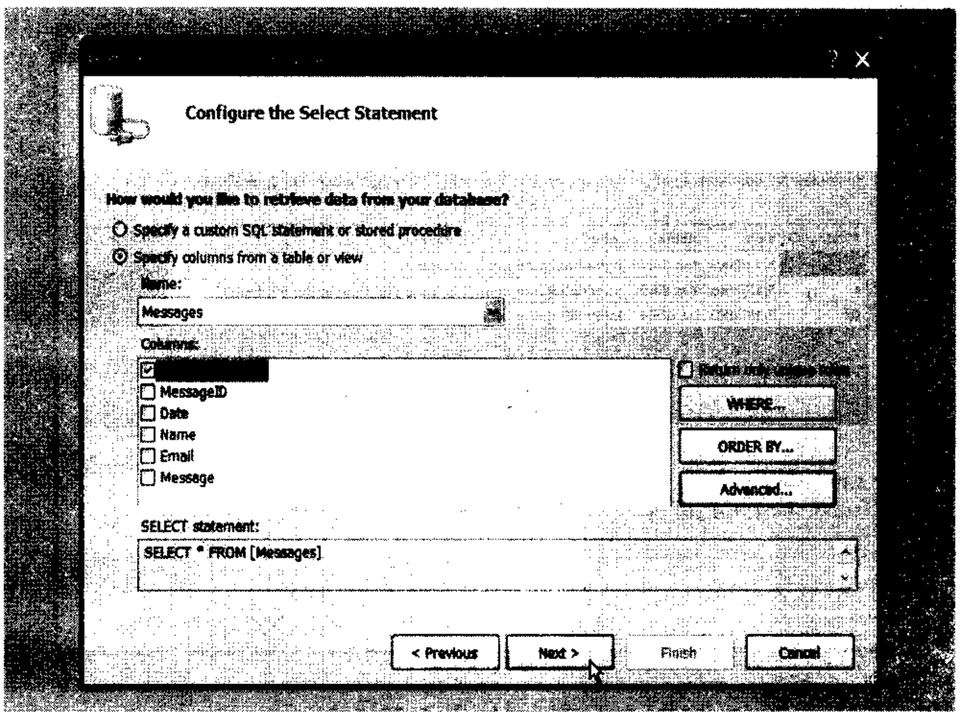
Now that the database is part of the project, we can configure the GridView to display its data. Open the **GridView Tasks** smart tag menu, then select **<New data source...>** from the **Choose Data Source** drop-down list. In the **Data Source Configuration Wizard** that appears, select **Database**. In this example, we use a SqlDataSource control that allows the application to interact with the Guestbook database. Set the ID of the data source to messagesSqlDataSource and click **OK** to begin the **Configure Data Source** wizard. In the **Choose Your Data Connection** screen, select Guestbook.mdf from the drop-down list (Fig. 25.31), then click **Next >** twice to continue to the **Configure the Select Statement** screen.

The **Configure the Select Statement** screen (Fig. 25.32) allows you to specify which data the SqlDataSource should retrieve from the database. Your choices on this page design a SELECT statement, shown in the bottom pane of the dialog. The **Name** drop-down list identifies a table in the database. The Guestbook database contains only one table named Messages, which is selected by default. In the **Columns** pane, click the checkbox marked with an asterisk (*) to indicate that you want to retrieve the data from all the columns in the **Message** table. Click the **Advanced** button, then check the box next to


Choose Your Data Connection

Which data connection should your application use to connect to the database?

Fig. 25.31 | **Configure Data Source** dialog in Visual Web Developer.



Configure the Select Statement

How would you like to retrieve data from your database?

- ○ Specify a custom SQL statement or stored procedure
- ◉ Specify columns from a table or view

Name: Messages

Columns:

- ☑ [illegible]
- ☐ MessageID
- ☐ Date
- ☐ Name
- ☐ Email
- ☐ Message

☐ Return only unique rows

SELECT statement:

```
SELECT * FROM [Messages]
```

Fig. 25.32 | Configuring the SELECT statement used by the SqlDataSource to retrieve data.

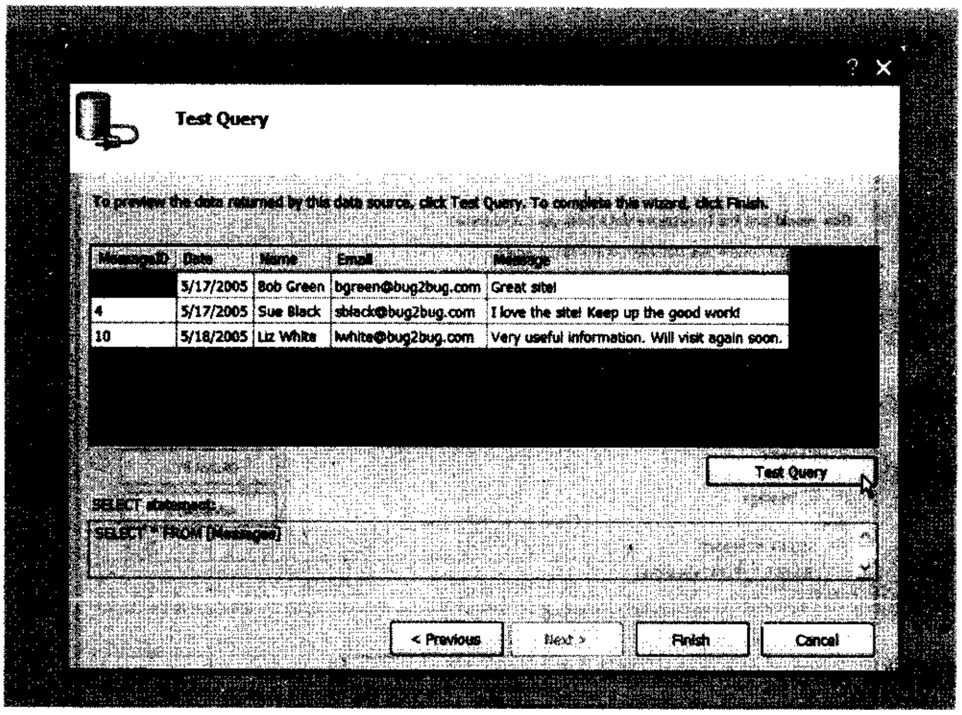
Generate UPDATE, INSERT and DELETE statements. This configures the SqlDataSource control to allow us to change data in, insert new data into and delete data from the database. We discuss inserting new guestbook entries based on users' form submissions shortly. Click **OK**, then click **Next >** to continue the **Configure Data Source** wizard.

The next screen of the wizard allows you to test the query that you just designed. Click **Test Query** to preview the data that will be retrieved by the SqlDataSource (shown in Fig. 25.33).

Finally, click **Finish** to complete the wizard. Notice that a control named messagesSqlDataSource now appears on the Web Form directly below the GridView (Fig. 25.34). This control is represented in **Design** mode as a gray box containing its type and name. This control will *not* appear on the web page—the gray box simply provides a way to manipulate the control visually through **Design** mode. Also notice that the GridView now has column headers that correspond to the columns in the Messages table and that the rows each contain either a number (which signifies an autoincremented column) or **abc** (which indicates string data). The actual data from the Guestbook database file will appear in these rows when the ASPX file is executed and viewed in a web browser.

Step 6: Modifying the Columns of the Data Source Displayed in the **GridView**

It is not necessary for site visitors to see the MessageID column when viewing past guestbook entries—this column is merely a unique primary key required by the Messages table within the database. Thus, we modify the GridView so that this column does not display



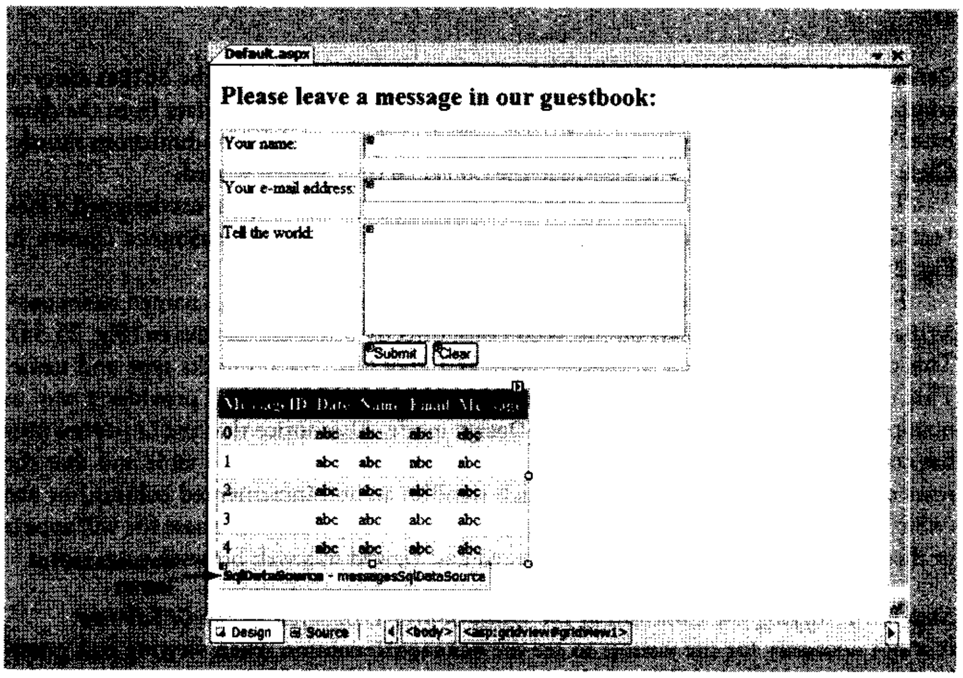
To preview the data returned by this data source, click Test Query. To complete this wizard, click Finish.

MessageID	Date	Name	Email	Message
	5/17/2005	Bob Green	bgreen@bug2bug.com	Great site!
4	5/17/2005	Sue Black	sblack@bug2bug.com	I love the site! Keep up the good work!
10	5/18/2005	Liz White	lwhite@bug2bug.com	Very useful information. Will visit again soon.

SELECT statement:

```
SELECT * FROM [Messages]
```

Fig. 25.33 | Previewing the data retrieved by the `SqlDataSource`.



Please leave a message in our guestbook:

Your name:	
Your e-mail address:	
Tell the world:	

MessageID	Date	Name	Email	Message
0	abc	abc	abc	abc
1	abc	abc	abc	abc
2	abc	abc	abc	abc
3	abc	abc	abc	abc
4	abc	abc	abc	abc

SqlDataSource - messagesSqlDataSource

Fig. 25.34 | **Design** mode displaying `SqlDataSource` control for a `GridView`.

on the Web Form. In the **GridView Tasks** smart tag menu, click **Edit Columns**. In the resulting **Fields** dialog (Fig. 25.35), select **MessageID** in the **Selected fields** pane, then click the **X**. This removes the `MessageID` column from the `GridView`. Click **OK** to return to the main IDE window. The `GridView` should now appear as in Fig. 25.30.

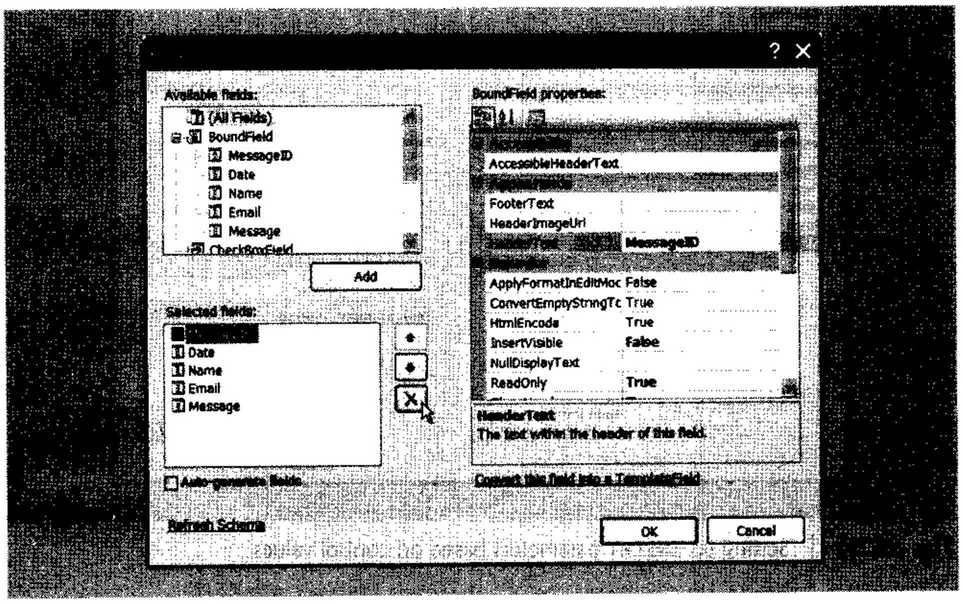
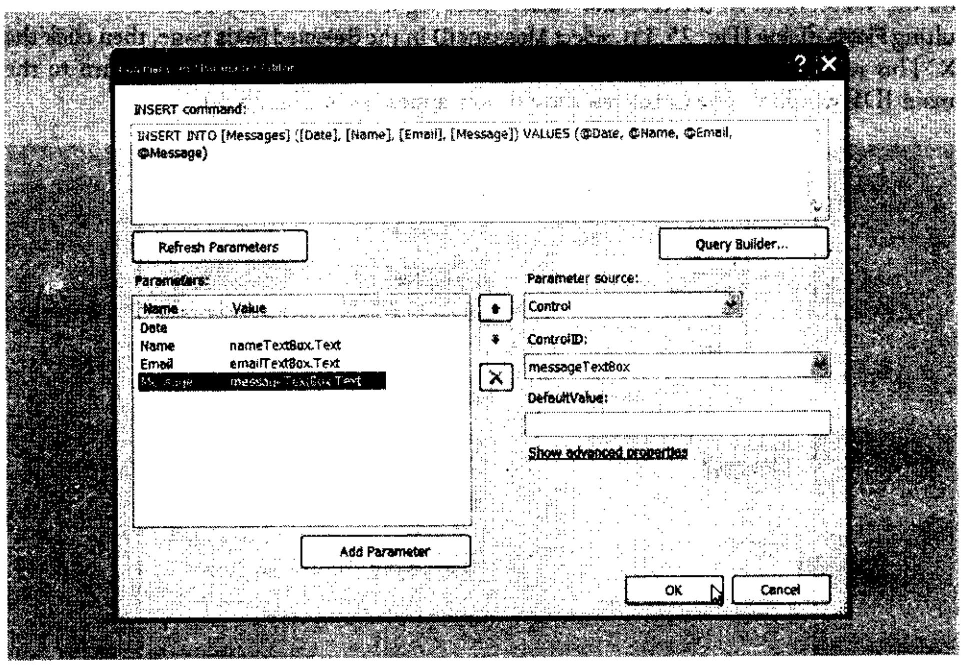


Fig. 25.35 | Removing the `MessageID` column from the `GridView`.

Step 7: Modifying the Way the `SqlDataSource` *Control Inserts Data*

When you create a `SqlDataSource` in the manner described here, it is configured to permit `INSERT` SQL operations against the database table from which it gathers data. You must specify the values to insert either programmatically or through other controls on the Web Form. In this example, we wish to insert the data entered by the user in the `nameTextBox`, `emailTextBox` and `messageTextBox` controls. We also want to insert the current date—we will specify the date to insert programmatically in the code-behind file, which we present shortly.

To configure the `SqlDataSource` to allow such an insertion, select the `messagesSqlDataSource` control then click the ellipsis button next to the control's **`InsertQuery`** property of the `messagesSqlDataSource` control in the **Properties** window. The **Command and Parameter Editor** (Fig. 25.36) that appears displays the `INSERT` command used by the `SqlDataSource` control. This command contains parameters `@Date`, `@Name`, `@Email` and `@Message`. You must provide values for these parameters before they are inserted into the database. Each parameter is listed in the **Parameters** section of the **Command and Parameter Editor**. Because we will set the **Date** parameter programmatically, we do not modify it here. For each of the remaining three parameters, select the parameter, then select **Control** from the **Parameter source** drop-down list. This indicates that the value of the parameter should be taken from a control. The **ControlID** drop-down list contains all the controls on the Web Form. Select the appropriate control for each parameter, then click **OK**. Now the `SqlDataSource` is configured to insert the user's name, e-mail address and message in the



INSERT command:

```
INSERT INTO [Messages] ([Date], [Name], [Email], [Message]) VALUES (@Date, @Name, @Email,
@Message)
```

Parameters:

Name	Value
Date	
Name	nameTextBox.Text
Email	emailTextBox.Text
Message	messageTextBox.Text

Parameter source: Control

ControlID: messageTextBox

DefaultValue:

Fig. 25.36 | Setting up `INSERT` parameters based on control values.

`Messages` table of the `Guestbook` database. We show how to set the date parameter and initiate the insert operation when the user clicks **Submit** shortly.

ASPX File for a Web Form That Interacts with a Database

The ASPX file generated by the guestbook GUI (and `messagesSqlDataSource` control) is shown in Fig. 25.37. This file contains a large amount of generated markup. We discuss only those parts that are new or noteworthy for the current example. Lines 19–58 contain the XHTML and ASP.NET elements that comprise the form that gathers user input. The `GridView` control appears in lines 60–85. The `<asp:GridView>` start tag (lines 60–63) contains properties that set various aspects of the `GridView`'s appearance and behavior, such as whether grid lines should be displayed between rows and columns. The **`DataSourceID`** property identifies the data source that is used to fill the `GridView` with data at runtime.

Lines 66–75 define the `Columns` that appear in the `GridView`. Each column is represented as a **`BoundField`**, because the values in the columns are bound to values retrieved from the data source (i.e., the `Messages` table of the `Guestbook` database). The `DataField` property of each `BoundField` identifies the column in the data source to which the column in the `GridView` is bound. The `HeaderText` property indicates the text that appears as the column header. By default, this is the name of the column in the data source, but you can change this property as desired. Lines 76–84 contain nested elements that define the styles used to format the `GridView`'s rows. The IDE configured these styles based on your selection of the **Simple** style in the **Auto Format** dialog for the `GridView`.

The `messagesSqlDataSource` is defined by the markup in lines 86–115 in Fig. 25.37. Line 87 contains a **`ConnectionString`** property, which indicates the connection through which the SqlDataSource control interacts with the database. The value of this property uses an ASP.NET **expression**, delimited by **<%$** and **%>**, to access the Guestbook-ConnectionString stored in the ConnectionStrings section of the application's Web.config configuration file. Recall that we created this connection string earlier in this section using the **Configure Data Source** wizard.Lines 88–95 define the **DeleteCommand**, **InsertCommand**, **SelectCommand** and **UpdateCommand** properties, which contain the DELETE, INSERT, SELECT and UPDATE SQL statements, respectively. These were generated by the **Configure Data Source** wizard. In this example, we use only the InsertCommand. We discuss invoking this command shortly.

```
<%-- Fig. 25.37: Guestbook.aspx --%>
<%-- Guestbook Web application with a form for users to submit --%>
<%-- guestbook entries and a GridView to view existing entries. --%>
<%@ Page Language="VB" AutoEventWireup="false"
   CodeFile="Guestbook.aspx.vb" Inherits="Guestbook" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Guestbook</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h2>
           Please leave a message in our guestbook:</h2>
        <table>
           <tr>
              <td style="width: 130px; height: 21px" valign="top">
                 Your name:<br />
              </td>
              <td style="width: 300px; height: 21px" valign="top">
                 <asp:TextBox ID="nameTextBox" runat="server"
                    Width="300px"></asp:TextBox>
              </td>
           </tr>
           <tr>
              <td style="width: 130px" valign="top">
                 Your e-mail address:<br />
              </td>
              <td style="width: 300px" valign="top">
                 <asp:TextBox ID="emailTextBox" runat="server"
                    Width="300px"></asp:TextBox>
              </td>
           </tr>
           <tr>
              <td style="width: 130px" valign="top">
                 Tell the world:<br />
              </td>
```

Fig. 25.37 | ASPX file for the guestbook application. (Part 1 of 4.)

```
[illegible]
            </asp:TextBox>
         </td>
[illegible]
      <td style="width: 130px" valign="top">
      </td>
      <td style="width: 300px" valign="top">
         <asp:Button ID="submitButton" runat="server"
[illegible]

   <asp:GridView ID="messagesGridView" runat="server"
[illegible]
                                DataSourceID="messagesSqlDataSource"
[illegible]
      <Columns>
         <asp:BoundField DataField="Date" HeaderText="Date"
            SortExpression="Date" />
         <asp:BoundField DataField="Name" HeaderText="Name"
            SortExpression="Name" />
         <asp:BoundField DataField="Email" HeaderText="Email"
            SortExpression="Email" />
         <asp:BoundField DataField="Message" HeaderText="Message"
            SortExpression="Message" />
      </Columns>
[illegible]
   <asp:SqlDataSource ID="messagesSqlDataSource" runat="server"
      ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
[illegible]
      SelectCommand="SELECT * FROM [Messages]"
[illegible]
```

Fig. 25.37 | ASPX file for the guestbook application. (Part 2 of 4.)