

Exploring ASP.NET and Web Forms

The last chapter covered the setup and configuration of the development environment. The development environment is an important necessity and will make the rest of your development more enjoyable.

This chapter explores Web Forms. Web Forms bring the structure and fun back to Web development. This chapter starts by looking at the two programming models for ASP.NET. It then looks at how ASP.NET uses server controls and at the HTML (HyperText Markup Language) and Web server controls. It finishes by looking at view state and post back.

Web Forms

Web Forms are an exciting part of the ASP.NET platform. Web Forms give the developer the ability to drag and drop ASP.NET server controls onto the form and easily program the events that are raised by the control. Web Forms have the following benefits:

Rendering. Web Forms are automatically rendered in any browser. In addition, Web Forms can be tweaked to work on a specific browser to take advantage of its features.

64 Chapter 3

Programming. Web Forms can be programmed using any .NET language, and Win32 API calls can be made directly from ASP.NET code.

.NET Framework. Web Forms are part of the .NET Framework, therefore Web Forms provide the benefits of the .NET Framework, such as performance, inheritance, type safety, structured error handling, automatic garbage collection, and xcopy deployment.

Extensibility. User controls, mobile controls, and other third-party controls can be added to extend Web Forms.

WYSIWYG. Visual Studio .NET provides the WYSIWYG (what you see is what you get) editor for creating Web Forms by dragging and dropping controls onto the Web Form.

Code Separation. Web Forms provide a code-behind page to allow the separation of HTML content from program code.

State Management. Provides the ability to maintain the view state of controls across Web calls.

Classroom Q & A

Q: I am currently developing my entire Web page by typing the HTML and client-side script into an ASCII editor. Are standard HTML tags with client-side script still available? Also, can I still use JavaScript for client-side code?

A: Yes. ASP.NET is focused around the added functionality of server controls, but you can still use standard HTML tags with client-side script as you have done in the past. As you become more familiar with the Visual Studio .NET environment, you may choose to change some of your controls to server controls to take advantage of the benefits that server controls provide.

Q: Can I use ASP and ASP.NET pages on the same Web site? Can I use Session and Application variables to share data between the ASP and ASP.NET pages?

A: Yes and no. You can run ASP and ASP.NET Web pages on the same Web site; in other words, the pages can coexist. You cannot share application and session data between the ASP and ASP.NET pages, because the ASP and ASP.NET run under separate contexts. What you will find is that you will have one set of Session and Application variables for the ASP pages and a different set of Session and Application variables for the ASP.NET pages.

Q: It's my understanding that there are two types of server controls. Can both types of server controls be used on the same Web page?

A: Yes. Visual Studio .NET provides HTML and Web server controls. You can provide a mixture of these controls on the same page. These controls will be covered in this chapter.

Q: I currently use VBScript and JavaScript to write my server-side code. Basically, if I am writing the ASP page and a function is easier to accomplish in JavaScript, I write it in JavaScript. For all other programming, on the page, I use VBScript. Can I continue to mix Visual Basic .NET and JavaScript on the same page?

A: No. ASP.NET requires server-side script to be written in the same language on a page-by-page basis. In addition, Visual Studio .NET requires a project to be written in a single server-side language.

Two ASP.NET Programming Models

People who are familiar with traditional ASP are accustomed to creating a single file for each Web page. ASP.NET supports the single-file programming model. Using the single-page programming model, the server code and the client-side tags and code are placed in the same file with an .aspx file extension. This doesn't do anything to help clean up spaghetti code, but the single-file model can be especially useful to ease the pain of migrating ASP code to ASP.NET.

The two-page model provides a separation of the server-side code and client-side HTML and code. The model offers the ability to use an .aspx page for the client-side presentation logic and a Visual Basic *code-behind* file with a .vb file extension for the server-side code.

This chapter starts by using the single-page model due to its simplicity. After most of the basic concepts are covered, the chapter switches to the two-page model. The two-page, or code-behind, model is used exclusively throughout the balance of the book due to the benefits it provides.

Simple ASP.NET Page

Using the single-page programming model, a simple Hello World page using ASP.NET can be written and saved to a file called vb.aspx containing the following:

```
<%@ Page Language="vb" %>
<HTML>
  <HEAD><title>Hello World Web Page</title></HEAD>
```

66 Chapter 3

```
<body>
  <form id="Form1" method="post" runat="server">
    <asp:TextBox id="Hi" runat="server">
      Hello World
    </asp:TextBox>
    <asp:Button id="Button1" runat="server" Text="Say Hi">
    </asp:Button>
  </form>
</body>
</HTML>
```

The first line of code contains the page directive, which contains the compiler language attribute. The compiler language attribute can only be used once on a page. If additional language attributes are on the page, they are ignored. Some language identifiers are shown in Table 3.1. If no language identifier is specified, the default is vb. The page directive has many other attributes, which will be covered throughout this book.

tip

The language identifiers that are configured on your machine may be found by looking in the machine.config file, which is located in the %systemroot%\Microsoft.NET\Framework\version\CONFIG folder. The machine.config file is an xml configuration file, which contains settings that are global to your machine. A search for *compilers* will expose all of the language identifiers that are configured on your computer. Always back up the machine.config file before making changes, as this file affects all .NET applications on the machine.

The rest of the page looks like standard HTML, except that this page contains three server controls: the form, the asp:TextBox, and the asp:Button. Server controls have the run="server" attribute. Server controls automatically maintain client-entered values across round trips to the server. ASP.NET automatically takes care of the code that is necessary to maintain state by placing the client-entered value in an attribute. In some cases, no acceptable attribute is available to hold the client-entered values. In those situations, the client-entered values are placed into a <input type="hidden"> tag.

Table 3.1 ASP.NET Language Identifiers

LANGUAGE	ACCEPTABLE IDENTIFIERS
Visual Basic .NET	vb; vbs; visualbasic; vbscript
Visual C#	c#; cs; csharp
Visual J#	VJ#; VJS; VJSharp
Visual JavaScript	js; jscript; javascript

When the page is displayed in the browser, the text box displays the initial Hello World message. A look at the client-side source reveals the following:

```
<HTML>
<HEAD><title>Hello World Web Page</title></HEAD>
  <body>
    <form name="Form1" method="post" action="vb.aspx" id="Form1">
      <input type="hidden"
        name="__VIEWSTATE"
        value="dDwtMTc2MjYxNDA2NTs7Pp6EUc0BOodWTOrpgefKJJjg3yEt"/>
      <input type="text"
        name="Hi"
        value="Hello World" id="Hi" />
      <input type="submit"
        name="Button1"
        value="Say Hi" id="Button1" />
    </form>
  </body>
</HTML>
```

The form server control was rendered as a standard HTML form tag with the action (the location that the data is posted to) set to the current page. A new control has been added automatically, called the __VIEWSTATE control. (More on the __VIEWSTATE control is provided in this chapter.) The asp:TextBox Web server control was rendered as an HTML text box and has its value set to "Hello World." The asp:button Web server control was rendered as an HTML Submit button.

If *Hi Universe* is typed into the text box and the button is clicked, the button will submit the form data to the server and return a response. The response simply redisplay the page, but *Hi Universe* is still in the text box, thereby maintaining the state of the text box automatically.

A glimpse at the client-side source reveals the following:

```
<HTML>
<HEAD><title>Hello World Web Page</title></HEAD>
  <body>
    <form name="Form1" method="post" action="vb.aspx" id="Form1">
      <input type="hidden"
        name="__VIEWSTATE"
        value="dDwtMTc2MjYxNDA2NTs7Pp6EUc0BOodWTOrpgefKJJjg3yEt"/>
      <input type="text"
        name="Hi"
        value="Hi Universe" id="Hi" />
      <input type="submit"
        name="Button1"
        value="Say Hi" id="Button1" />
    </form>
  </body>
</HTML>
```

68 Chapter 3

Table 3.2 ASP.NET Server Tags

SERVER TAG	MEANING
<%@ Directive %>	Directives no longer need to be the first line in the code, and many new directives may be used in a single ASP.NET file.
<tag runat="server" >	Tags that have the runat="server" attribute are server controls.
<script runat="server" >	ASP.NET subs and functions must be placed inside the server-side script tag and cannot be placed inside the <% %> tags.
<%# DataBinding %>	This is a new tag in ASP.NET. It is used to connect, or bind, to data. This will be covered in more detail in Chapter 8, "Data Access with ADO.NET"
<%-- Server Comment --%>	Allows a server-side comment to be created.
<!-- #include -->	Allow a server-side file to be included in a document.
<%= Render code %> and <% %>	Used as in-line code sections, primarily for rendering a snippet of code at the proper location in the document. Note that no functions are permitted inside <% %> tags.

The only change is that the text box now has a value of *Hi Universe*. With traditional ASP, additional code was required to get this functionality that is built into ASP.NET server controls.

Many changes have been made in the transition from ASP to ASP.NET. Table 3.2 shows server tags that are either new or have a different meaning in ASP.NET. Understanding these changes will make an ASP to ASP.NET migration more palatable.

Server Controls

A server control is a control that is programmable by writing server-side code. Server controls automatically maintain their state between calls to the server. Server controls can be easily identified by their *runat="server"* attribute. A server control must have an ID attribute to be referenced in code. ASP.NET provides two types of server controls; HTML and Web. This section looks at these controls.

HTML Server Controls

HTML server controls resemble the traditional HTML controls, except they have a *runat="server"* attribute. There is typically a one-to-one mapping of an HTML server control and the HTML tag that it renders. HTML server controls are primarily used when migrating older ASP pages to ASP.NET. For example, the following ASP page needs to be converted to ASP.NET:

```
<HTML>
  <HEAD><title>Employee Page</title></HEAD>
  <body>
    <form name="Form1" method="post" action="vb.asp" id="Form1">
      <input type="text"
        name="EmployeeName"
        id=" EmployeeName " >
      <input type="submit"
        name="SubmitButton"
        value="Submit" id=" SubmitButton" >
    </form>
  </body>
</HTML>
```

This sample page can be converted by adding the *runat="server"* attribute to the form and input tags, and removing the *action="vb.asp"* attribute on the form. The filename needs an .aspx extension. The modified Web page looks like this:

```
<HTML>
  <HEAD><title>Employee Page</title></HEAD>
  <body>
    <form name="Form1" method="post" id="Form1" runat="server">
      <input type="text"
        name="EmployeeName"
        id="EmployeeName" runat="server" >
      <input type="submit"
        name="SubmitButton"
        value="Submit" id="SubmitButton" runat="server">
    </form>
  </body>
</HTML>
```

This example shows how the use of HTML controls can ease a conversion process. If the existing tags had JavaScript events attached, those client-side events would continue to operate.

This ease of migration benefit can also be a drawback. Being HTML-centric, the object model for these controls is not consistent with other .NET controls. This is where Web server controls provide value.

70 Chapter 3

Web Server Controls

Web server controls offer more functionality than HTML controls, their object model is more consistent, and more elaborate controls are available. Web server controls are designed to provide an object model that is heavily focused on the purpose of the object rather than the HTML that is generated. In fact, the Web server control's source code will typically be substantially different from the HTML it generates. Some Web server controls, such as the Calendar and DataGrid, produce complex tables with JavaScript client-side code.

Web server controls have the ability to detect the browser capabilities and generate HTML that uses the browser to its fullest potential.

During design, a typical Web server control's source code will look like the following:

```
<asp:button attributes runat="server"/>
```

The attributes of the Web server control are properties of that control, and may or may not be attributes in the generated HTML.

Server Control Recommendations

Consider using HTML server controls when:

- Migrating existing ASP pages to ASP.NET.
- The control needs to have custom client-side script attached to the control's events.
- The Web page requires a great amount of client-side code, where client-side events need to be programmed extensively.

In all other situations, it's preferable to use Web server controls.

Server Control Event Programming

An important feature of server controls is the ability to write code that executes at the server in response to an event from the control.

ViewState

When a Web Form is rendered to the browser, a hidden HTML input tag is dynamically created, called `__VIEWSTATE` (ViewState). This input contains base64-encoded data that can be used by any object that inherits from `System.Web.UI.Control`, which represents all of the Web controls and the

Web Page object itself. ViewState is a property tag that is optimized to hold primitive type, strings, HashTables, and ArrayLists, but can also hold any object that is serializable or data types that provide a custom TypeConverter.

An object may use ViewState to persist information across calls to the server when that information cannot easily be persisted via traditional HTML attributes. In some instances, ViewState is not necessary, because the content of a control may automatically be persisted across calls to the server. For example, a TextBox automatically sends its contents back to the server via its value property, and the server can repopulate the value property when rendering it back to the browser. If, however, additional information is needed that cannot easily be represented with traditional HTML attributes, ViewState comes to the rescue.

One example of using ViewState would be a scenario where a ListBox is populated by querying a database. It may not be desirable to requery the database everytime the page is posted to the server. The ListBox uses ViewState to hold the complete list of items that are placed in the ListBox. ViewState stores the list of items that were programmatically placed into the ListBox. By placing the list of items in ViewState, the ListBox will be repopulated automatically. The server will not need to requery the database to repopulate the ListBox, because the ListBox is maintaining its own state. In the following code sample, an asp:ListBox server control has been added, as has been a subroutine to simulate loading the ListBox programmatically from a database.

```
<HTML>
  <script runat="server">
    sub Form_Load(sender as object, e as System.EventArgs) _
      handles MyBase.Load
      'simulate loading the ListBox from a database
      ListBox1.Items.Add(New ListItem("apple"))
      ListBox1.Items.Add(New ListItem("orange"))
    end sub
  </script>
  <HEAD><title>Hello World Web Page</title></HEAD>
  <body>
    <form id="Form1" method="post" runat="server">
      <asp:TextBox id="Hi" runat="server">
        Hello World
      </asp:TextBox>
      <asp:Button id="Button1" runat="server" Text="Nothing">
      </asp:Button>
      <asp:ListBox id="ListBox1" runat="server">
      </asp:ListBox>
    </form>
  </body>
</HTML>
```

72 Chapter 3

After the ListBox and code have been added, browsing to this sample page will show the ListBox, which will contain the Apple and Orange items that were added by the form load procedure. Viewing the source reveals a much larger ViewState as shown next.

```
<input type="hidden" name="__VIEWSTATE"
value="dDwtMzE3ODYxNTUzO3Q8O2w8aTwyPjs+O2w8dDw7bDxpPDU+Oz47bDx
0PHQ8O3A8bDxpPDA+O2k8MT47PjtsPHA8YXBwbGU7YXBwbGU+O3A8b3JhbmdlO2
9yYW5nZT47Pj47Pjs7Pjs+Pjs+Pjs+Gyn1i+uQFP6LoU14/8djhigkR4Q=" />
```

Correcting Multiple Entries

This page contains a button, which has not been programmed to do anything, but will cause all of the form data to be posted back to the server. If the button is clicked, the ListBox will contain Apple, Orange, Apple, and Orange. What happened?

ASP.NET will automatically rebuild the ListBox using the items that are in ViewState. Also, the form load subroutine contains code that simulates loading the ListBox from a query to a database. The result is that we end up with repeated entries in the ListBox. One of the following solutions can be applied.

Use the *IsPostBack* Property

ASP.NET provides the *IsPostBack* property of the Page object to see if the page is being requested for the first time. The first time that a page is requested, its *IsPostBack* property will be false. When data is being sent back to the server, the *IsPostBack* property will be true (see Figure 3.1).

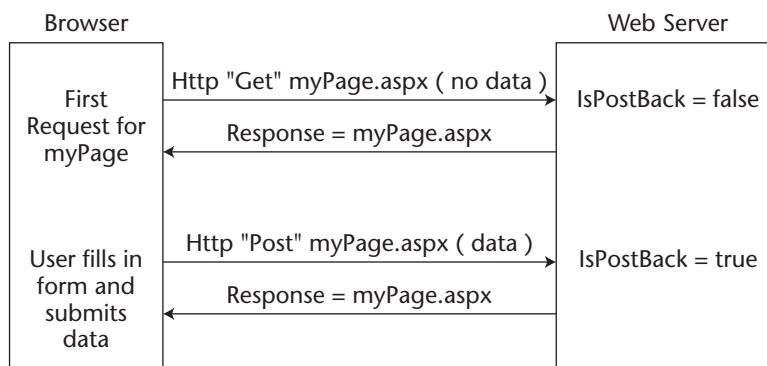


Figure 3.1 The first time that a page is requested, the *IsPostBack* property of the page is equal to false. When the page data is submitted back to the server, the *IsPostBack* property will be true.

Change the form load subroutine by adding a condition that checks to see if the page is being loaded for the first time, and if so, load the TextBox from the database. If not, use the ViewState to populate the ListBox. Here is a sample:

```
sub Form_Load(sender as object, e as System.EventArgs) _
    handles MyBase.Load
    'simulate loading the ListBox from a database
    If not IsPostBack then
        ListBox1.Items.Add(New ListItem("apple"))
        ListBox1.Items.Add(New ListItem("orange"))
    End if
end sub
```

This routine uses the IsPostBack method to see if the page is being posted back. If true, then there is no need to load the information from the database.

Turn off ViewState

It may more desirable to requery the database, especially if the data changes regularly. In this example, instead of turning off the query to the database, the ViewState can be turned off. Here is a sample:

```
<asp:ListBox id="ListBox1" EnableViewState="False">
</asp:ListBox>
```

Turning off ViewState for this control reduces the size of the data that ViewState passes to and from the server.

Post Back

In the previous examples, all ASP.NET server controls were encapsulated in a form that has the *runat="server"* attribute. This is a requirement. Also notice that the original form tag in the source code is:

```
<form id="Form1" method="post" runat="server">
```

A view of the client source reveals that the form tag was transformed to:

```
<form name="Form1" method="post" action="vb.aspx" id="Form1">
```

Notice that the action attribute is not valid in the original source, but ASP.NET adds the *action="vb.aspx"* attribute, where vb.aspx is the name of the current page. In essence, the page will always post back to itself.

74 Chapter 3

Each server control has the ability to be configured to submit, or post, the form data back to the server. For the TextBox, `AutoPostBack` is set to false by default, which means that the text is not sent back to the server until a different control posts the data back to the server. If `AutoPostBack` is set to true and the text is changed, then the text box will automatically post the form data back to the server when the text box loses focus. The following line shows how to turn on the `AutoPostBack` feature for the text box.

```
<asp:TextBox id="Hi" runat="server" AutoPostBack="True">
```

In many cases, the default behavior for the TextBox is appropriate. The ListBox and DropDownList also have their `AutoPostBack` set to false. But it may be desirable to change `AutoPostBack` to true. When set to true, the ListBox and DropDownList will post back to the server when a selection is made.

Responding to Events

`AutoPostBack` is great, but usually something needs to be accomplished with the data that is posted back to the server. This is where events come in. Using the single-page model, event-handling code can be added into the .aspx page to respond to an event such as the click of a button or the changing of a selection in a ListBox. The following syntax is used:

```
<control id="myctl" runat="server" event="ProcName">
```

The `event="ProcName"` attribute defines the name of a procedure that will be executed with the event is raised. The attribute creates a link, or Event Handler, to connect the control to the procedure that will be executed.

In the following example, the `lblDateTime` label control is populated with the current date and time when `btnSelect` is clicked.

```
<HTML>
  <HEAD>
    <title>Hello World Web Page</title>
    <script runat="server">
      sub ShowDateTime(sender as object, e as System.EventArgs)
        lblDateTime.Text = DateTime.Now
      end sub
    </script>
  </HEAD>
  <body>
    <form id="Form1" method="post" runat="server">
      <asp:label id="lblDateTime"
        runat="server">
      </asp:label>
```

```

        <asp:button id="btnSelect" Text="Select"
            Runat="server"
            OnClick="ShowDateTime">
        </asp:button>
    </form>
</body>
</HTML>

```

The previous example works exactly as expected, because `AutoPostBack` defaults to `true` for buttons. Controls that do not have their `AutoPostBack` attribute set to `true` will not execute their event handler code until a control posts back to the server. In the following example, `lstFruit` has been programmed to populate `txtSelectedFruit` when `SelectedIndexChanged` has occurred.

```

<HTML>
  <HEAD>
    <title>Hello World Web Page</title>
    <script runat="server">
      sub ShowDateTime(sender as object, e as System.EventArgs)
        lblDateTime.Text = DateTime.Now
      end sub
      sub FruitSelected(sender as object,
        e as System.EventArgs)
        txtSelectedFruit.Text = lstFruit.SelectedItem.Value
      end sub
      sub Form_Load(sender as object, e as System.EventArgs) _
        handles MyBase.Load
        if not IsPostBack then
          'simulate loading the ListBox from a database
          lstFruit.Items.Add(New ListItem("apple"))
          lstFruit.Items.Add(New ListItem("orange"))
        end if
      end sub
    </script>
  </HEAD>
  <body>
    <form id="Form1" method="post" runat="server">
      <asp:label id="lblDateTime"
        runat="server">
      </asp:label>
      <asp:textbox id="txtSelectedFruit"
        runat="server">Hello World
      </asp:textbox>
      <asp:listbox id="lstFruit" Runat="server"
        OnSelectedIndexChanged="FruitSelected">
      </asp:listbox>
      <asp:button id="btnSelect" Text="Select"
        Runat="server"

```

76 Chapter 3

```
        OnClick="ShowDateTime">
    </asp:button>
</form>
</body>
</HTML>
```

In the previous example, selecting a fruit did not update the txtSelectedFruit TextBox. If the button is clicked, the txtSelectedFruit TextBox will be updated, because the button will post all of the Web Form's data back to the server, and the server will detect that the selected index has changed on the lstFruit ListBox.

Although this behavior may be okay in some solutions, in other solutions it may be more desirable to update the txtSelectedFruit TextBox immediately upon change of the lstFruit selection. This can be done by adding *AutoPostBack="true"* to the lstFruit control.

Event Handler Procedure Arguments

All events in the Web Forms environment have been standardized to have two arguments. The first argument, *sender as object*, represents the object that triggered, or raised, the event. The second argument, *e as EventArgs*, represents an EventArgs object or an object that derives from EventArgs. By itself, the EventArgs object is used when there are no additional arguments to be passed to the event handler. In essence, if EventArgs is used as the second argument, then there is no additional data being sent to the event handler. If custom arguments need to be passed to the event handler, a new class is created that inherits the EventArgs class and adds the appropriate data.

Examples of some of the custom argument classes that already exist are ImageClickEventArgs, which contains the x and y coordinates of a click on an ImageButton control, and DataGridItemEventArgs, which contains all of the information related to the row of data in a DataGrid control. Events will be looked at more closely in Chapter 4, "The .NET Framework and Visual Basic .NET Object Programming."

Code-Behind Page

The two page model for designing Web Forms uses a *Web Forms page* (with an .aspx extension) for visual elements that will be displayed at the browser, and a *code-behind page* (with the .vb extension for Visual Basic .NET) for the code that will execute at the server. When a new WebForm is added to an ASP.NET project using Visual Studio .NET, it will always be the two-page model type.

With the two-page model, all of the code-behind pages must be compiled into a single .dll file for the project. Each code-behind page contains a class that derives from `System.Web.UI.Page`. The `System.Web.UI.Page` class contains the functionality to provide context and rendering of the page.

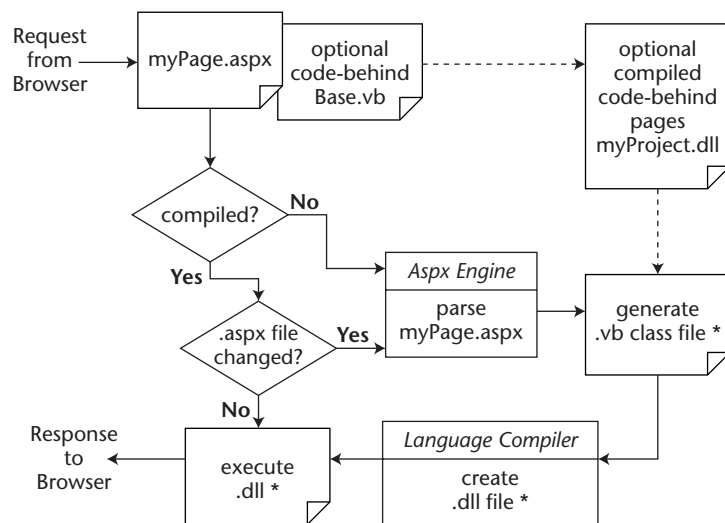
The Web Form page is not compiled until a user requests the page from a browser (see Figure 3.2).

The Web Form page is then converted to a class that inherits from the code-behind class. Then, the class is compiled, stored to disk, and executed. Once the Web Form page has been compiled, additional requests for the same Web Form page will execute the page's .dll code without requiring another compile. If the .aspx file has been changed, the .aspx file will be reparsed and recompiled.

The connection of the Web Form page and the code-behind page is accomplished by adding additional attributes to the Web Form page's Page directive, as in the following:

```
<%@ Page Language="vb" Codebehind="myPage.aspx.vb"
    Inherits="ch3.myPage"%>
```

The *Codebehind* attribute identifies the filename of the code-behind page. The *Inherits* attribute identifies the class that the Web Form page will inherit from, which is in the code-behind page.



* Files created within the following folder structure:
 %SystemRoot%\Microsoft.NET\Framework\version\Temporary ASP.NET Files\

Figure 3.2 A Web page is dynamically compiled, as shown in this diagram, when a user navigates to the page for the first time.

78 Chapter 3

In Visual Studio .NET, when a Web Form is created, Visual Studio .NET automatically creates the Web Form page, which has the .aspx extension, and the code-behind page, which has the .aspx.vb extension. The code-behind page will not be visible until the Show All Files button is clicked in the Solution Explorer.

Accessing Controls and Events on the Code-Behind Page

In Visual Studio .NET, when a control is dragged and dropped onto the Web Form page, a matching control variable is defined inside the code-behind class. This control contains all of the properties, methods, and events that belong to the control that is rendered on to the Web Form page (see Figure 3.3).

The following code is created in the Web Form page when a new page is created in Visual Studio .NET called myPage. A TextBox and Button are added, and code is added that displays the current date and time in the TextBox when the button is clicked.

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="myPage.aspx.vb"
Inherits="ch3.myPage"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD>
    <title>myPage</title>
    <meta name="GENERATOR"
      content="Microsoft Visual Studio .NET 7.0">
    <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
    <meta name="vs_defaultClientScript" content="JavaScript">
    <meta name="vs_targetSchema"
      content="http://schemas.microsoft.com/intellisense/ie5">
  </HEAD>
  <body MS_POSITIONING="GridLayout">
    <form id="Form1" method="post" runat="server">
      'positioning style elements removed for clarity
      <asp:TextBox id="TextBox1"
        runat="server">
      </asp:TextBox>
      <asp:Button id="Button1"
        runat="server" Text="Button">
      </asp:Button>
    </form>
  </body>
</HTML>
```


Notice that there is no server-side code in this page. All server-side code is packed into the code-behind page. The following is a code listing of the code-behind class.

```
Public Class myPage
    Inherits System.Web.UI.Page
    Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button
    #Region " Web Form Designer Generated Code "
    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
    End Sub
    Private Sub Page_Init(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the
        'Web Form Designer
        'Do not modify it using the Code Editor.
        InitializeComponent()
    End Sub
    #End Region
    Private Sub Page_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here.
    End Sub
    Private Sub Button1_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles Button1.Click
        TextBox1.Text = DateTime.Now
    End Sub
End Class
```

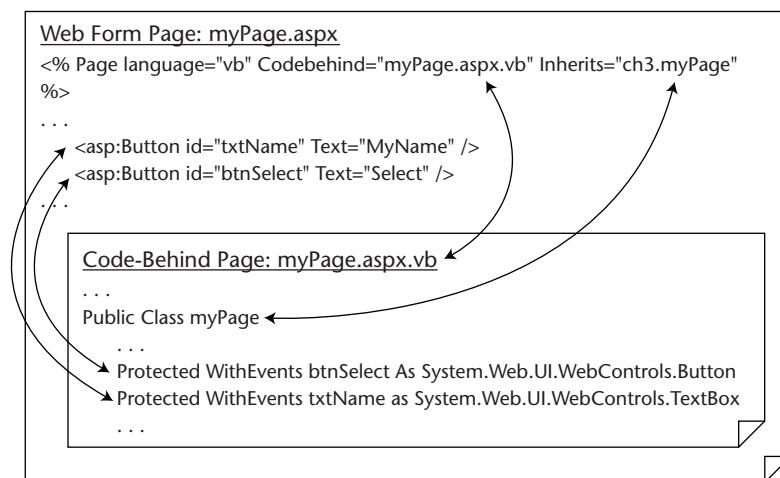


Figure 3.3 The code-behind page contains matching objects, which gives the code the ability to access the control from within the code-behind page.

80 Chapter 3

Button1's event handler code is connected to Button1's click event by the *Handles Button1.Click* tag at the end of the Button1_Click subprocedure. The Button1_Click subprocedure can be renamed without losing the connection between the Web Form page and the code-behind page. For example, if two buttons are programmed to execute the subprocedure, it may be more beneficial to rename the subprocedure to something that is more generic. Additional events can be added to the Handles keyword, separated by commas. The following code snippet shows how Button2's click event can execute the same procedure.

```
Private Sub Clicked(ByVal sender As System.Object,  
    ByVal e As System.EventArgs) _  
    Handles Button1.Click, Button2.Click
```

Visual Studio .NET also exposes all events that are available for a given control. Figure 3.4 shows the code window, which has a class selection drop-down list and an event method drop-down list. Selecting an event will generate template code inside the code-behind page for the event.

Web Form Designer Generated Code

The code-behind page contains a region called Web Form Designer Generated Code. This region is controlled by the Web Form Designer, which can be opened to reveal the code that the Web Form Designer generates. Exploring and understanding this region can be beneficial. If changes to the code that is in the region are required, it is best to make the changes through the Web Form Designer.

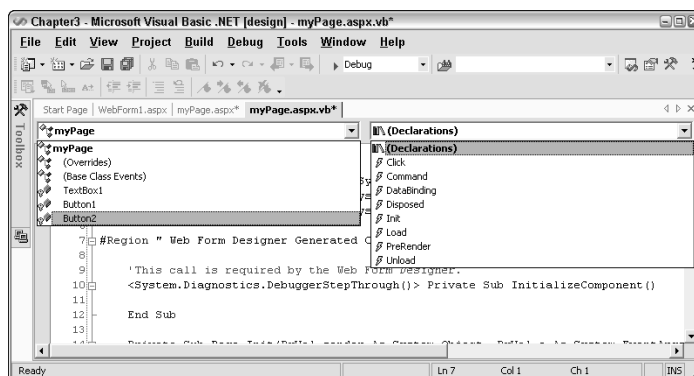


Figure 3.4 The class and event method selection lists are shown. First select an item from the class list and then select an event method. This will add template code for the method, if it doesn't exist.

Life Cycle of a Web Form and Its Controls

It's important to understand the life cycle of a Web Form and its controls. Every time a browser hits a Web site, the browser is requesting a page. The Web server constructs the page, sends the page to the browser, and destroys the page. Pages are destroyed to free up resources. This allows the Web server to scale nicely, but poses problems with maintaining state between calls to the server. The use of ViewState allows the state to be sent to the browser. Posting the entire Web Form's data, including ViewState, back to the server allows the previous state to be reconstructed to recognize data that has changed between calls to the server.

All server controls have a series of methods and events that execute as the page is being created and destroyed. The Web page derives from the Control class as well, so the page also executes the same methods and events as it is being created and destroyed. Table 3.3 contains a description of the events that take place when a page is requested, paying particular attention to ViewState and its availability.

Table 3.3 Page/Control Life Cycle Method and Events

PAGE/CONTROL METHOD AND (EVENT)	DESCRIPTION
OnInit (Init)	Each control is initialized.
LoadViewState	Loads the ViewState of the control.
LoadPostData	Retrieves the incoming form data and updates the control's properties accordingly.
Load (OnLoad)	Actions that are common to every request can be place here.
RaisePostDataChangedEvent	Raises change events in response to the postback data changing between the current postback and the previous postback. For example, if a TextBox has a TextChanged event and AutoPostBack is turned off, clicking a Button causes the TextChanged event to execute in this stage before handling the click event of the button (next stage).
RaisePostBackEvent	Handles the client-side event that caused the postback to occur.
PreRender (OnPreRender)	Allows last minute changes to the control. This event takes place after all regular postback events have taken place. Since this event takes place before saving ViewState, any changes made here will be saved.

(continued)

82 Chapter 3

Table 3.3 (continued)

PAGE/CONTROL METHOD AND (EVENT)	DESCRIPTION
SaveViewState	Saves the current state of the control to ViewState. After this stage, any changes to the control will be lost.
Render	Generates the client-side HTML, DHTML, and script that are necessary to properly display this control at the browser. In this stage, any changes to the control are not persisted into ViewState.
Dispose	Cleanup code goes here. Releases any unmanaged resources in this stage. Unmanaged resources are resources that are not handled by the .NET common language runtime, such as file handles and database connections.
Unload	Cleanup code goes here. Releases any managed resources in this stage. Managed resources are resources that are handled by the runtime, such as instances of classes created by the .NET common language runtime.

Page Layout

Each Web Form has a `pageLayout` property, which can be set to `GridLayout` or `FlowLayout`. These layouts have different control positioning behaviors. This setting can be set at the project level, which will affect new pages that are added. The setting can also be set on each Web Form.

FlowLayout

`FlowLayout` behavior is similar to traditional ASP/HTML behavior. The controls on the page do not have dynamic positioning. When a control is added to a Web Form, it is placed in the upper-left corner. Pressing the Spacebar or Enter can push the control to the right, or downward, but this model usually uses tables to control the positioning of controls on the page.

GridLayout

`GridLayout` behavior uses dynamic positioning to set the location of a control on the page. A control can be placed anywhere on the page. This mode also

allows controls to be snapped to a grid. Behind the scenes, `GridLayout` is accomplished by adding the attribute `ms_positioning="GridLayout"` to the body tag of a Web Form.

Selecting the Proper Layout

`GridLayout` can save lots of development time, since positioning of controls does not require an underlying table structure. `GridLayout` is usually a good choice for a fixed-size form.

Since `FlowLayout` does not use absolute positioning, it can be an effective choice when working with pages that are resizable. In many cases, it is desirable to hide a control and let the controls that follow shift to move into the hole that was created.

The benefits of both layout types can be implemented on the same page by using panel controls. The panel control acts as a container for other controls. Setting the visibility of the panel to false turns off all rendered output of the panel and its contained controls. If the panel is on a page where `FlowLayout` is selected, any controls that follow the panel are shifted to fill in the hole that was created by the absence of the panel.

Figure 3.5 shows an example of a `FlowLayout` page that has two HTML Grid Layout Panels that are configured to run as HTML server controls. Web server controls were added for the Button, Labels, and TextBoxes. The `Page_Load` method is programmed to display the top Grid Layout Panel if this is the first request for the page. If data is being posted to this page, the lower Grid Layout Panel is displayed. The .aspx page contains the following HTML code:

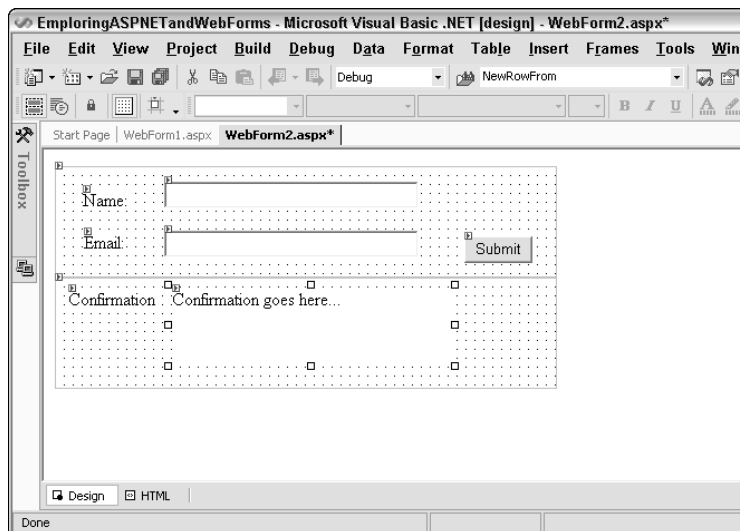


Figure 3.5 This Web page is configured for `FlowLayout` and contains two HTML Grid Layout Panels.

84 Chapter 3

```
<%@ Page
    Language="vb"
    AutoEventWireup="false"
    Codebehind="WebForm1.aspx.vb"
    Inherits="chapter3.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>WebForm1</title>
    <meta name="GENERATOR"
      content="Microsoft Visual Studio .NET 7.0">
    <meta name="CODE_LANGUAGE"
      content="Visual Basic 7.0">
    <meta name="vs_defaultClientScript"
      content="JavaScript">
    <meta name="vs_targetSchema"
      content="http://schemas.microsoft.com/intellisense/ie5">
  </head>
  <body>
    <form id="Form1" method="post" runat="server">
      <div style="WIDTH: 450px;
        POSITION: relative;
        HEIGHT: 100px"
        ms_positioning="GridLayout"
        id="TopPanel"
        runat="server">
        <asp:TextBox
          id="txtName"
          style="Z-INDEX: 101;
            LEFT: 98px;
            POSITION: absolute;
            TOP: 13px"
          runat="server"
          Width="228"
          height="24">
        </asp:TextBox>
        <asp:TextBox
          id="txtEmail"
          style="Z-INDEX: 102;
            LEFT: 98px;
            POSITION: absolute;
            TOP: 57px"
          runat="server"
          Width="228"
          height="24">
        </asp:TextBox>
        <asp:Label
          id="Label1"
          style="Z-INDEX: 103;
            LEFT: 25px;
```

```
        POSITION: absolute;
        TOP: 21px"
        runat="server">
        Name:
    </asp:Label>
    <asp:Label
        id=Label2
        style="Z-INDEX: 104;
        LEFT: 26px;
        POSITION: absolute;
        TOP: 59px"
        runat="server">
        Email:
    </asp:Label>
    <asp:Button
        id=btnSubmit
        style="Z-INDEX: 105;
        LEFT: 367px;
        POSITION: absolute;
        TOP: 63px"
        runat="server"
        Text="Submit">
    </asp:Button>
</div>
<div
    style="WIDTH: 450px;
    POSITION: relative;
    HEIGHT: 100px"
    ms_positioning="GridLayout"
    id=BottomPanel
    runat="server">
    <asp:Label
        id=lblConfirmation
        style="Z-INDEX: 101;
        LEFT: 105px;
        POSITION: absolute;
        TOP: 10px"
        runat="server"
        Width="249px"
        Height="66px">
        Confimarion goes here...
    </asp:Label>
    <asp:Label
        id=Label3
        style="Z-INDEX: 102;
        LEFT: 12px;
        POSITION: absolute;
        TOP: 10px"
        runat="server"
        Width="76px">
```

86 Chapter 3

```

        Confirmation
    </asp:Label>
</div>
</form>
</body>
</html>

```

Notice that the HTML Grid Layout Panels are nothing more than DIV tags with the *ms_positioning="GridLayout"* attribute. The other controls are contained in the DIV tags.

The code-behind page contains the following code:

```

Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Label1 As _
        System.Web.UI.WebControls.Label
    Protected WithEvents Label2 As _
        System.Web.UI.WebControls.Label
    Protected WithEvents lblConfirmation As _
        System.Web.UI.WebControls.Label
    Protected WithEvents txtName As _
        System.Web.UI.WebControls.TextBox
    Protected WithEvents txtEmail As _
        System.Web.UI.WebControls.TextBox
    Protected WithEvents Label3 As _
        System.Web.UI.WebControls.Label
    Protected WithEvents TopPanel As _
        System.Web.UI.HtmlControls.HtmlGenericControl
    Protected WithEvents BottomPanel As _
        System.Web.UI.HtmlControls.HtmlGenericControl
    Protected WithEvents btnSubmit As _
        System.Web.UI.WebControls.Button
    #Region " Web Form Designer Generated Code "
        'This call is required by the Web Form Designer.
        <System.Diagnostics.DebuggerStepThrough()> _
        Private Sub InitializeComponent()
        End Sub
        Private Sub Page_Init(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles MyBase.Init
            'CODEGEN: This method call is required by the Web Form Designer
            'Do not modify it using the code editor.
            InitializeComponent()
        End Sub
    #End Region
    Private Sub Page_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

```



```

If Page.IsPostBack Then
    TopPanel.Visible = False
    BottomPanel.Visible = True
Else
    TopPanel.Visible = True
    BottomPanel.Visible = False
End If
End Sub
Private Sub btnSubmit_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnSubmit.Click

    lblConfirmation.Text = "Hello " & txtName.Text & "<br>"
    lblConfirmation.Text &= "Your email address is " & txtEmail.Text
End Sub
End Class

```

When the page is viewed for the first time (see Figure 3.6), only the top panel is displayed. When data is entered and submitted, the top panel is hidden and the bottom panel is displayed. Since the page layout is set to FlowLayout, the bottom panel will shift to the top of the page to fill in the hole that was created by setting the top panel's visible property to false.

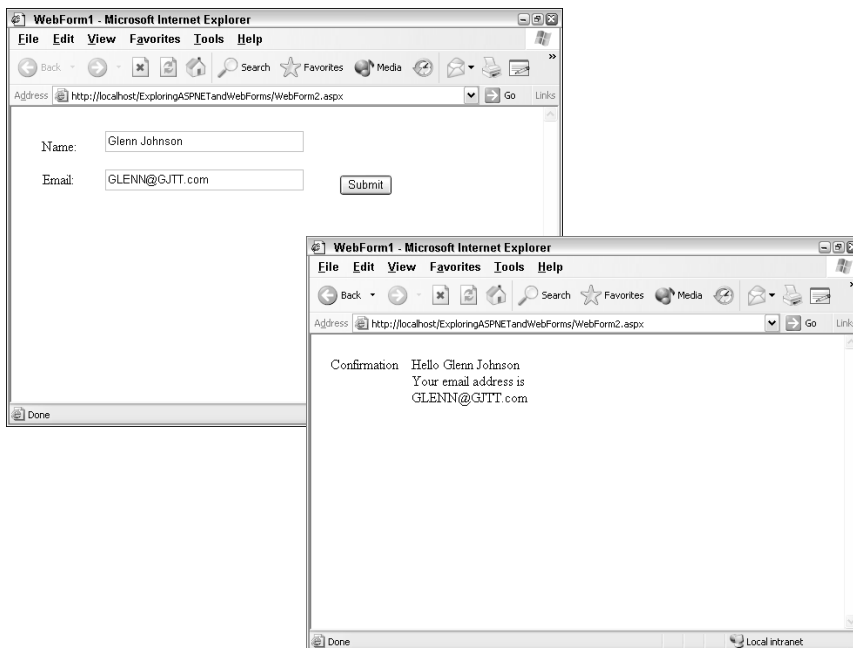


Figure 3.6 Only one panel is displayed at a time. When the first panel is hidden, the second panel moves into the space that was originally occupied by the first panel.



Lab 3.1: Web Forms

In this lab, you will create a Web Form using Visual Studio .NET and then explore the life cycle of the Web Form and its controls.

Create the Web Form

In this section, you will create a Web Form called NewCustomer.aspx, which allows you to collect customer information. Later, you will store this information in a database.

- 1. To start this lab, open the OrderEntrySolution from Lab 2.1 or Lab 2.2.
- 2. Right-click the Customer project, click Add, Add Web Form, and type NewCustomer.aspx for the name of the new Web Form. When prompted to check out the project, click the Check Out button.
- 3. Add the Web server controls in Table 3.4 to the Web Form. Figure 3.7 shows the completed page. Save your work.

Table 3.4 NewCustomer.aspx Web Server Controls

ID	TYPE	PROPERTIES
lblCustomer	asp:Label	Text=Customer Name
txtCustomerName	asp:TextBox	Text=
lblAddress	asp:Label	Text=Address
txtAddress1	asp:TextBox	Text=
txtAddress2	asp:TextBox	Text=
lblCity	asp:Label	Text=City
txtCity	asp:TextBox	Text=
lblState	asp:Label	Text=State
drpState	asp:DropDownList	Items = Enter the states below plus an empty entry as the default. Text= Value= Text=FL Value=FL Text=MA Value=MA Text=OH Value=OH Text=TX Value=TX
lblZipCode	asp:Label	Text=Zip
txtZipCode	asp:TextBox	Text=
btnAddCustomer	asp:Button	Text=Add Customer
lblConfirmation	asp:Label	Text=

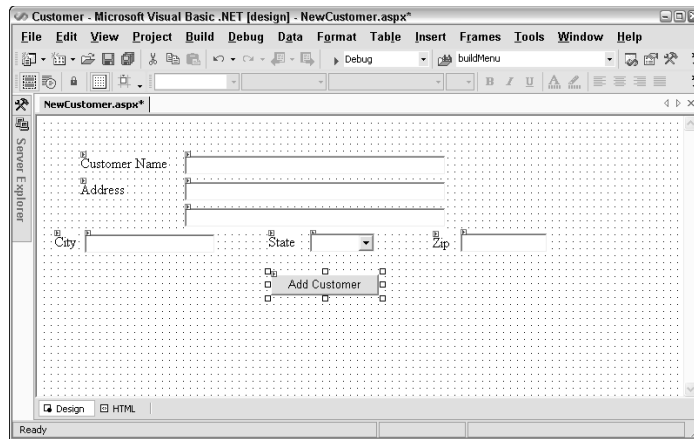


Figure 3.7 The completed Web page after entering the Web server controls in Table 3.4.

Test Your Work

Test your work by performing the following steps:

1. Compile your project. Click Build, Build Solution. You should see an indication in the output window that all three projects compiled successfully.
2. Select a startup page for the Visual Studio .NET Debugger. Locate the NewCustomer.aspx page in the Solution Explorer. Right-click NewCustomer.aspx, and then click Set As Start Page.
3. Press F5 to launch the Visual Studio .NET debugger, which will display your page in your browser.
4. Test ASP.NET's ability to maintain state. Type some text into each TextBox, select a state from the DropDownList, and click the Add Customer button.

What happened? When the button was clicked, the data was posted back to the server. If your page functioned properly, the server received the data that was entered into the Web Form. No code has been assigned to the Add Customer button's click event, so the server simply returns the page to the browser.

What is most interesting is that the data is still on the form; the TextBoxes still have the data that you typed in, and the DropDownList still has the selected state. This demonstrates ASP.NET's ability to maintain state.

90 Chapter 3

Adding Code to Process the Data

In this section, you will add some code to the Add Customer's click event. The code simply displays a summary message on the current page. This data will be put into a database in a later lab.

Double-click the Add Customer button. This opens the code-behind page and adds template code for the button's click event. Add code to the button's click event procedure so that it looks like this:

```
Private Sub btnAddCustomer_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnAddCustomer.Click
    Dim s As String
    s = "<font size='5'>Confirmation Info:</font>" & "<BR>"
    s += txtCustomerName.Text & "<BR>"
    s += txtAddress1.Text & "<BR>"
    If txtAddress2.Text.Length > 0 Then
        s += txtAddress2.Text & "<BR>"
    End If
    s += txtCity.Text & ", "
    s += drpState.SelectedItem.Text & " "
    s += txtZipCode.Text & "<BR>"
    lblConfirmation.Text = s
End Sub
```

Test Your Work

1. Save your work.
2. Press F5 to launch the Visual Studio .NET debugger, which will display your page in your browser.
3. Test ASP.NET's ability to process the data on the Web Form by entering data and then clicking the Add Customer button.
4. When the Add Customer button is clicked, the confirmation label will be populated with data from the Web Form.

Exploring ViewState

In this section, you will explore the ViewState to appreciate the need for this hidden object.

1. Add a Web server control button to the NewCustomer.aspx page. Change its ID to "btnViewState" and its Text to "ViewState Test." Don't add any code to this button's click event.
2. Press F5 to view the page.
3. View the size of the ViewState hidden object. When the page is displayed, click View, Source. Note the size of the ViewState, which should be approximately 50 characters.

4. Enter data into the Web Form, and click the Add Customer button. Note the change in the ViewState, which should be significantly larger, depending on the amount of data that was entered on the Web Form.
5. Click the ViewState Test button. Notice that the data is posted back to the server, and the information that is in the Confirmation label has not been not lost.
6. View the size of the ViewState hidden object. When the page is displayed, click View, Source. Note the size of the ViewState, which is much larger that before. ASP.NET stores the value of the Confirmation label in ViewState.

Identifying ViewState Contributors

As ViewState grows, you will need to identify the controls that are placing data into ViewState. This section will use the ASP.NET trace function to identify the objects that are using ViewState.

1. Open the Web.Config file. This is an XML file that contains settings for the Web site.
2. Locate the following trace element:

```
<trace
    enabled="false"
    requestLimit="10"
    pageOutput="false"
    traceMode="SortByTime"
    localOnly="true"
/>
```

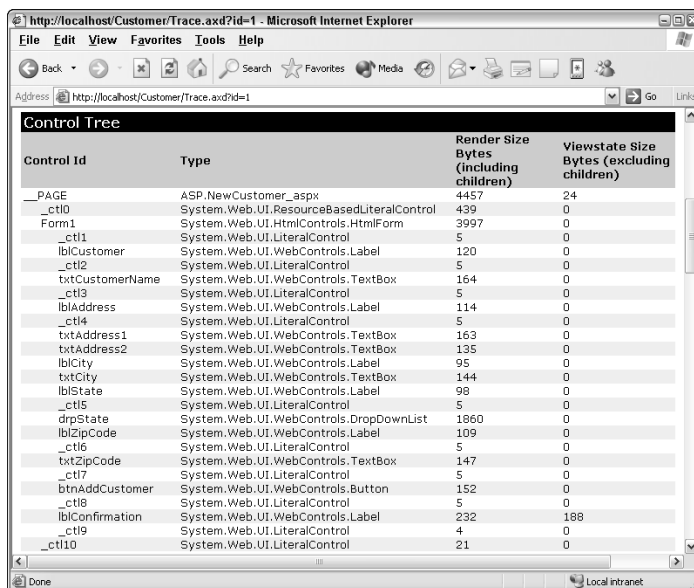
3. Make the following changes:

```
<trace
    enabled="true"
    requestLimit="100"
    pageOutput="false"
    traceMode="SortByTime"
    localOnly="true"
/>
```

4. Save the Web.Config file.
5. Press F5 to view the page.
6. Enter data into the Web Form, and click the Add Customer button. Note the change in the ViewState, which should be significantly larger, depending on the amount of data that was entered on the Web Form.

92 Chapter 3

7. Click the ViewState Test button. Notice that the data is posted back to the server, and the information that is in the Confirmation label has not been not lost.
8. Change the URL from `http://localhost/Customr/NewCustomer.aspx` to `http://localhost/Customr/trace.axd` and press Enter. The trace page is displayed. The trace page has an entry for each time you requested the NewCustomer.aspx page. Notice that the first time the page was requested, a GET was performed. Each additional page request resulted in a POST of data back to the page.
9. Click the View Details link of the first page request. This page contains lots of information. Locate the Control Tree section, which shows all of the controls that are on the page and the quantity of bytes that each control has placed into ViewState. On the first request for the page, only the page itself has contributed to ViewState (typically 20 bytes). The page automatically stores globalization information in ViewState.
10. Click the Back button in the browser, and then click the View Details link of the second request. Locate the Control Tree section. Notice that the page still contributes the same quantity of bytes to ViewState, and the `lblConfirmation` (Confirmation Label) contributes many bytes of data to ViewState, depending on the size of the data that needed to be remembered (see Figure 3.8).



Control Id	Type	Render Size Bytes (including children)	Viewstate Size Bytes (excluding children)
_PAGE	ASP.NewCustomer.aspx	4457	24
_ctl0	System.Web.UI.ResourceBasedLiteralControl	439	0
Form1	System.Web.UI.HtmlControls.HtmlForm	3997	0
_ctl1	System.Web.UI.LiteralControl	5	0
lblCustomer	System.Web.UI.WebControls.Label	120	0
_ctl2	System.Web.UI.LiteralControl	5	0
txtCustomerName	System.Web.UI.WebControls.TextBox	164	0
_ctl3	System.Web.UI.LiteralControl	5	0
lblAddress	System.Web.UI.WebControls.Label	114	0
_ctl4	System.Web.UI.LiteralControl	5	0
txtAddress1	System.Web.UI.WebControls.TextBox	163	0
txtAddress2	System.Web.UI.WebControls.TextBox	135	0
lblCity	System.Web.UI.WebControls.Label	95	0
txtCity	System.Web.UI.WebControls.TextBox	144	0
lblState	System.Web.UI.WebControls.Label	98	0
_ctl5	System.Web.UI.LiteralControl	5	0
drpState	System.Web.UI.WebControls.DropDownList	1860	0
lblZipCode	System.Web.UI.WebControls.Label	109	0
_ctl6	System.Web.UI.LiteralControl	5	0
txtZipCode	System.Web.UI.WebControls.TextBox	147	0
_ctl7	System.Web.UI.LiteralControl	5	0
btnAddCustomer	System.Web.UI.WebControls.Button	152	0
_ctl8	System.Web.UI.LiteralControl	5	0
lblConfirmation	System.Web.UI.WebControls.Label	232	188
_ctl9	System.Web.UI.LiteralControl	4	0
_ctl10	System.Web.UI.LiteralControl	21	0

Figure 3.8 Use Trace to identify ViewState contributors. Notice that the page always contributes approximately 20 bytes, and that the confirmation label contributes many bytes to ViewState, depending on the amount of data that is in the label.

Understanding the Page Life Cycle (Optional)

This section will help you understand the page's life cycle by adding code to some of the page's significant events.

1. Close all open files.
2. Open the WebForm1.aspx file that is located in the Customer project.
3. Add a TextBox and a Button to the page from the Web Forms tab of the ToolBox. When prompted to check out files from Visual Source-Safe, click the Check Out button.
4. Double-click the button to go to the code-behind page.
5. Add the following code to the Button1_Click event method.

```
Response.Write("Button Clicked<br>")
```

6. The upper part of the code window contains two drop-down boxes. The first drop-down box is used to select a class, and the second drop-down box is used to select an event method. Select TextBox1 from the class drop-down list, and select TextChanged from the event method drop-down list.
7. Add the following code to the TextBox1_TextChanged event method.

```
Response.Write("Text Changed<br>")
```

8. In the Page_Load subroutine, add the following code:

```
Response.Write("Page_Load")
```

9. With WebForm1 selected from the class drop-down list, select the Page_Init event method. Add a Response.Write method as you did in the previous steps.
10. Select Base Class Events from the class drop-down list and select the PreRender event method. Add Response.Write code as you did in the previous steps.
11. In the Solution Explorer, right-click WebForm1.aspx and click Set as Start Page. Press F5 to see the page. The page will display a message indicating that the Page Init, Page Load, and PreRender events took place.
12. Enter some information into the TextBox, and click the Button. The page will display a message indicating that the Page Init, Page Load, Text Changed, Button Clicked, and PreRender events took place. Although AutoPostBack is set to false on the TextBox, the TextChanged still executes, but not until a posting control, such as the Button, caused the data to be posted back to the server.

94 Chapter 3

Summary

- ASP.NET supports the traditional single-page programming model. It also provides the two-page coding model, which utilizes the code-behind page for the separation of client-side and server-side code.
- ASP.NET provides two types of server controls: HTML server controls and Web server controls.
- HTML server controls are used when migrating existing ASP pages to ASP.NET because a `runat="server"` attribute can be easily added to an HTML tag to convert it to an HTML server control.
- Web server controls are the preferred controls for new projects because of their consistent programming model and their ability to provide browser-specific code. ASP.Net provides Web server controls that can produce many lines of complex HTML output to accomplish a task rather than the one-to-one mapping that exists when using Web server controls.
- Use the `Page.IsPostBack` property to see if this is the first time that the page has been requested.
- Controls such as the `DropDownList` and the `ListBox` have their `AutoPostBack` property set to `false`. This setting can be changed to `true` to post back to the server each time a new item is selected.
- Events in ASP.NET pass two arguments: the sender and the `EventArgs`. The sender is the object that raised the event and the `EventArgs` may contain extra data, such as the x and y coordinates of the mouse.

Review Questions

1. What are the two types of controls that ASP.NET provides?
2. What would be the best controls to use when migrating an existing ASP page to ASP.NET?
3. What is the best control to use when client-side JavaScript code will be executing from a control's events?
4. Name some benefits to using Web server controls.
5. A user complains that each time a button is pressed on the Web page, another copy of the data in a ListBox is being added to the ListBox. What is the problem? How can it be corrected?
6. You added a DropDownList to a Web page. You programmed the DropDownList to do a database lookup as soon as a new item is selected from the list. Although you wrote the code to do the lookup, selecting a new item from the list doesn't appear to work. After investigating further, you find that the lookup works, but not until a button on the form is clicked. What is the most likely problem?
7. What is the key benefit to using code-behind pages?

Answers to Review Questions

- 1.** HTML server controls and Web server controls.
- 2.** HTML server controls, because existing HTML tags can be converted to HTML server controls by adding the `runat="server"` attribute.
- 3.** HTML server controls, because it is simple to attach client-side code to these controls using traditional HTML and DHTML methods.
- 4.** Web server controls have the following benefits:
 - a.** A more consistent programming model.
 - b.** A single control can create complex HTML output.
 - c.** They produce browser-specific HTML code, taking advantage of the browser's capabilities.
- 5.** The data is being programmatically added to the `ListBox`, using code that is in the `Page_Load` event method. Since the `ListBox` remembers its data (via `ViewState`) between calls to the server, each time the page is requested, another copy of the data is added to existing data. To solve the problem, check to see if the page is being posted back to the server using the `Page.IsPostBack` property. If so, there is no need to repopulate the `ListBox`.
- 6.** The default setting of `AutoPostBack` is set to `false` on the `DropDownList` control.
- 7.** Code-behind pages provide the ability to separate client-side and server-side code.