FOR PUBLIC RELEASE

FOURTEEN

ASP.NET and Web Forms

An important part of .NET is its use in creating Web applications through a technology known as ASP.NET. Far more than an incremental enhancement to Active Server Pages (ASP), the new technology is a unified Web development platform that greatly simplifies the implementation of sophisticated Web applications. In this chapter we introduce the fundamentals of ASP.NET and cover Web Forms, which make it easy to develop interactive Web sites. In Chapter 15 we cover Web services, which enable the development of collaborative Web applications that span heterogeneous systems.

What Is ASP.NET?

We begin our exploration of ASP.NET by looking at a very simple Web application. Along the way we will establish a testbed for ASP.NET programming, and we will review some of the fundamentals of Web processing. Our little example will reveal some of the challenges in developing Web applications, and we can then appreciate the features and benefits of ASP.NET, which we will elaborate in the rest of the chapter.

Web Application Fundamentals

A Web application consists of document and code pages in various formats. The simplest kind of document is a static HTML page, which contains information that will be formatted and displayed by a Web browser. An HTML page may also contain hyperlinks to other HTML pages. A hyperlink (or just *link*) contains an address, or a Uniform Resource Locator (URL), specifying where the target document is located. The resulting combination of content and links is sometimes called *hypertext* and provides easy navigation to a vast amount of information on the World Wide Web.

SETTING UP THE WEB EXAMPLES

As usual, all the example programs for this chapter are in the chapter folder. To run the examples, you will need to have Internet Information Services (IIS) installed on your system. IIS is installed by default with Windows 2000 Server. You will have to explicitly install it with Windows 2000 Workstation. Once installed, you can access the documentation on IIS through Internet Explorer via the URL **http://localhost**, which will redirect you to the starting IIS documentation page, as illustrated in Figure 14–1.

The management tool for IIS is a Microsoft Management Console (MMC) snap-in, the Internet Services Manager, which you can find under Administrative Tools in the Control Panel. Figure 14–2 shows the main window of the Internet Services Manager. You can start and stop the Web server and perform other tasks by right-clicking on Default Web Site. Choosing Properties from the context menu will let you perform a number of configurations on the Web server.

The default home directory for publishing Web files is **\Inetpub\wwwroot** on the drive where Windows is installed. You can change this home directory using Internet Services Manager. You can access Web pages

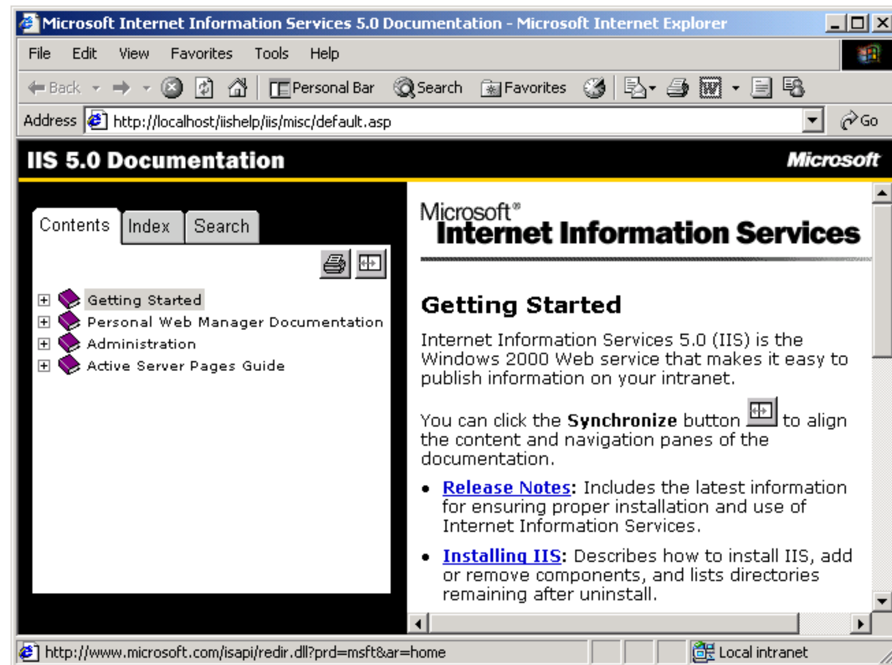


FIGURE 14–1 *Internet Information Services documentation.*

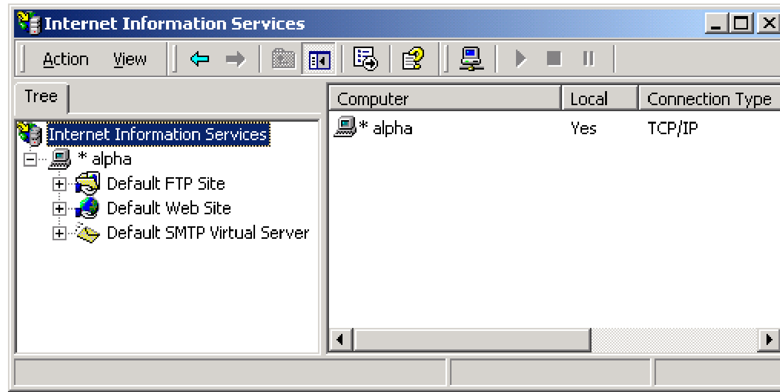


FIGURE 14–2 *Internet Services Manager.*

stored at any location on your hard drive by creating a virtual directory. The easiest way to create one is from Windows Explorer. Right-click over the desired directory, choose Sharing..., select the Web Sharing tab, click on the Add button, and enter the desired alias, which will be the name of the virtual directory. Figure 14–3 illustrates creating an alias **Chap14**, or virtual directory, for the folder **\OI\NetVb\Chap14**. You should perform this operation now on your own system in order that you may follow along as the chapter's examples are discussed.

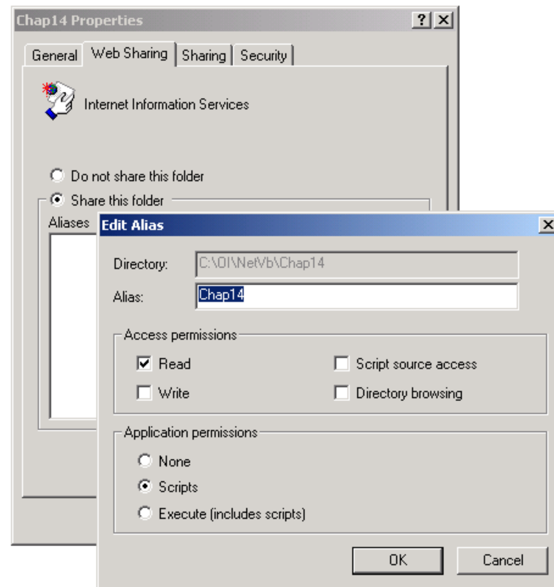
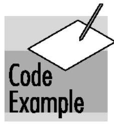


FIGURE 14–3 *Creating a virtual directory.*

ch14.fm Page 590 Wednesday, May 22, 2002 1:38 PM



Once a virtual directory has been created, you can access files in it by including the virtual directory in the path of the URL. In particular, you can access the file **default.htm** using the URL **http://localhost/Chap14/**. The file **default.htm** contains a home page for all the ASP.NET example programs for this chapter. See Figure 14–4.

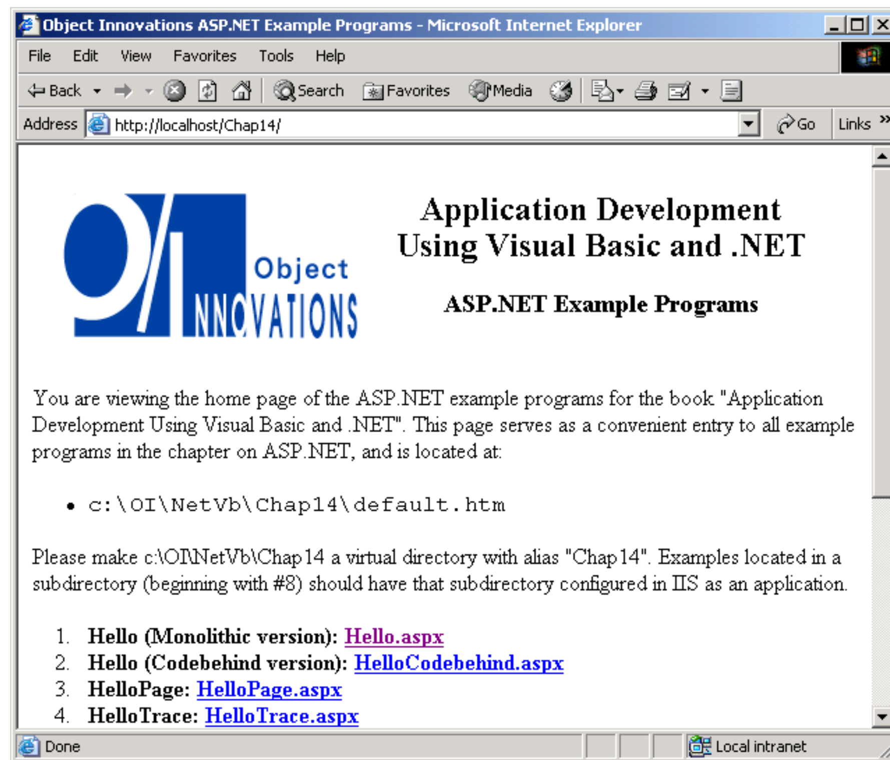
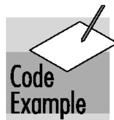


FIGURE 14–4 *Home page for ASP.NET example programs.*

An Echo Program



The first example program for this chapter is **Hello.aspx**, shown as a link on the home page. The example is complete in one file and contains embedded server code. Here is the source code, which consists of HTML along with some VB.NET script code. There are also some special tags for "server controls," recognized by ASP.NET.

```
<!-- Hello.aspx -->
<%@ Page Language="VB" %>
<HTML>
```

```
<HEAD>
   <SCRIPT RUNAT="SERVER">
   Sub cmdEcho_Click(Source As Object, e As EventArgs)
      lblGreeting.Text="Hello, " & txtName.Text
   End Sub
   </SCRIPT>
</HEAD>
<BODY>
<FORM RUNAT="SERVER">Your name: 
<asp:textbox id=txtName Runat="server"></asp:textbox>
<p><asp:button id=cmdEcho onclick=cmdEcho_Click Text="Echo"
runat="server" tooltip="Click to echo your name">
</asp:button></p>
<asp:label id=lblGreeting runat="server"></asp:label>
<P></P>
</FORM>
</BODY>
</HTML>
```

You can run the program using the URL **http://localhost/Chap14/Hello.aspx** or by clicking on the link **Hello.aspx** in the home page of the examples programs. The page shows a text box where you can type in your name, and there is an “Echo” button. Clicking the button will echo your name back, with a “Hello” greeting. The simple form is again displayed, so you could try out other names. If you slide the browser’s mouse cursor over the button, you will see the tool tip “Click to echo your name” displayed in a yellow box. Figure 14–5 illustrates a run of this example.

This little program would not be completely trivial to implement with other Web application tools, including ASP. The key user-interface feature of such an application is its thoroughly forms-based nature. The user is presented with a form and interacts with the form. The server does some processing, and the user continues to see the same form. This UI model is second nature in desktop applications but is not so common in Web applications. Typically the Web server will send back a different page.

This kind of application could certainly be implemented using a technology like ASP, but the code would be a little ugly. The server would need to synthesize a new page that looked like the old page, creating the HTML tags for the original page, plus extra information sent back (such as the greeting shown at the bottom in our echo example). A mechanism is needed to remember the current data that is displayed in the controls in the form.

Another feature of this Web application is that it does some client-side processing too—the “tooltip” displayed in the yellow box is performed by the browser. Such rich client-side processing can be performed by some browsers, such as Internet Explorer, but not others.

ch14.fm Page 592 Wednesday, May 22, 2002 1:38 PM

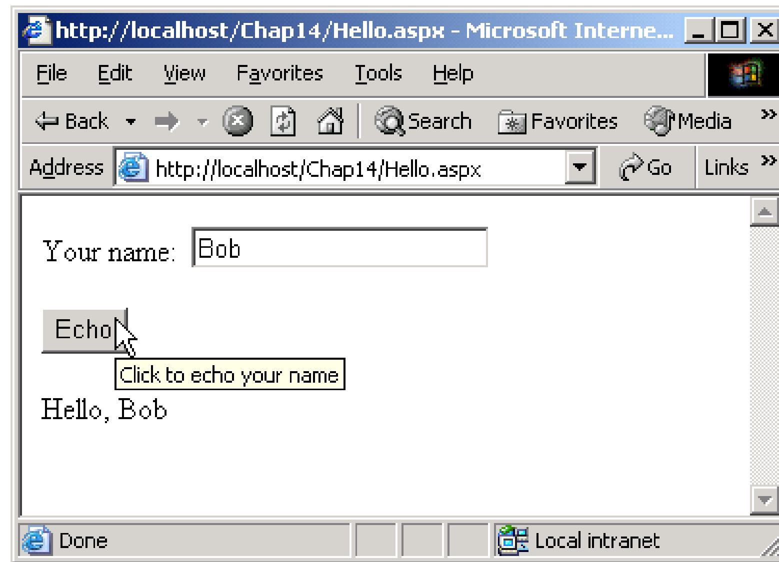


FIGURE 14–5 *Running the **Hello.aspx** echo program.*

As can be seen by the example code, with ASP.NET it is very easy to implement this kind of Web application. We will study the code in detail later. For now, just observe how easy it is!

ASP.NET Features

ASP.NET provides a programming model and infrastructure that facilitates developing new classes of Web applications. Part of this infrastructure is the .NET runtime and framework. Server-side code is written in .NET compiled languages. Two main programming models are supported by ASP.NET.

- Web Forms helps you build form-based Web pages. A WYSIWYG development environment enables you to drag controls onto Web pages. Special "server-side" controls present the programmer with an event model similar to what is provided by controls in ordinary Windows programming. This chapter discusses Web Forms in detail.
- Web services make it possible for a Web site to expose functionality via an API that can be called remotely by other applications. Data is exchanged using standard Web protocols and formats such as HTTP and XML, which will cross firewalls. We will discuss Web services in the next chapter.

Both Web Forms and Web services can take advantage of the facilities provided by .NET, such as the compiled code and .NET runtime. In addition,

ASP.NET itself provides a number of infrastructure services, including state management, security, configuration, caching, and tracing.

COMPILED CODE

Web Forms (and Web services) can be written in any .NET language that runs on top of the CLR, including C#, VB.NET, and C++ with Managed Extensions. This code is compiled, and thus offers better performance than ASP pages with code written in an interpreted scripting language such as VBScript. All of the benefits, such as a managed execution environment, are available to this code, and of course the entire .NET Framework class library is available. Legacy unmanaged code can be called through the .NET interoperability services, which are discussed in Chapter 17.

SERVER CONTROLS

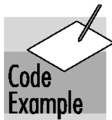
ASP.NET provides a significant innovation known as *server controls*. These controls have special tags such as <asp:textbox>. Server-side code interacts with these controls, and the ASP.NET runtime generates straight HTML that is sent to the Web browser. The result is a programming model that is easy to use and yet produces standard HTML that can run in any browser.

BROWSER INDEPENDENCE

Although the World Wide Web is built on standards, the unfortunate fact of life is that browsers are not compatible and have special features. A Web page designer then has the unattractive options of either writing to a lowest common denominator of browser, or else writing special code for different browsers. Server controls help remove some of this pain. ASP.NET takes care of browser compatibility issues when it generates code for a server control. If the requesting browser is upscale, the generated HTML can take advantage of these features, otherwise the generated code will be vanilla HTML. ASP.NET takes care of detecting the type of browser.

SEPARATION OF CODE AND CONTENT

Typical ASP pages have a mixture of scripting code interspersed with HTML elements. In ASP.NET there is a clean separation between code and presentation content. The server code can be isolated within a single <SCRIPT RUNAT="SERVER"> ... /SCRIPT> block or, even better, placed within a "code-behind" page. We will discuss code-behind pages later in this chapter. If you would like to see an example right away, you can examine the second example program **HelloCodebehind.aspx**, with code in the file **HelloCodebehind.aspx.vb**. (These files are in the top-level chapter directory.)



ch14.fm Page 594 Wednesday, May 22, 2002 1:38 PM

STATE MANAGEMENT

HTTP is a stateless protocol. Thus, if a user enters information in various controls on a form and sends this filled-out form to the server, the information will be lost if the form is displayed again, unless the Web application provides special code to preserve this state. ASP.NET makes this kind of state preservation totally transparent. There are also convenient facilities for managing other types of session and application state.

Web Forms Architecture

A Web Form consists of two parts:

- The visual content or presentation, typically specified by HTML elements.
- Code that contains the logic for interacting with the visual elements.

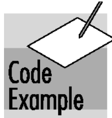
A Web Form is physically expressed by a file with the extension **.aspx**. Any HTML page could be renamed to have this extension and could be accessed using the new extension with identical results to the original. Thus Web Forms are upwardly compatible with HTML pages.

The way code can be separated from the form is what makes a Web Form special. This code can be either in a separate file (having an extension corresponding to a .NET language, such as **.vb** for VB.NET) or in the **.aspx** file, within a <SCRIPT RUNAT="SERVER"> ... /SCRIPT> block. When your page is run in the Web server, the user interface code runs and dynamically generates the output for the page.

We can understand the architecture of a Web Form most clearly by looking at the code-behind version of our "Echo" example. The visual content is specified by the **.aspx** file **HelloCodebehind.aspx**.

```
<!-- HelloCodebehind.aspx -->
<%@ Page Language="VB#" Src="HelloCodebehind.aspx.vb"
Inherits= MyWebPage %>
<HTML>
  <HEAD>
  </HEAD>
<BODY>
<FORM RUNAT="SERVER">YOUR NAME: 
<asp:textbox id=txtName Runat="server"></asp:textbox>
<p><asp:button id=cmdEcho onclick=cmdEcho_Click Text="Echo"
runat="server" tooltip="Click to echo your name">
</asp:button></p>
   <asp:label id=lblGreeting runat="server"></asp:label>
<P></P>
</FORM>
</BODY>
</HTML>
```

ch14.fm Page 595 Wednesday, May 22, 2002 1:38 PM



The user interface code is in the file **HelloCodebehind.aspx.vb**,

```
' HelloCodebehind.aspx.vb

Imports System
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls

Public Class MyWebPage
   Inherits System.Web.UI.Page

   Protected txtName As TextBox
   Protected cmdEcho As Button
   Protected lblGreeting As Label

   Protected Sub cmdEcho_Click(Source As Object, _
    e As EventArgs)
      lblGreeting.Text="Hello, " & txtName.Text
   End Sub
End Class
```

Page Class

The key namespace for Web Forms and Web services is **System.Web**. Support for Web Forms is in the namespace **System.Web.UI**. Support for server controls such as textboxes and buttons is in the namespace **System.Web.UI.WebControls**. The class that dynamically generates the output for an **.aspx** page is the **Page** class, in the **System.Web.UI** namespace, and classes derived from **Page**, as illustrated in the code-behind page in this last example.

INHERITING FROM PAGE CLASS

The elements in the **.aspx** file, the code in the code-behind file (or script block), and the base **Page** class work together to generate the page output. This cooperation is achieved by ASP.NET's dynamically creating a class for the **.aspx** file, which is derived from the code-behind class, which in turn is derived from **Page**. This relationship is created by the Inherits attribute in the .aspx file. Figure 14–6 illustrates the inheritance hierarchy. Here **MyWebPage** is a class we implement, derived from **Page**.

The most derived page class, shown as *My .aspx Page* in Figure 14–6, is dynamically created by the ASP.NET runtime. This class extends the page class, shown as *MyWebPage* in the figure, to incorporate the controls and HTML text on the Web Form. This class is compiled into an executable, which is run when the page is requested from a browser. The executable code creates the HTML that is sent to the browser.

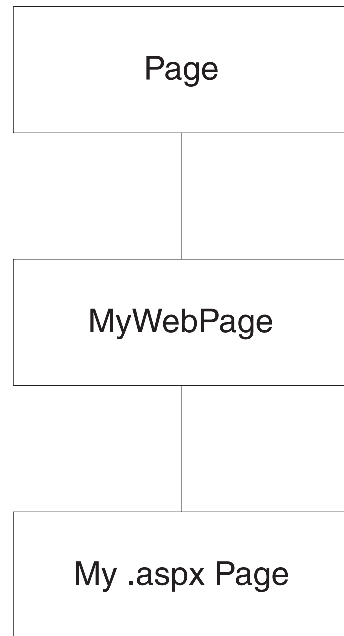


FIGURE 14–6 *Hierarchy of page classes.*

Web Forms Page Life Cycle

We can get a good high-level understanding of the Web Forms architecture by following the life cycle of our simple Echo application. We will use the code-behind version (the second example), **HelloCodebehind.aspx**.

1. User requests the **HelloCodebehind.aspx** Web page in the browser.
2. Web server compiles the page class from the **.aspx** file and its associated code-behind page. The Web server executes the code, creating HTML, which is sent to the browser. (In Internet Explorer you can see the HTML code from the menu View | Source.) Note that the server controls are replaced by straight HTML. The following code is what arrives at the browser, *not the original code on the server*.

```
<!-- HelloCodebehind.aspx -->

<HTML>
  <HEAD>
  </HEAD>
<BODY>
<form name="ctrl0" method="post"
action="HelloCodebehind.aspx" id="ctrl0">
```

```
<input type="hidden" name="__VIEWSTATE"
value="dDwxMzc4MDMwNTk1Ozs+" />
YOUR NAME:  <input name="txtName" type="text"
id="txtName" />
<p><input type="submit" name="cmdEcho" value="Echo"
id="cmdEcho" title="Click to echo your name" /></p>
   <span id="lblGreeting"></span>
<P></P>
</form>
</BODY>
</HTML>
```

3. The browser renders the HTML, displaying the simple form shown in Figure 14–7. To distinguish this example from the first one, we show “YOUR NAME” in all capitals. Since this is the first time the form is displayed, the text box is empty, and no greeting message is displayed.
4. The user types in a name (e.g., Mary Smith) and clicks the Echo button. The browser recognizes that a Submit button has been clicked. The method for the form is POST[1] and the action is HelloCodebehind.aspx. We thus have what is called a *postback* to the original **.aspx** file.
5. The server now performs processing for this page. An event was raised when the user clicked the Echo button, and an event handler in the **MyWebPage** class is invoked.

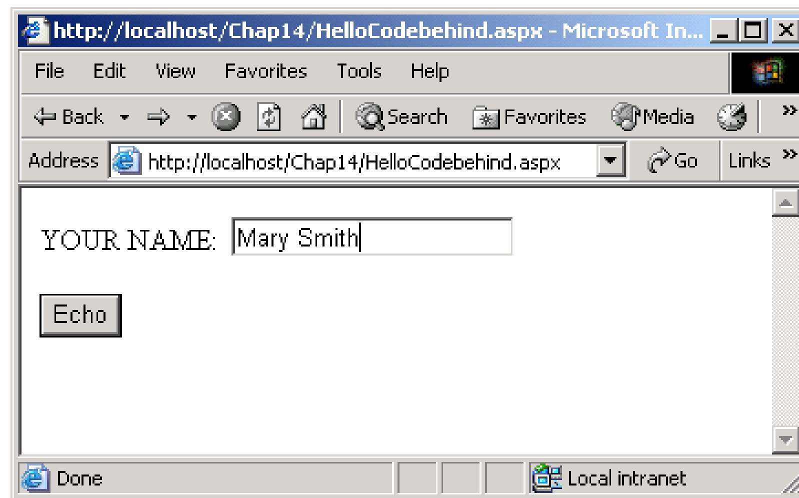


FIGURE 14–7 *The form for the Echo application is displayed for the first time.*

1. The HTTP POST method sends form results separately as part of the data body rather than by concatenating it onto the URL, as is done in the GET method.

```
Protected Sub cmdEcho_Click(Source As Object, _
 e As EventArgs)
   lblGreeting.Text="Hello, " & txtName.Text
End Sub
```

6. The **Text** property of the **TextBox** server control **txtName** is used to read the name submitted by the user. A greeting string is composed and assigned to the **Label** control **lblGreeting**, again using property notation.
7. The server again generates straight HTML for the server controls and sends the whole response to the browser. Here is the HTML.

```
...
<form name="ctrl0" method="post"
action="HelloCodebehind.aspx" id="ctrl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwxMzc4MDMwNTk1O3Q8O2w8aTwyPjs+O2w8dDw7bDxpPDU+Oz47
bDx0PHA8cDxsPFRleHQ7PjtsPEhlbGxvLCBNYXJ5IFNtaXRoOz4+Oz47Oz4
7Pj47Pj47Pg==" />
YOUR NAME:  <input name="txtName" type="text"
value="Mary Smith" id="txtName" />
<p><input type="submit" name="cmdEcho" value="Echo"
id="cmdEcho" title="Click to echo your name" /></p>
   <span id="lblGreeting">Hello, Mary Smith</span>
...
```

8. The browser renders the page, as shown in Figure 14–8. Now a greeting message is displayed.

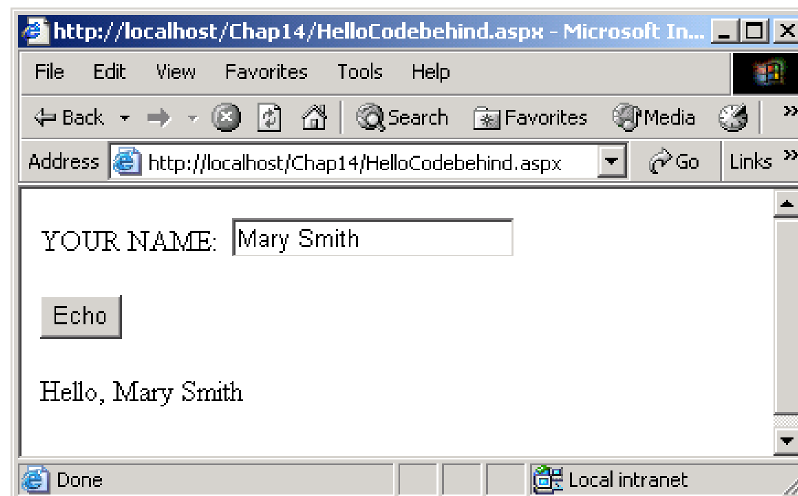


FIGURE 14–8 *After a round trip, a greeting message is displayed.*

ch14.fm Page 599 Wednesday, May 22, 2002 1:38 PM

View State

An important characteristic of Web Forms is that all information on forms is "remembered" by the Web server. Since HTTP is a stateless protocol, this preservation of state does not happen automatically but must be programmed. A nice feature of ASP.NET is that this state information, referred to as "view state," is preserved automatically by the framework, using a "hidden" control.

```
...
<input type="hidden" name="__VIEWSTATE"
value="dDwxMzc4MDMwNTk1O3Q8O2w8aTwyPjs+O2w8dDw7bDxpPDU+Oz47
bDx0PHA8cDxsPFRleHQ7PjtsPEhlbGxvLCBNYXJ5IFNtaXRoOz4+Oz47Oz4
7Pj47Pj47Pg==" />
...
```

Later in the chapter we will examine other facilities provided by ASP.NET for managing session state and application state.

Web Forms Event Model

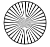
From the standpoint of the programmer, the event model for Web Forms is very similar to the event model for Windows Forms. Indeed, this similarity is what makes programming with Web Forms so easy. What is actually happening in the case of Web Forms, though, is rather different. The big difference is that events get raised on the client and processed on the server.[2]

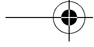
Our simple form with one textbox and one button is not rich enough to illustrate event processing very thoroughly. Let's imagine a more elaborate form with several textboxes, listboxes, checkboxes, buttons, and the like. Because round trips to the server are expensive, events do not automatically cause a postback to the server. Server controls have what is known as an intrinsic event set of events that automatically cause a postback to the server. The most common such intrinsic event is a button click. Other events, such as selecting an item in a list box, do not cause an immediate postback to the server. Instead, these events are cached, until a button click causes a post to the server. Then, on the server the various change events are processed, in no particular order, and the button-click event that caused the post is processed.

Page Processing

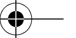
Processing a page is a cooperative endeavor between the Web server, the ASP.NET runtime, and your own code. The **Page** class provides a number of

2. Some controls, such as the Calendar control, raise some events on the server. Also, the Page itself raises events on the server.





ch14.fm Page 600 Wednesday, May 22, 2002 1:38 PM

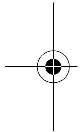
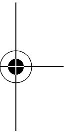


events, which you can handle to hook into page processing. The **Page** class also has properties and methods that you can use. We cover some of the major ones here. For a complete description, consult the .NET Framework documentation. The example programs in this chapter will illustrate features of the **Page** class.

PAGE EVENTS

A number of events are raised on the server as part of the normal processing of a page. These events are actually defined in the **Control** base class and so are available to server controls also. The most important ones are listed below.

- **Init** is the first step in the page's life cycle and occurs when the page is initialized. There is no view-state information for any of the controls at this point.
- **Load** occurs when the controls are loaded into the page. View-state information for the controls is now available.
- **PreRender** occurs just before the controls are rendered to the output stream. Normally this event is not handled by a page but is important for implementing your own server controls.
- **Unload** occurs when the controls are unloaded from the page. At this point it is too late to write your own data to the output stream.

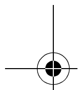
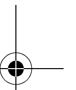
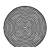



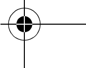
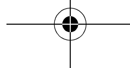
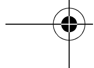
PAGE PROPERTIES

The **Page** class has a number of important properties. Some of the most useful are listed below.

- **EnableViewState** indicates whether the page maintains view state for itself and its controls. You can get or set this property. The default is **true**, view state is maintained.
- **ErrorPage** specifies the error page to which the browser should be redirected in case an unhandled exception occurs.
- **IsPostBack** indicates whether the page is being loaded in response to a postback from the client or is being loaded for the first time.
- **IsValid** indicates whether page validation succeeded.[3]
- **Request** gets the HTTP Request object, which allows you to access data from incoming HTTP requests.
- **Response** gets the HTTP Response object, which allows you to send response data to a browser.
- **Session** gets the current Session object, which is provided by ASP.NET for storing session state.

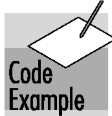
3. We discuss validation later in this chapter in the section "Server Controls."

- **Trace** gets a **TraceContext** object for the page, which you can use to write out trace information.

SAMPLE PROGRAM



We can illustrate some of these features of page processing with a simple extension to our Echo program. The page **HelloPage.aspx** (located in the top-level chapter directory) provides handlers for a number of page events, and we write simple text to the output stream, using the **Response** property. For each event we show the current text in the **txtName** and **lblGreeting** server controls. In the handler for **Load** we also show the current value of **IsPostBack**, which should be **false** the first time the page is accessed, and subsequently **true**.

```
<!-- HelloPage.aspx -->
<%@ Page Language="VB" Debug="true" %>
<HTML>
<HEAD>
 <SCRIPT RUNAT="SERVER">
Sub cmdEcho_Click(Source As Object, e As EventArgs)
   lblGreeting.Text="Hello, " & txtName.Text
End Sub

Sub Page_Init(sender As Object, E As EventArgs)
   Response.Write("Page_Init<br>")
   Response.Write("txtName = " & txtName.Text & "<br>")
   Response.Write("lblGreeting = " & lblGreeting.Text _
      & "<br>")
End Sub

Sub Page_Load(sender As Object, E As EventArgs)
   Response.Write("Page_Load<br>")
   Response.Write("IsPostBack = " & IsPostBack & "<br>")
   Response.Write("txtName = " & txtName.Text & "<br>")
   Response.Write("lblGreeting = " & lblGreeting.Text _
      & "<br>")
End Sub

Sub Page_PreRender(sender As Object, E As EventArgs)
   Response.Write("Page_PreRender<br>")
   Response.Write("txtName = " & txtName.Text & "<br>")
   Response.Write("lblGreeting = " & lblGreeting.Text _
      & "<br>")
End Sub

</SCRIPT>
</HEAD>
<BODY>
<FORM RUNAT="SERVER">Your name: 
<asp:textbox id=txtName Runat="server"></asp:textbox>
```

ch14.fm Page 602 Wednesday, May 22, 2002 1:38 PM


```
<p><asp:button id=cmdEcho onclick=cmdEcho_Click Text="Echo"
runat="server" tooltip="Click to echo your name">
</asp:button></p>
<asp:label id=lblGreeting runat="server"></asp:label>
<P></P>
</FORM>
</BODY>
</HTML>
```

When we display the page the first time the output reflects the fact that both the text box and the label are empty, since we have entered no information. **IsPostBack** is **false**.

Now enter a name and click the Echo button. We obtain the following output from our handlers for the page events:

```
Page_Init
txtName =
lblGreeting =
Page_Load
IsPostBack = True
txtName = Robert
lblGreeting =
Page_PreRender
txtName = Robert
lblGreeting = Hello, Robert
```

In **Page_Init** there is no information for either control, since view state is not available at page initialization. In **Page_Load** the text box has data, but the label does not, since the click-event handler has not yet been invoked. **IsPostBack** is now **true**. In **Page_PreRender** both controls now have data.

Click Echo a second time. Again, the controls have no data in **Page_Init**. This time, however, in **Page_Load** the view state provides data for both controls. Figure 14–9 shows the browser output after Echo has been clicked a second time.

Page Directive

An **.aspx** file may contain a *page directive* defining various attributes that can control how ASP.NET processes the page. A page directive contains one or more attribute/value pairs of the form

```
attribute="value"
```

within the page directive syntax

```
<@ Page ... @>
```

Our example program **HelloCodebehind.aspx** illustrates an **.aspx** page that does not have any code within it. The code-behind file **HelloCodebehind.aspx.vb** that has the code is specified using the **Src** attribute.

ch14.fm Page 603 Wednesday, May 22, 2002 1:38 PM

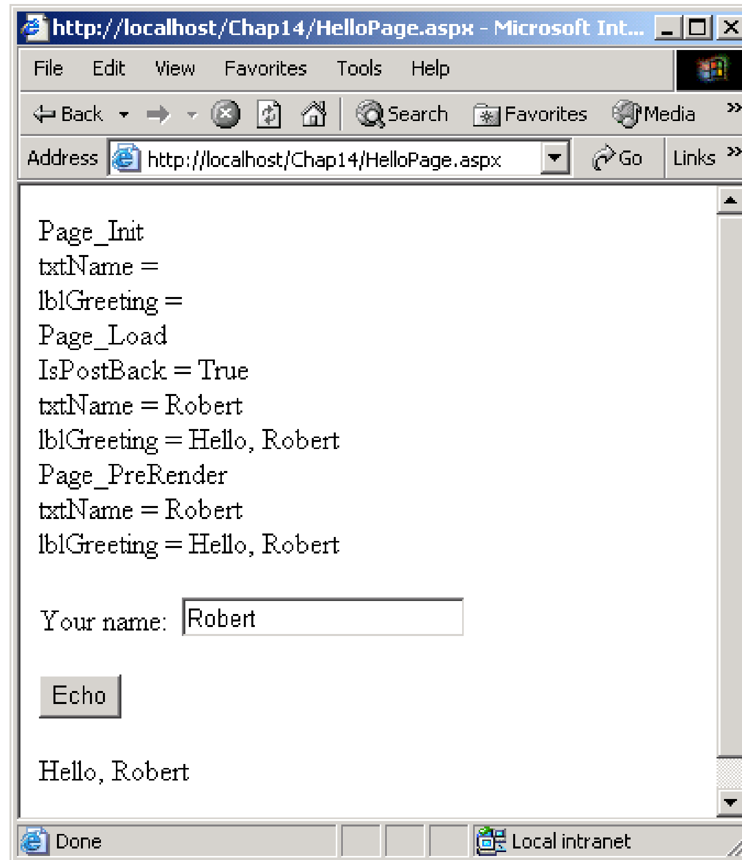


FIGURE 14–9 *Browser output after Echo has been clicked a second time.*

```
<!-- HelloCodebehind.aspx -->
<%@ Page Language="VB" Src="HelloCodebehind.aspx.vb"
Inherits=MyWebPage %>
...
```

Src

The **Src** attribute identifies the code-behind file.

Language

The **Language** attribute specifies the language used for the page. The code in this language may be in either a code-behind file or a SCRIPT block within the same file. Values can be any .NET-supported language, including C# and VB.NET.

Inherits

The **Inherits** directive specifies the page class from which the **.aspx** page class will inherit.

Debug

The **Debug** attribute indicates whether the page should be compiled with debug information. If **true**, debug information is enabled, and the browser can provide detailed information about compile errors. The default is **false**.

ErrorPage

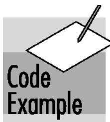
The **ErrorPage** attribute specifies a target URL to which the browser will be redirected in the event that an unhandled exception occurs on the page.

Trace

The **Trace** attribute indicates whether tracing is enabled. A value of **true** turns tracing on. The default is **false**.

Tracing

ASP.NET provides extensive tracing capabilities. Merely setting the **Trace** attribute for a page to **true** will cause trace output generated by ASP.NET to be sent to the browser. In addition, you can output your own trace information using the **Write** method of the **TraceContext** object, which is obtained from the **Trace** property of the **Page**.



The page **HelloTrace.aspx** illustrates using tracing in place of writing to the **Response** object.

```
<!-- HelloTrace.aspx -->
<%@ Page Language="C#" Debug="true" Trace = "true" %>
<HTML>
<HEAD>
 <SCRIPT RUNAT="SERVER">
Sub cmdEcho_Click(Source As Object, e As EventArgs)
   lblGreeting.Text="Hello, " & txtName.Text
End Sub

Sub Page_Init(sender As Object, E As EventArgs)
   Trace.Write("Page_Init<br>")
   Trace.Write("txtName = " & txtName.Text & "<br>")
   Trace.Write("lblGreeting = " & lblGreeting.Text _
      & "<br>")
End Sub
...
```

ch14.fm Page 605 Wednesday, May 22, 2002 1:38 PM

Figure 14–10 shows the browser output after the initial request for the page. Notice that the trace output is shown *after* the form, along with trace information that is generated by ASP.NET itself.

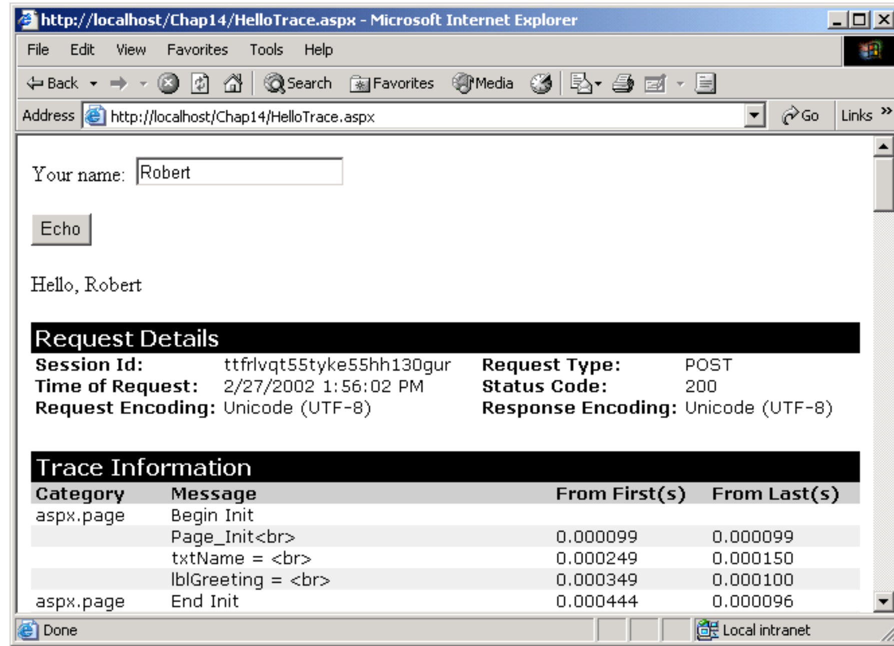


FIGURE 14–10 *Browser output showing trace information.*

Request/Response Programming

The server control architecture is built on top of a more fundamental processing architecture, which may be called *request/response*. Understanding request/response is important to solidify our overall grasp of ASP.NET. Also, in certain programming situations request/response is the natural approach.

HttpRequest Class

The **System.Web** namespace contains a useful class **HttpRequest** that can be used to read the various HTTP values sent by a client during a Web request. These HTTP values would be used by a classical CGI program in acting upon a Web request, and they are the foundation upon which higher level processing is built. Table 14–1 shows some of the public instance properties of

ch14.fm Page 606 Wednesday, May 22, 2002 1:38 PM

HttpRequest. If you are familiar with HTTP, the meaning of these various properties should be largely self-explanatory. Refer to the .NET Framework documentation of the **HttpRequest** class for full details about these and other properties.

TABLE 14–1 *Public Instance Properties of HttpRequest*

Property	Meaning
AcceptTypes	String array of client-supported MIME accept types
Browser	Information about client's browser capabilities
ContentLength	Length in bytes of content sent by the client
Cookies	Collection of cookies sent by the client
Form	Collection of form variables
Headers	Collection of HTTP headers
HttpMethod	HTTP transfer method used by client (e.g., GET or POST)
Params	Combined collection of QueryString, Form, ServerVariables, and Cookies items
Path	Virtual request of the current path
QueryString	Collection of HTTP query string variables
ServerVariables	Collection of Web server variables

The **Request** property of the **Page** class returns a **HttpRequest** object. You may then extract whatever information you need, using the properties of **HttpRequest**. For example, the following code determines the length in bytes of content sent by the client and writes that information to the **Response** object.

```
Dim length As Integer = Request.ContentLength
Response.Write("ContentLength = " & length & "<br>")
```

COLLECTIONS

A number of useful collections are exposed as properties of **HttpRequest**. The collections are of type **NamedValueCollection** (in **System.Collections.Specialized** namespace). You can access a value from a string key. For example, the following code extracts values for the QUERY_STRING and HTTP_USER_AGENT server variables using the **ServerVariables** collection.

```
Dim strQuery As String = _
   Request.ServerVariables("QUERY_STRING")
Dim strAgent as String = _
   Request.ServerVariables("HTTP_USER_AGENT")
```

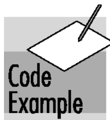
Server variables such as these are at the heart of classical Common Gateway Interface (CGI) Web server programming. The Web server passes information to a CGI script or program by using environment variables. ASP.NET makes this low-level information available to you, in case you need it.

A common task is to extract information from controls on forms. In HTML, controls are identified by a **name** attribute, which can be used by the server to determine the corresponding value. The way in which form data is passed to the server depends on whether the form uses the HTTP GET method or the POST method.

With GET, the form data is encoded as part of the query string. The **QueryString** collection can then be used to retrieve the values. With POST, the form data is passed as content after the HTTP header. The **Forms** collection can then be used to extract the control values. You could use the value of the REQUEST_METHOD server variable (GET or POST) to determine which collection to use (the **QueryString** collection in the case of GET and the **Forms** collection in case of POST).

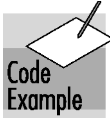
With ASP.NET you don't have to worry about which HTTP method was used in the request. ASP.NET provides a **Params** collection, which is a combination (union in the mathematical sense) of the **ServerVariables**, **QueryString**, **Forms**, and **Cookies** collections.

EXAMPLE PROGRAM



We illustrate all these ideas with a simple page **Squares.aspx** that displays a column of squares.

```
<!-- Squares.aspx -->
<%@ Page Language="VB" Trace="true"%>
<script runat="server">
Sub Page_Init(sender As Object, e As EventArgs)
   Dim strQuery As String = _
      Request.ServerVariables("QUERY_STRING")
   Response.Write("QUERY_STRING = " & strQuery & "<br>")
   Dim strAgent as String = _
      Request.ServerVariables("HTTP_USER_AGENT")
   Response.Write("HTTP_USER_AGENT = " & strAgent & "<br>")
   Dim length As Integer = Request.ContentLength
   Response.Write("ContentLength = " & length & "<br>")
   Dim strCount As String = Request.Params("txtCount")
   Dim count As Integer = Convert.ToInt32(strCount)
   Dim i As Integer
   For i = 1 To count
      Response.Write(i*i)
      Response.Write("<br>")
   Next
End Sub
</script>
```



How many squares to display is determined by a number submitted on a form. The page **GetSquares.aspx** submits the request using GET, and **PostSquares.aspx** submits the request using POST. These two pages have the same user interface, illustrated in Figure 14–11.

Here is the HTML for **GetSquares.aspx**. Notice that we are using straight HTML. Except for the Page directive, which turns tracing on, no features of ASP.NET are used.

```
<!-- GetSquares.aspx  -->
<%@ Page Trace = "true" %>
<html>
<head>
</head>
<body>
<P>This program will print a column of squares</P>
<form method="get" action = Squares.aspx>
How many:
<INPUT type=text size=2 value=5 name=txtCount>
<P></P>
<INPUT type=submit value=Squares name=cmdSquares>
</form>
</body>
</html>
```

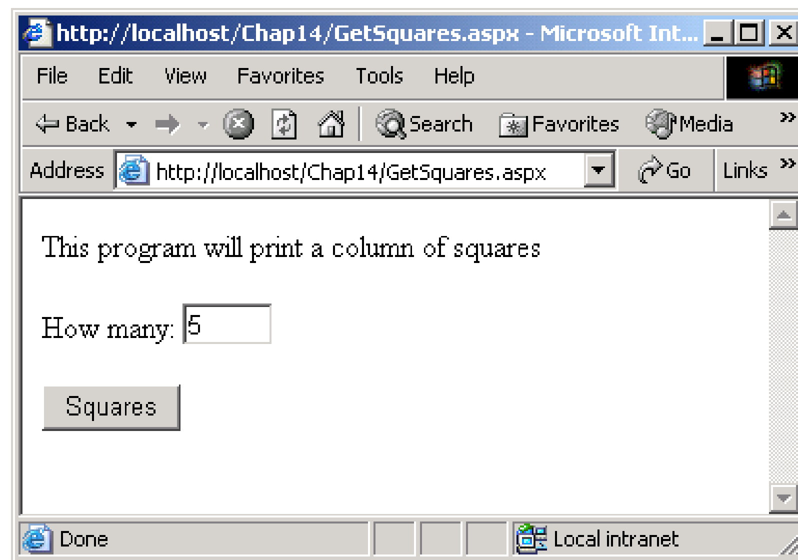


FIGURE 14–11 *Form for requesting a column of squares.*

The **form** tag has attributes specifying the method (GET or POST) and the action (target page). The controls have a **name** attribute, which will be used by server code to retrieve the value.

Run **GetSquares.aspx** and click Squares. You will see some HTTP information displayed, followed by the column of squares. Tracing is turned on, so details about the request are displayed by ASP.NET. Figure 14–12 illustrates the output from this GET request.

You can see that form data is encoded in the query string, and the content length is 0. If you scroll down on the trace output, you will see much information. For example, the **QueryString** collection is shown.

Now run **PostSquares.aspx** and click Squares. Again you will then see some HTTP information displayed, followed by the column of squares. Tracing is turned on, so details about the request are displayed by ASP.NET. Figure 14–13 illustrates the output from this POST request.

You can see that now the query string is empty, and the content length is 29. The form data is passed as part of the content, following the HTTP header information. If you scroll down on the trace output, you will see that now there is a **Form** collection, which is used by ASP.NET to provide access to the form data in the case of a POST method.

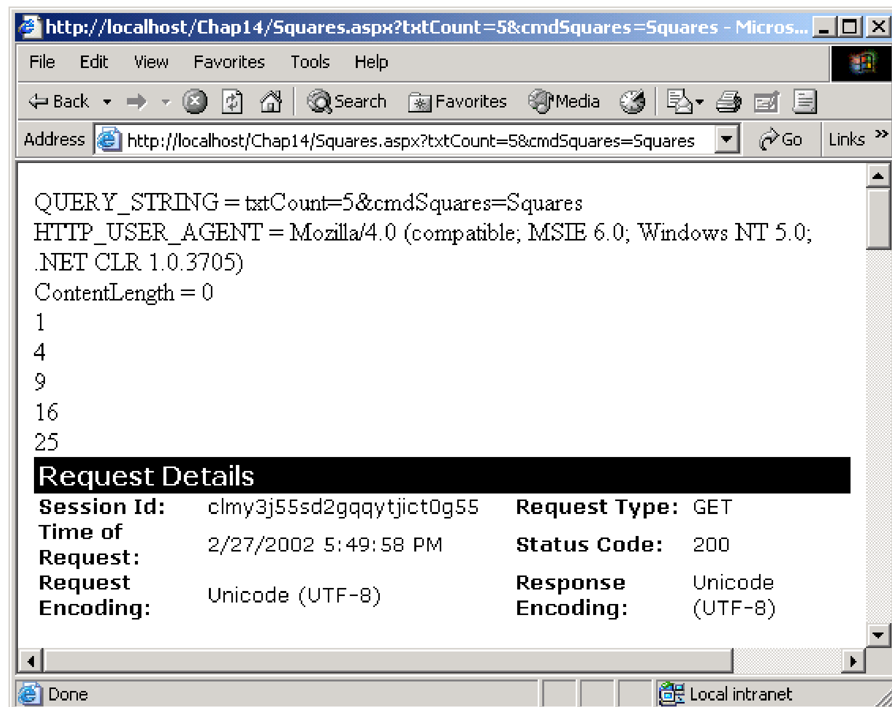


FIGURE 14–12 *Output from a GET request.*

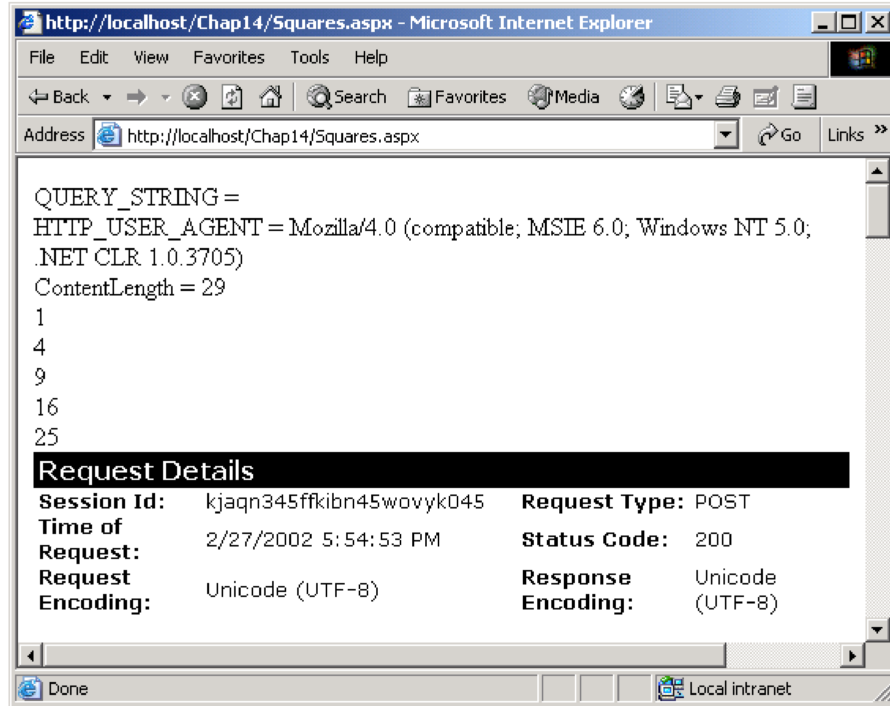


FIGURE 14–13 *Output from a POST request.*

By comparing the output of these two examples, you can clearly see the difference between GET and POST, and you can also see the data structures used by ASP.NET to make it easy for you to extract data from HTTP requests.

HttpResponse Class

The **HttpResponse** class encapsulates HTTP response information that is built as part of an ASP.NET operation. The Framework uses this class when it is creating a response that includes writing server controls back to the client. Your own server code may also use the **Write** method of the **Response** object to write data to the output stream that will be sent to the client. We have already seen many illustrations of **Response.Write**.

REDIRECT

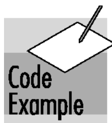
The **HttpResponse** class has a useful method, **Redirect**, that enables server code to redirect an HTTP request to a different URL. A simple redirection without passing any data is trivial—you need only call the **Redirect** method and pass the URL. An example of such usage would be a reorganization of a

ch14.fm Page 611 Wednesday, May 22, 2002 1:38 PM

Web site, where a certain page is no longer valid and the content has been moved to a new location. You can keep the old page live by simply redirecting traffic to the new location.

It should be noted that redirection always involves an HTTP GET request, like following a simple link to a URL. (POST arises as an option when submitting form data, where the action can be specified as GET or POST.)

A more interesting case involves passing data to the new page. One way to pass data is to encode it in the query string. You must preserve standard HTTP conventions for the encoding of the query string. The class **HttpUtility** provides a method **UrlEncode**, which will properly encode an individual item of a query string. You must yourself provide code to separate the URL from the query string with a "?" and to separate items of the query string with "&".



The folder **Hotel** provides an example of a simple Web application that illustrates this method of passing data in redirection. The file **default.aspx** provides a form for collecting information to be used in making a hotel reservation. The reservation itself is made on the page **Reservation1.aspx**. You may access the starting **default.aspx** page through the URL

```
http://localhost/Chap14/Hotel/
```

As usual, we provide a link to this page in our home page of example programs. Figure 14–14 illustrates the starting page of our simple hotel reservation example.

FIGURE 14–14 *Starting page for making a hotel reservation.*

ch14.fm Page 612 Wednesday, May 22, 2002 1:38 PM

Here is the script code that is executed when the Make Reservation button is clicked.

```
Sub cmdMakeReservation_Click(sender As Object, _
 e As EventArgs)
   Dim query As String = "City=" & _
      HttpUtility.UrlEncode(txtCity.Text)
   query += "&Hotel=" & _
      HttpUtility.UrlEncode(txtHotel.Text)
   query += "&Date=" & _
      HttpUtility.UrlEncode(txtDate.Text)
   query += "&NumberDays=" & _
      HttpUtility.UrlEncode(txtNumberDays.Text)
   Response.Redirect("Reservation1.aspx?" + query)
End Sub
```

We build a query string, which gets appended to the **Reservation1.aspx** URL, separated by a "?". Note the ampersand that is used as a separator of items in the query string. We use the **HttpUtility.UrlEncode** method to encode the individual items. Special encoding is required for the slashes in the date and for the space in the name San Jose. Clicking the button brings up the reservation page. You can see the query string in the address window of the browser. Figure 14–15 illustrates the output shown by the browser.

Our program does not actually make the reservation; it simply prints out the parameters passed to it.

```
<%@ Page language="VB" Debug="true" Trace="false" %>
<script runat="server">
   Sub Page_Load(sender As Object, e As EventArgs)
      Response.Write("Making reservation for ...")
```

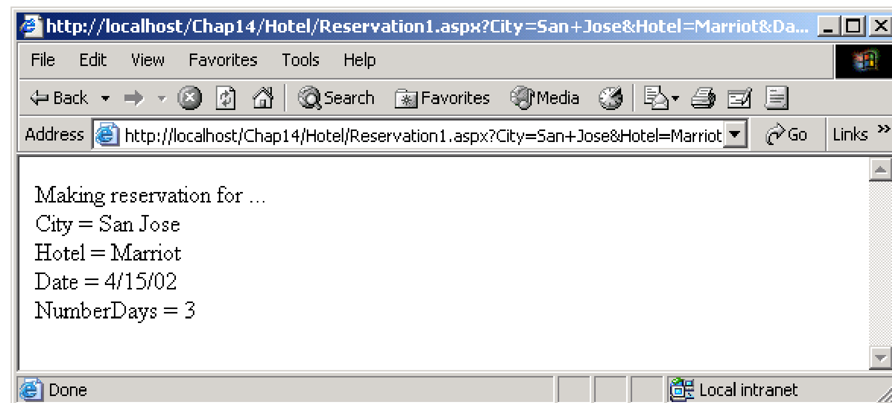


FIGURE 14–15 *Browser output from making a hotel reservation.*

```
      Response.Write("<br>")
      Dim city As String = Request.Params("City")
      Response.Write("City = " & city)
      Response.Write("<br>")
      Dim hotel As String = Request.Params("Hotel")
      Response.Write("Hotel = " & hotel)
      Response.Write("<br>")
      Dim strDate As String = Request.Params("Date")
      Response.Write("Date = " & strDate)
      Response.Write("<br>")
      Dim strDays As String = Request.Params("NumberDays")
      Response.Write("NumberDays = " & strDays)
      Response.Write("<br>")
   End Sub
</script>
<HTML>
<body>
</body>
</HTML>
```

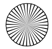
You can turn on tracing (in the file **Reservation1.aspx**), and the trace output should serve to reinforce the ideas we have been discussing about request/response Web programming. In particular, you should examine the **QueryString** collection, as illustrated in Figure 14–16.

Querystring Collection	
Name	**Value**
City	San Jose
Hotel	Marriot
Date	4/15/02
NumberDays	3

FIGURE 14–16 *The query string is used for passing parameters in redirection.*

Web Applications Using Visual Studio .NET

We have examined the fundamentals of ASP.NET and have created some simple Web pages. To carry the story further it will be very helpful to start using Visual Studio .NET. Everything we do could also be accomplished using only the .NET Framework SDK, but our work will be much easier using the facilities of Visual Studio. A special kind of project, an "ASP.NET Web Application," creates the boilerplate code. The Forms Designer makes it very easy to create Web forms by dragging controls from a palette. We can add event handlers for



controls in a manner very similar to the way event handlers are added in Windows Forms. In fact, the whole Web application development process takes on many of the rapid application development (RAD) characteristics typical of Visual Basic.

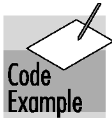
In this section we will introduce the Web application development features of Visual Studio by creating the first step of our Acme Travel Web site. We will elaborate on specific features of ASP.NET in later sections.

Form Designers for Windows and Web Applications

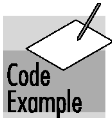
The basic look and feel of the Form Designers for Windows and Web applications is the same. You drag controls from a toolbox. You set properties in a Property window. You navigate between a code view and a designer view with toolbar buttons. In the following discussion we assume you have a basic familiarity with this visual paradigm. You may find it helpful to refer back to Chapter 7.

Hotel Information Web Page (Step 0)

We begin by creating a simple Web page that will display information about hotels. Dropdown listboxes are provided to show cities and hotels. Selecting a city from the first dropdown will cause the hotels in that city to be shown in the second dropdown. We obtain the hotel information from the **Hotel.dll** component, and we use data binding to populate the listboxes. As a source for the **Hotel.dll** and **Customer.dll** components used later, we provide a version of the GUI application from Chapter 7, **AcmeGui**. The **Hotel.dll** component we need in the following demonstration is in the folder **AcmeGui**.



If you would like to follow along hands-on with Visual Studio, do your work in the **Demos** folder for this chapter. The completed project is in **AcmeWeb\Step0**.



CONFIGURING WEB SERVER CONNECTION

Before getting started you may wish to check, and possibly change, your Visual Studio Web Server Connection setting. The two options are File share and FrontPage. If you are doing all your development on a local computer, you might find File share to be faster and more convenient. To access this setting, select the Visual Studio menu Tools | Options…. Choose Web Settings underneath Projects. You can then set the Preferred Access Method by using a radio button, as illustrated in Figure 14–17.

ch14.fm Page 615 Wednesday, May 22, 2002 1:38 PM

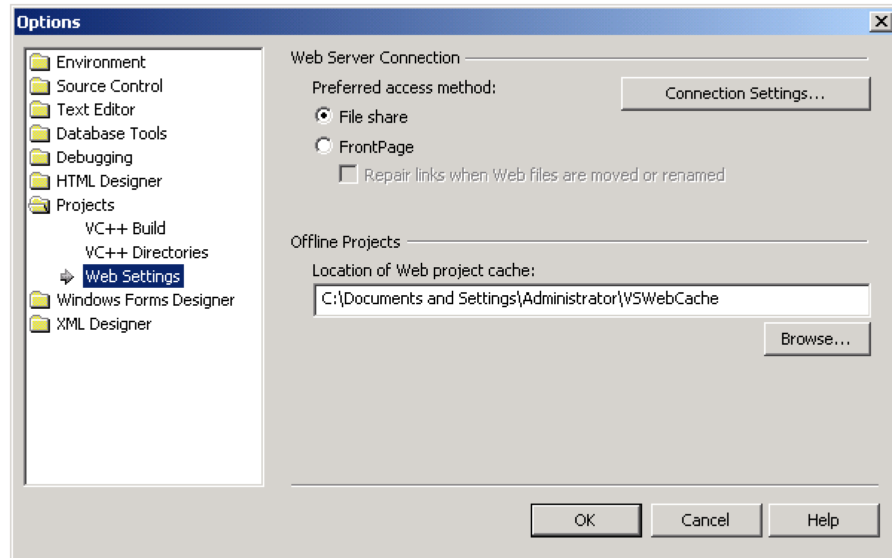


FIGURE 14–17 *Configuring Web server connection preferred access method.*

CREATING AN ASP.NET WEB APPLICATION

1. In Visual Studio select the menu File | New | Project....
2. In the New Project dialog box choose Visual Basic Projects as the Project Type and *ASP.NET Web Application* as the Template.
3. Enter **http://localhost/Chap14/Demos/AcmeWeb** as the location of your project, as illustrated in Figure 14–18. This setting assumes you have made **\OI\NetVb\Chap14** into a virtual directory with alias **Chap14**.
4. Click OK. The project files will then be created in **\OI\NetVb\Chap14\Demos\AcmeWeb**. The VS.NET solution **AcmeWeb.sln** will then be created under **MyDocuments\Visual Studio Projects\AcmeWeb**.

USING THE FORM DESIGNER

1. Bring up the Toolbox from the View menu, if not already showing. Make sure the Web Forms tab is selected.
2. Drag two Label controls and two DropDownList controls onto the form.
3. Change the Text property of the Labels to *City* and *Hotel*. Resize the DropDownList controls to look as shown in Figure 14–19.
4. Change the (ID) of the DropDownList controls to **listCities** and **listHotels**.

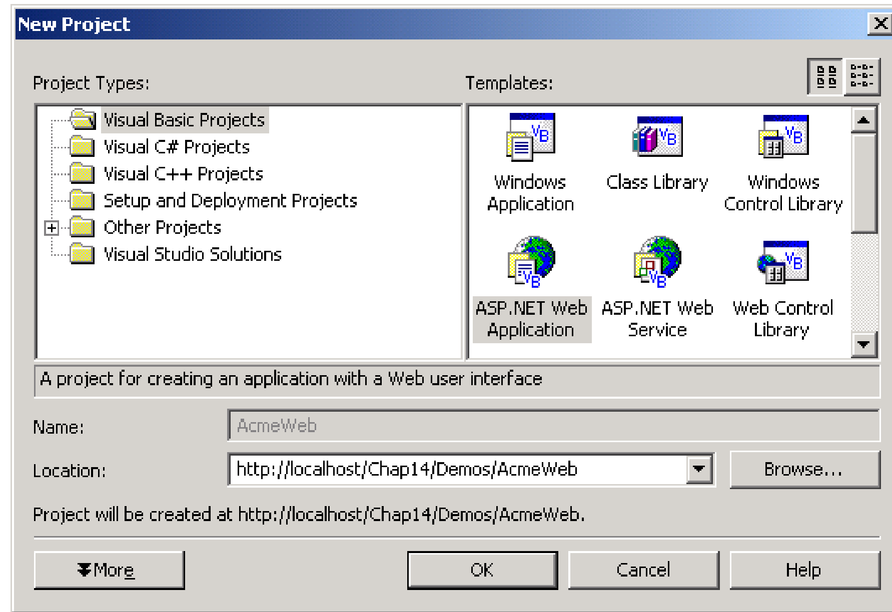


FIGURE 14–18 *Creating a Visual Studio ASP.NET Web Application project.*

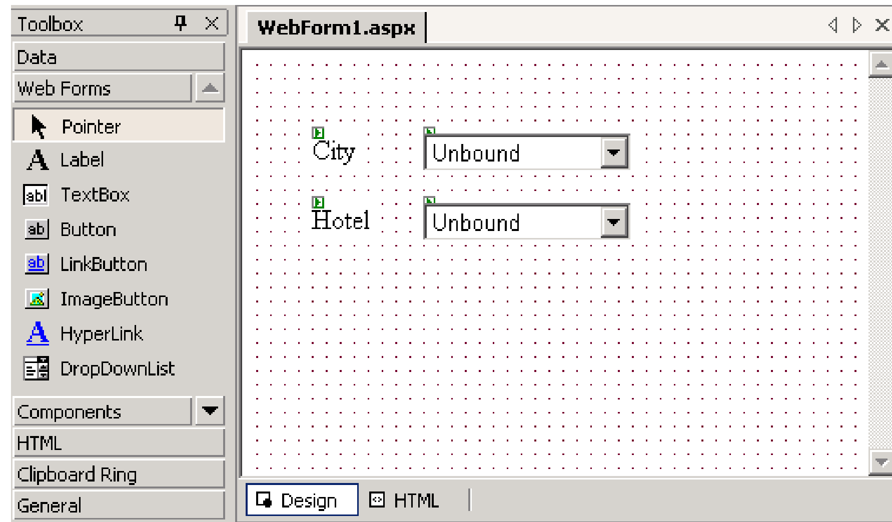
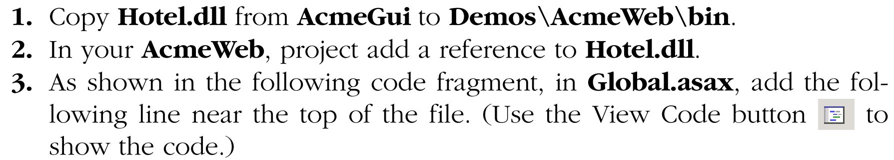


FIGURE 14–19 *Using the Form Designer to add controls to the form.*

INITIALIZING THE HOTELBROKER

1. Copy **Hotel.dll** from **AcmeGui** to **Demos\AcmeWeb\bin**.
2. In your **AcmeWeb**, project add a reference to **Hotel.dll**.
3. As shown in the following code fragment, in **Global.asax**, add the following line near the top of the file. (Use the View Code button  to show the code.)

```
Imports OI.NetVb.Acme
```

4. Add a public shared variable **broker** of type **HotelBroker**.
5. Add code to **Application_Start** to instantiate **HotelBroker**.

```
' Global.asax

Imports System.Web
Imports System.Web.SessionState
Imports OI.NetVb.Acme

Public Class Global
    Inherits System.Web.HttpApplication

#Region " Component Designer Generated Code "
   ...

   Public Shared broker As HotelBroker
   Sub Application_Start(ByVal sender As Object, _
    ByVal e As EventArgs)
      ' Fires when the application is started
      broker = New HotelBroker()
   End Sub
   ...
```

6. In **WebForm1.aspx.vb** add an **Imports OI.NetVb.Acme** statement, and declare a shared variable **broker** of type **HotelBroker**.

```
' WebForm1.aspx.vb

Imports OI.NetVb.Acme

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

   Private Shared broker As HotelBroker
   ...
```

DATA BINDING

Next we will populate the first DropDownList with the city data, which can be obtained by the **GetCities** method of **HotelBroker**. We make use of the *data*

ch14.fm Page 618 Wednesday, May 22, 2002 1:38 PM

binding capability of the DropDownList control. You might think data binding is only used with a database. However, in .NET data binding is much more general, and can be applied to other data sources besides databases. Binding a control to a database is very useful for two-tier, client/server applications. However, we are implementing a three-tier application, in which the presentation logic, whether implemented using Windows Forms or Web Forms, talks to a business logic component and not directly to the database. So we will bind the control to an **ArrayList**.

The .NET Framework provides a number of data binding options, which can facilitate binding to data obtained through a middle-tier component. A very simple option is binding to an **ArrayList**. This option works perfectly in our example, because we need to populate the DropDownList of cities with strings, and the **GetCities** method returns an array list of strings.

The bottom line is that all we need to do to populate the **listCities** DropDownList is to add the following code to the **Page_Load** method of the **WebForm1** class.

```
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
   If Not IsPostBack Then
      broker = Global.broker
      Dim cities As ArrayList = broker.GetCities()
      listCities.DataSource = cities
      DataBind()
   End If
End Sub
```

The call to **DataBind()** binds all the server controls on the form to their data source, which results in the controls being populated with data from the data source. The **DataBind** method can also be invoked on the server controls individually. **DataBind** is a method of the **Control** class, and is inherited by the **Page** class and by specific server control classes.

You can now build and run the project. Running a Web application under Visual Studio will bring up Internet Explorer to access the application over HTTP. Figure 14–20 shows the running application. When you drop down the list of cities, you will indeed see the cities returned by the **HotelBroker** component.

INITIALIZING THE HOTELS

We can populate the second DropDownList with hotel data using a similar procedure. It is a little bit more involved, because **GetHotels** returns an array list of **HotelListItem** structures rather than strings. We want to populate the **listHotels** DropDownList with the names of the hotels. The helper method **BindHotels** loops through the array list of hotels and creates an array list of

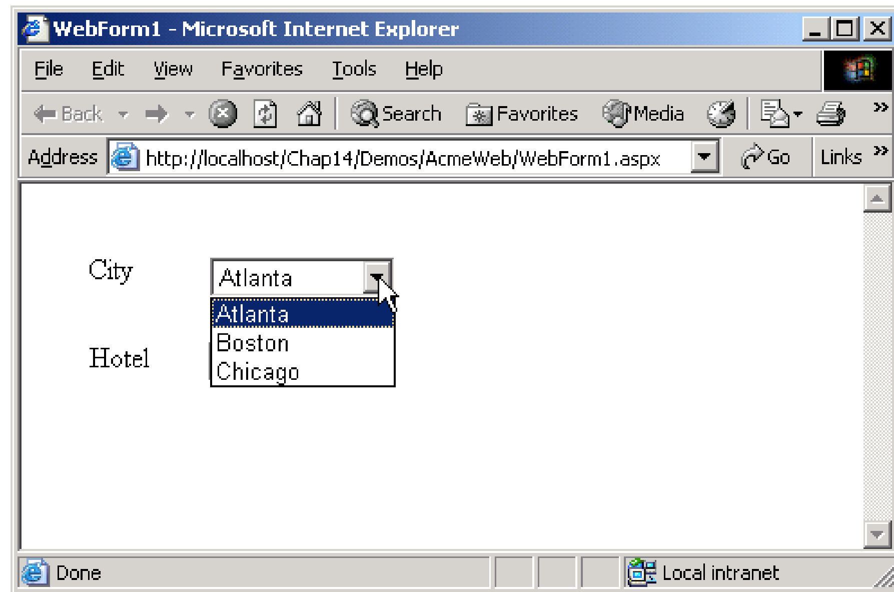


FIGURE 14–20 *Running the Web page to show information about cities.*

hotel names, which is bound to **listHotels**. Here is the complete code, which adds the logic for initializing the hotels for the first city (which has index 0).

```
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
   If Not IsPostBack Then
      broker = Global.broker
      Dim cities As ArrayList = broker.GetCities()
      listCities.DataSource = cities
      Dim hotels As ArrayList = _
         broker.GetHotels(CStr(cities(0)))
      BindHotels(hotels)
      DataBind()
   End If
End Sub

Private Sub BindHotels(ByVal hotels As ArrayList)
   Dim hotelNames As ArrayList = _
      New ArrayList(hotels.Count)
   Dim hotel As HotelListItem
   For Each hotel In hotels
      hotelNames.Add(hotel.HotelName.Trim())
   Next
   listHotels.DataSource = hotelNames
End Sub
```

ch14.fm Page 620 Wednesday, May 22, 2002 1:38 PM

SELECTING A CITY

Finally, we implement the feature that selecting a city causes the hotels for the selected city to be displayed. We can add an event handler for selecting a city by double-clicking on the **listCities** DropDownList control. This is a shortcut for adding a handler for the primary event for the control. Another method for adding an event handler for this control is to select **listCities** from the first dropdown in the **WebForm1.aspx.vb** code window. You can then choose an event from the second dropdown, as illustrated in Figure 14–21. The second method allows you to add a handler for *any* event of the control. Here is the code for the **SelectedIndexChanged** event:

```
Private Sub listCities_SelectedIndexChanged( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles listCities.SelectedIndexChanged
   Dim city As String = listCities.SelectedItem.Text
   Dim hotels As ArrayList = broker.GetHotels(city)
   BindHotels(hotels)
   DataBind()
End Sub
```

Build and run the project. Unfortunately, the event does not seem to be recognized by the server. What do you suppose the problem is?

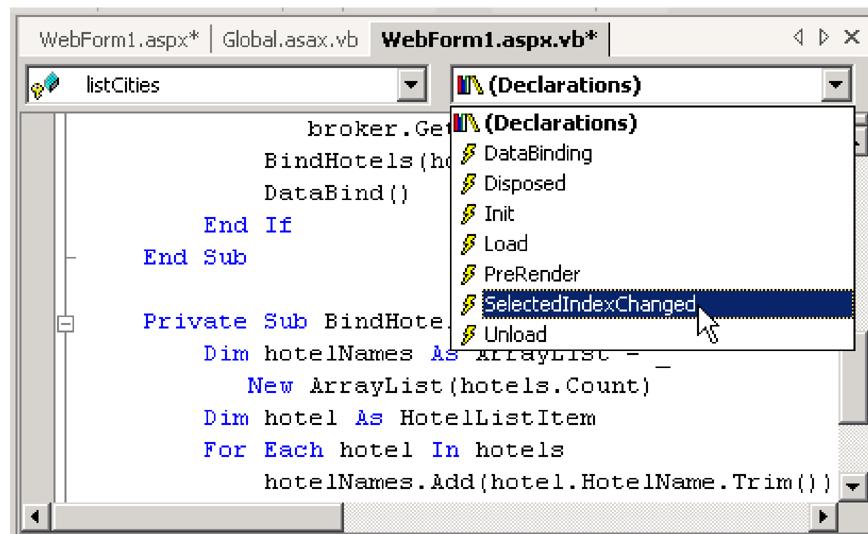


FIGURE 14–21 *Adding an event handler for a control.*

ch14.fm Page 621 Wednesday, May 22, 2002 1:38 PM

AUTOPOSTBACK

For an event to be recognized by the server, you must have a postback to the server. Such a postback happens automatically for a button click, but not for other events. Once this problem is recognized, the remedy is simple. In the Properties window for the cities DropDownList control, change the **AutoPostBack** property to **true**. Figure 14–22 illustrates setting the **AutoPostBack** property. The program should now work properly. The project is saved in the folder **AcmeWeb\Step0**.

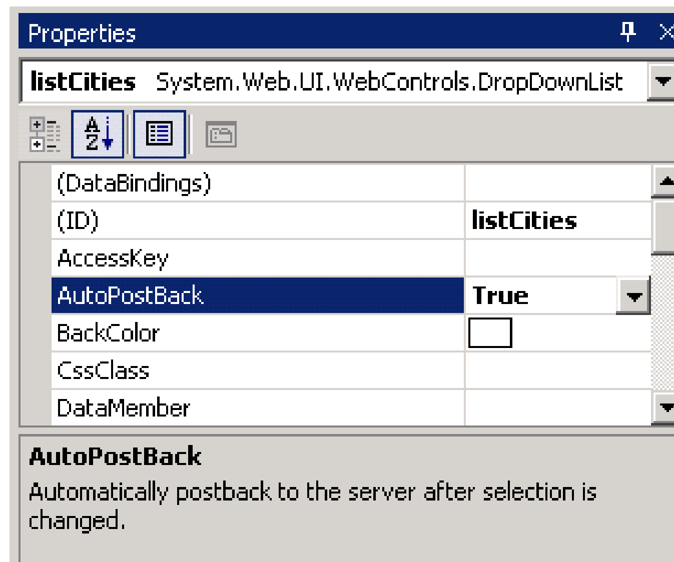


FIGURE 14–22 *Setting the AutoPostBack property of a DropDownList control.*

DEBUGGING

One advantage of using Visual Studio for developing your ASP.NET applications is the ease of debugging. You can set breakpoints, single-step, examine the values of variables, and so forth, in your code-behind files just as you would with any other Visual Studio program. All you have to do is build your project in Debug mode (the default) and start the program from within Visual Studio using Debug | Start (or F5 at the keyboard or the toolbar button ▶).

As an example, set a breakpoint on the first line of the **SelectedIndexChanged** event handler for **listCities**. Assuming you have set the **AutoPostBack** property to **True**, as we have discussed, you should hit the breakpoint.

Deploying a Web Application Created Using Visual Studio

Developing a Web application using Visual Studio is quite straightforward. You can do all your work within Visual Studio, including testing your application. When you start a Web application within Visual Studio, Internet Explorer will be brought up automatically. And it is easy to debug, as we have just seen.

Deploying a Web application created using Visual Studio is also easy, but you need to be aware of a few things.

1. The Project | Copy Project... menu can be used to deploy a Web project from Visual Studio.
2. Visual Studio precompiles Web pages, storing the executable in the **bin** folder.
3. The **Src** attribute in the Page directive is not used. Instead, the **Inherits** attribute is used to specify the Page class.
4. The directory containing the Web pages must be marked as a Web application. This marking is performed automatically by Visual Studio when you deploy the application. If you copy the files to another directory, possibly on another system, you must perform the marking as an application yourself, which you can do using Internet Services Manager. (We will discuss this procedure later in the chapter.)

USING PROJECT | COPY PROJECT...

To illustrate using Visual Studio to deploy a Web project, let's deploy the Acme Hotel Information page we have created. We will deploy it to a new directory **AcmeWeb** in the **Deploy** directory for Chapter 14.

1. Bring up the Copy Project dialog from the menu Project | Copy Project....
2. Enter the following information (see Figure 14–23).
   - **http://localhost/Chap14/Deploy/AcmeWeb** for Destination project folder
   - File share for Web access method
   - **\OI\NetVb\Chap14\Deploy\AcmeWeb** for Path
   - "Only files needed to run this application" for Copy
3. You can test the deployment by using Internet Explorer. Enter the following URL: **http://localhost/Chap14/Deploy/AcmeWeb/WebForm1.aspx**. You should then see the hotel information Web page displayed, and you should be able to select a city from the City dropdown and see the corresponding hotels displayed in the Hotel dropdown.

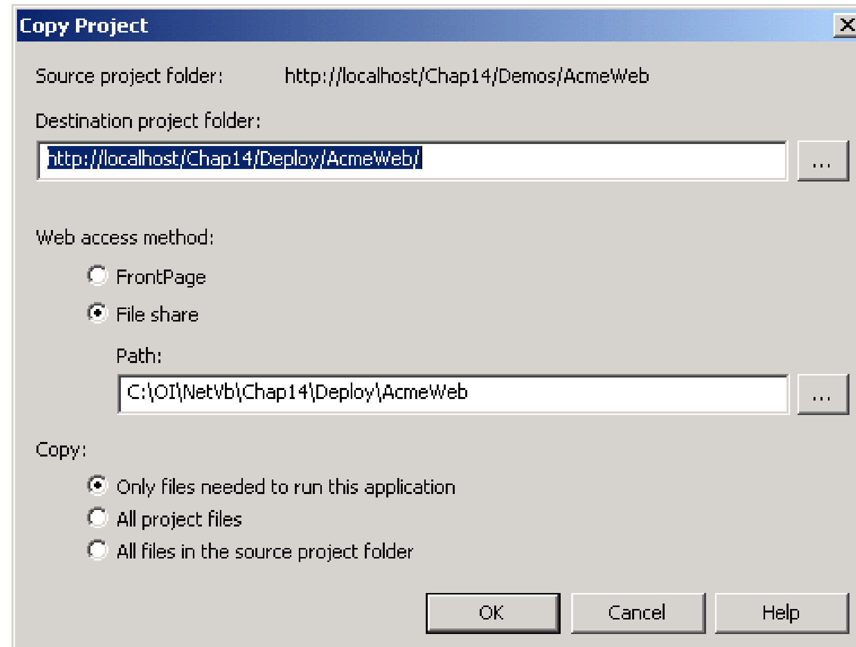


FIGURE 14–23 *Copying Web project files using Visual Studio.*

PRECOMPILED WEB PAGE

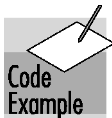
Examining the files in the folder **Deploy\AcmeWeb**, you will see no code-behind file **WebForm1.aspx.vb**. Instead, in the **bin** folder you will see the DLL **AcmeWeb.dll**.

INHERITS ATTRIBUTE IN PAGE DIRECTIVE

Examining the file **WebForm1.aspx**, we see there is no **Src** attribute. Instead, the **Inherits** attribute specifies the Page class **WebForm1**, which is implemented in the assembly **AcmeWeb.dll**.

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="WebForm1.aspx.vb"
Inherits="AcmeWeb.WebForm1"%>
```

CONFIGURING A VIRTUAL DIRECTORY AS AN APPLICATION



The identical files you copied to **Deploy\AcmeWeb** are also provided in the directory **AcmeRun**. Try the URL **http://localhost/Chap14/AcmeRun/WebForm1.aspx** in Internet Explorer. You will obtain a configuration error, as illustrated in Figure 14–24.

ch14.fm Page 624 Wednesday, May 22, 2002 1:38 PM

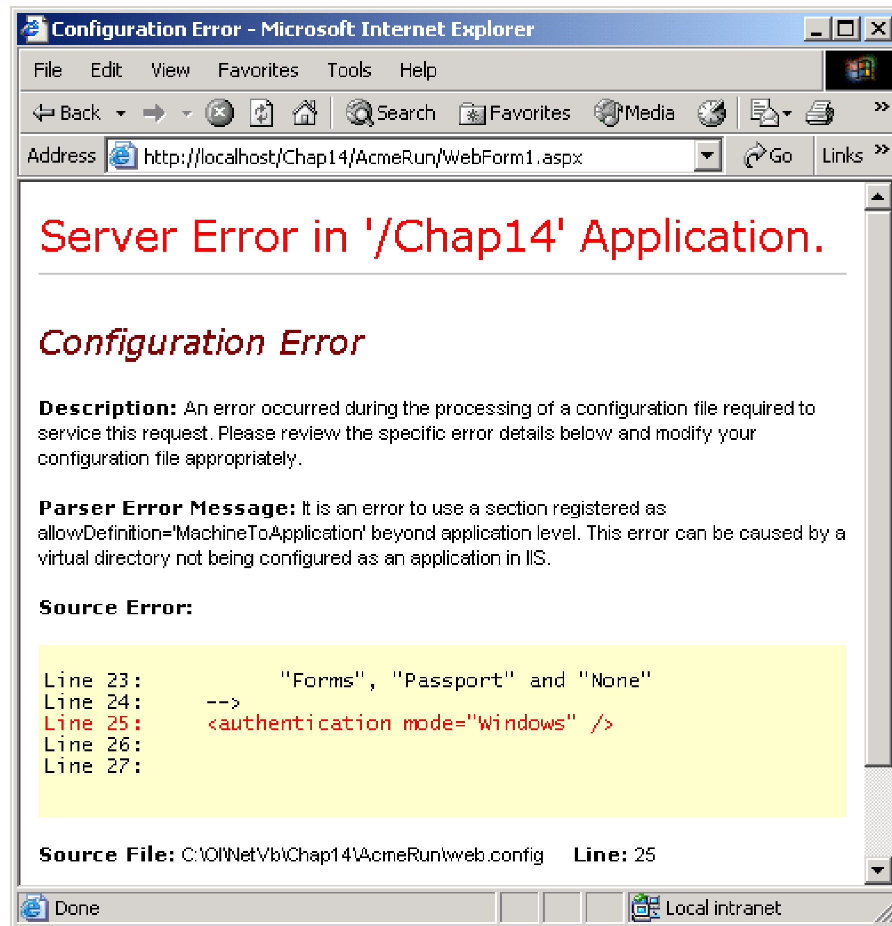


FIGURE 14–24 *Error message when virtual directory is not configured as an application.*

The key sentence in the error message is “This error can be caused by a virtual directory not being configured as an application in IIS.” The remedy is simple. Use Internet Services Manager to perform the following steps.

1. Find the folder **AcmeRun** in the virtual directory **Chap14**.
2. Right-click and choose properties. See Figure 14–25. Click Create.
3. Accept all the suggested settings and click OK.
4. Now again try **http://localhost/Chap14/AcmeRun/WebForm1.aspx** in Internet Explorer. You should be successful in bringing up the application.

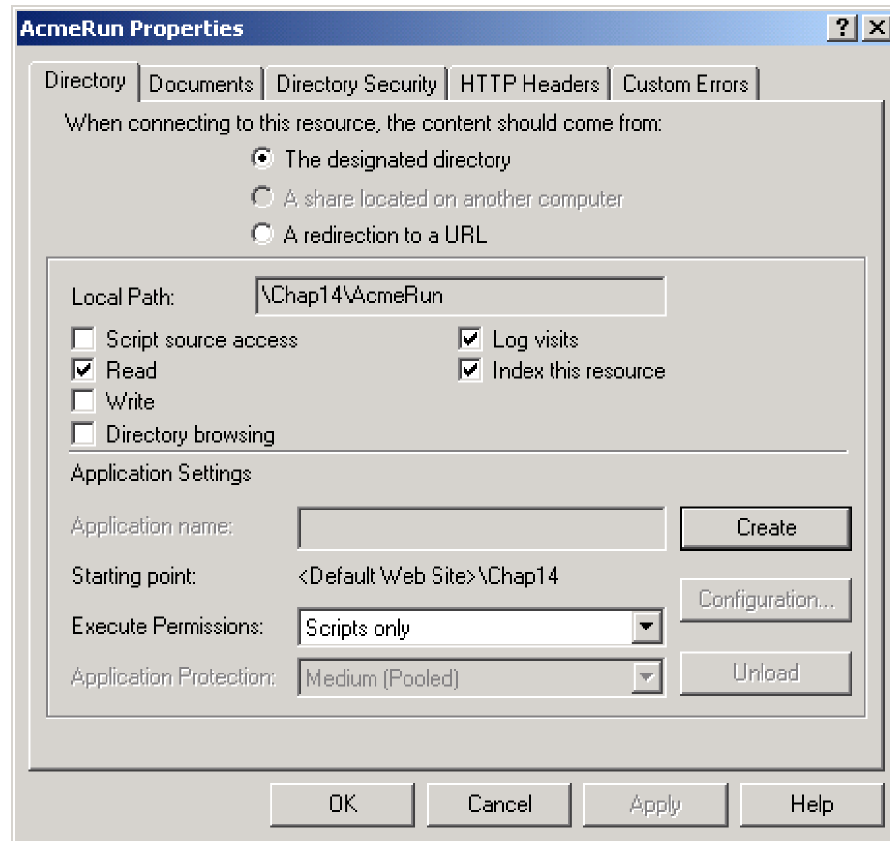


FIGURE 14–25 *Configuring a virtual directory as an application in IIS.*

MOVING A VISUAL STUDIO ASP.NET WEB APPLICATION PROJECT

Sometimes you will need to move an entire ASP.NET Web Application project so that you can continue development under Visual Studio. The simplest way to do this is to use the Visual Studio menu command Project | Copy Project. In the Copy Project dialog, select “All project files” for the Copy option. You will then enter the Destination project folder and the Path, as you did in deploying a Web application project. You will also need to edit the **.vbproj.webinfo** file to specify a correct URL path.

As an example, let’s copy the **AcmeWeb** project we have been working on in the **Demos** directory, saving our current work in a new folder, **AcmeWeb0** in the **Demos** directory.

1. Perform Copy | Copy Project, as described above. For Destination project folder enter **http://localhost/Chap14/Demos/AcmeWeb0**. Use

ch14.fm Page 626 Wednesday, May 22, 2002 1:38 PM

File share as the Web access method. Enter **C:\OI\NetVb\Chap14\Demos\AcmeWeb0** for the Path.

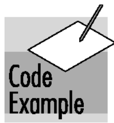
2. Edit the file **AcmeWeb.vbproj.webinfo** to rename **Web URLPath** to:

```
"http://localhost/Chap14/Demos/AcmeWeb0/AcmeWeb.vbproj"
```

3. Double-click on the file **AcmeWeb.vbproj**. This should bring up Visual Studio and create a new solution with a project **AcmeWeb**.
4. Build the solution. When presented with a Save As dialog, save the solution by the suggested name **AcmeWeb.sln**. You should get a clean build.
5. Try to run the project. You will be asked to set a start page. Set the start page as **WebForm1.aspx**.
6. Build and run. If you get a configuration error, use Internet Services Manager to configure the virtual directory as an application in IIS, as previously discussed. You should now be able to run the application at its new location.

You can view what we have done as establishing a snapshot of Step0. You can go back to new development in the main directory **Demos\AcmeWeb**, and if you want to compare with the original version, you have **Demos\AcmeWeb0** available.

Acme Travel Agency Case Study

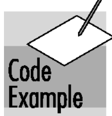


Throughout this book we have been using the Acme Travel Agency as a case study to illustrate many concepts of .NET. In this section we look at a Web site for the Acme Travel Agency. The code for the Web site is in the **AcmeWeb** directory in three progressive versions: Step0, Step1, and Step2. Step0 corresponds to our Visual Studio .NET demonstration from the preceding section. (A final step, discussed later in the chapter, is a database version of the case study. We deliberately avoid the database during most of the chapter, so as not to distract focus from the core Web programming topics.)

In this section we will give an overview of the case study, and in the next we will discuss some more details about Web applications, using the case study as an illustration.

Configuring the Case Study

Links are provided to the three steps of the case study on the ASP.NET example programs "home page" for this chapter, which you can access through the URL **http://localhost/Chap14/**. To be able to run the Web applications, you must use IIS to configure the directories **AcmeWeb/Step0**, **AcmeWeb/**



Step1, **AcmeWeb/Step2** as Web applications. Follow the instructions provided in the previous section. If you want to experiment with any of the versions in Visual Studio, you can double click on the **.vbproj** file to create a Visual Studio solution.

Acme Web Site Step 1

In Step 1 we provide a simple two-page Web site. In the first page you can make reservations, and in the second you can manage your reservations. We have hard-coded the customer as "Rocket Squirrel," who has a CustomerId of 1.

HotelReservations.aspx

The start page for the application is **HotelReservations.aspx**. Figure 14–26 shows this page in Internet Explorer, after a reservation has been booked at the hotel Dixie in Atlanta.

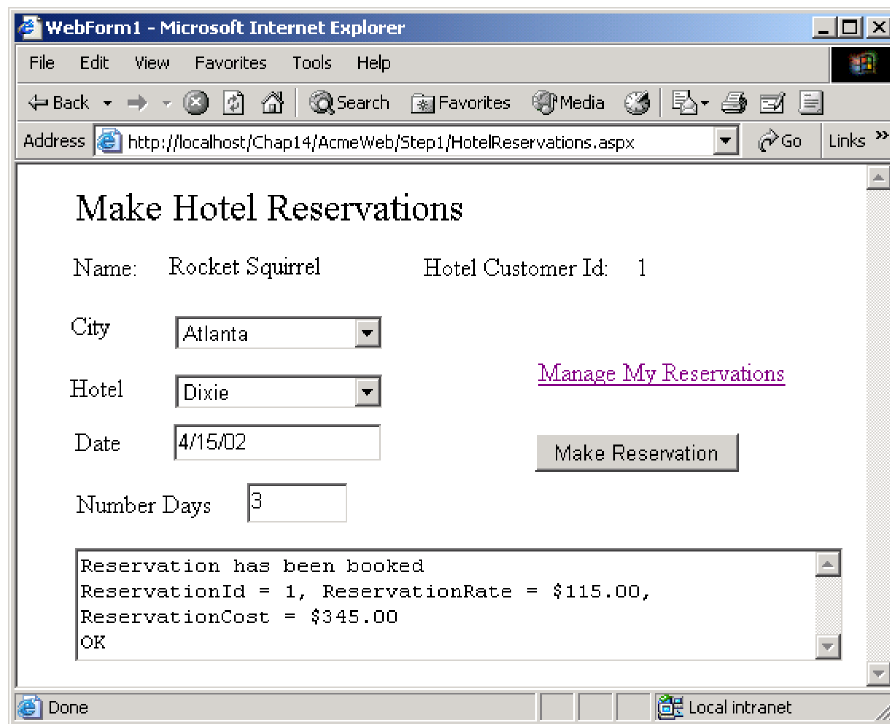


FIGURE 14–26 *Hotel reservations page of Acme Web site.*

ch14.fm Page 628 Wednesday, May 22, 2002 1:38 PM

The code for initializing the DropDownList controls is the same as for Step 0, as is the code for handling the **SelectedIndexChanged** event for the City dropdown. The key new code is making a reservation. This code should have no surprises for you. It makes use of the **HotelBroker** class, which we already have instantiated for displaying the hotels.

The design of the Web page enables a user to quickly make a number of reservations without leaving the page. We are relying on the postback mechanism of ASP.NET. When done making reservations, the user can follow the link "Manage My Reservations."

ManageReservations.aspx

The second page for the application is **ManageReservations.aspx**. Figure 14–27 shows this page in Internet Explorer, after reservations have been booked for Atlanta, Boston, and Chicago.

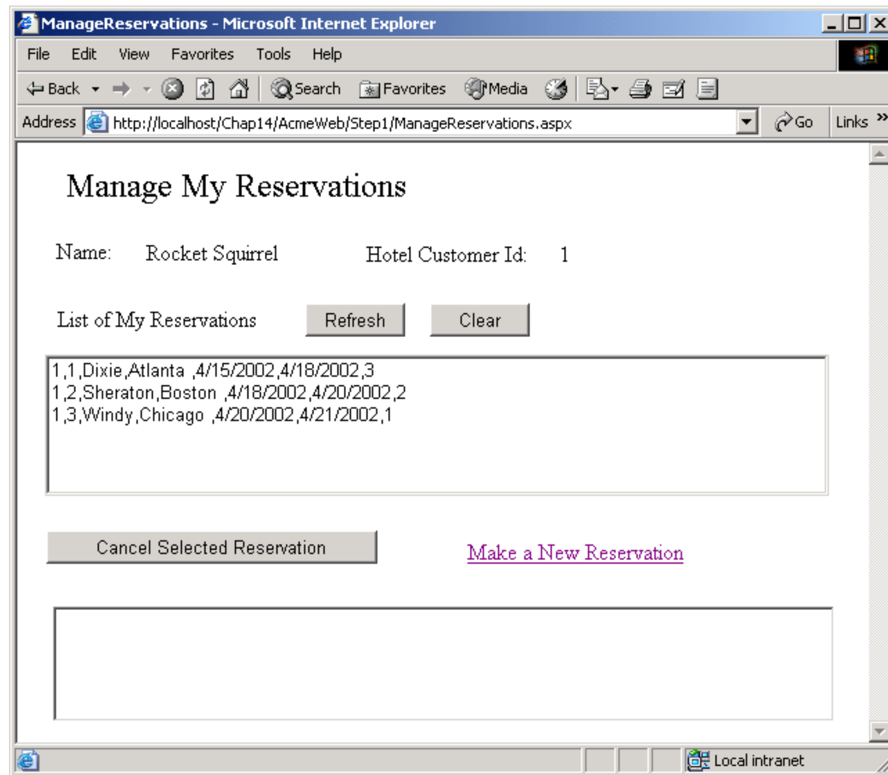


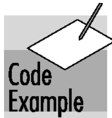
FIGURE 14–27 *Manage reservations page of Acme Web site.*

ch14.fm Page 629 Wednesday, May 22, 2002 1:38 PM

The user can cancel a reservation by selecting a reservation in the listbox and clicking the Cancel Selected Reservation button. A link is provided to the hotel reservations page. The code for this page is quite straightforward, making use of the capability to provide event handlers in a server-side control. Here is the code for a helper method to show the reservations in the listbox. This code is very similar to the Windows Forms code that we looked at in Chapter 7.

```
Private Sub ShowReservations()
   Dim id As Integer = _
      Convert.ToInt32(lblHotelCustomerId.Text)
   Dim array As ArrayList = _
      broker.FindReservationsForCustomer(id)
   If array Is Nothing Then
      Return
   End If
   ClearReservations()
   Dim item As ReservationListItem
   For Each item In array
      Dim rid As String = item.ReservationId.ToString()
      Dim hotel As String = item.HotelName
      Dim city As String = item.City
      Dim arrive As String = item.ArrivalDate.ToString("d")
      Dim depart As String = _
         item.DepartureDate.ToString("d")
      Dim number As String = item.NumberDays.ToString()
      Dim str As String = id & "," & rid & "," & hotel & _
         "," & city & " ," & arrive & "," & depart & "," _
         & number
      listReservations.Items.Add(str)
   Next
End Sub
```

Acme Web Site Step 2



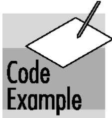
Step 2 is the full-blown implementation of our Web site case study. Acme customers do not interact with the Hotel Broker directly. Instead, they go through Acme's Web site. In order to use the Web site, a customer must register, providing a user ID, name, and email address. Subsequently, the user can log in by just providing the user ID.

ACMELIB COMPONENT

Internally, Acme maintains a database of user IDs and corresponding Hotel Customer IDs.[4] The interface **IAcmeUser** encapsulates this database main-

4. The Web site is Acme's, and Acme maintains user IDs for its own customers. Acme connects to various brokers (such as hotel and airline), and each broker will have its own customer ID.

ch14.fm Page 630 Wednesday, May 22, 2002 1:38 PM



tained by Acme. The class library project **AcmeLib** contains a collection-based implementation of such a database. The file **AcmeTravelDefs.cs** contains the definitions of interfaces and of a structure.

```
' AcmeTravelDefs.vb

Imports OI.NetVb.Acme

Public Interface IAcmeUser
   Function Login(ByVal uid As String) As Boolean
   Function Register(ByVal uid As String, _
      ByVal firstName As String, _
      ByVal lastName As String, _
      ByVal emailAddress As String) As Boolean
   Function Unregister(ByVal uid As String) As Boolean
   Function ChangeEmailAddress(ByVal uid As String, _
      ByVal emailAddress As String) As Boolean
   Function GetUserInfo(ByVal uid As String, _
      ByRef info As UserInfo) As Boolean
End Interface

Public Interface IAcmeAdmin
   Function GetUsers() As ArrayList
End Interface

Public Structure UserInfo
   Public HotelCustomerId As Integer
   Public FirstName As String
   Public LastName As String
   Public EmailAddress As String
End Structure
```

Login will return **True** if **uid** is found. **Register** will register a new user with the Hotel Broker. Methods are also provided to unregister and change email address. These methods will call the corresponding methods of the **ICustomer** interface. **GetUserInfo** will return a **UserInfo** struct as a **ByRef** parameter. This structure defines an Acme user. The method **GetUsers** of the **IAcmeAdmin** interface returns an array list of **UserInfo** structures.

The class **Acme** wraps access to the **Customers** class, whose methods get invoked indirectly through methods of **IAcmeUser**. The class **Acme** also contains a public member **broker** of type **HotelBroker**. Thus to gain complete access to the Hotel Broker system, a client program or Web page simply has to instantiate an instance of **Acme**. Here is the start of the definition of **Acme**.

```
Public Class Acme
   Implements IAcmeUser, IAcmeAdmin

   Public broker As HotelBroker
   Private custs As Customers
```

ch14.fm Page 631 Wednesday, May 22, 2002 1:38 PM


```
   Private users As ArrayList
   Private currUser As User

   Public Sub New()
      users = New ArrayList()
      broker = New HotelBroker()
      custs = New Customers()
      InitializeUsers()
   End Sub

   ' Initialize users with data from Customers list
   Private Sub InitializeUsers()
      Dim arr As ArrayList = custs.GetCustomer(-1)
      Dim cust As CustomerListItem
      For Each cust In arr
         Dim uid As String = cust.FirstName
         Dim custid As Integer = cust.CustomerId
         Dim us As User = New User(uid, custid)
         users.Add(us)
      Next
   End Sub
   ...
```

The class **Acme** also implements the interface **IAcmeAdmin**.

```
Public Interface IAcmeAdmin
   Function GetUsers() As ArrayList
End Interface
```

The method **GetUsers** returns an array list of **UserInfo**.

Login.aspx

To get a good feel for how this Web application works, it would be a good idea for you to register and make a few reservations. You could then try logging in as another user.[5] You can start up the application through the ASP.NET Example programs home page, link to Acme (Step 2), or else directly enter the URL:

```
http://localhost/Chap14/AcmeWeb/Step2/Main.aspx
```

The start page for the application is **Main.aspx**. If there is no currently logged-in user, the new user will be redirected to **Login.aspx**. We will examine the logic in **Main.aspx** shortly. For now, let's do the experiment of registering and logging in. Figure 14–28 shows the login page. In our implementation we offer "Rocket" as a possible user ID. Later you can quickly log in as "Rocket Squirrel" by simply clicking Login. But now click Register.

---

5. We are ignoring security considerations in this chapter. Security in ASP.NET will be discussed in Chapter 16.

ch14.fm Page 632 Wednesday, May 22, 2002 1:38 PM

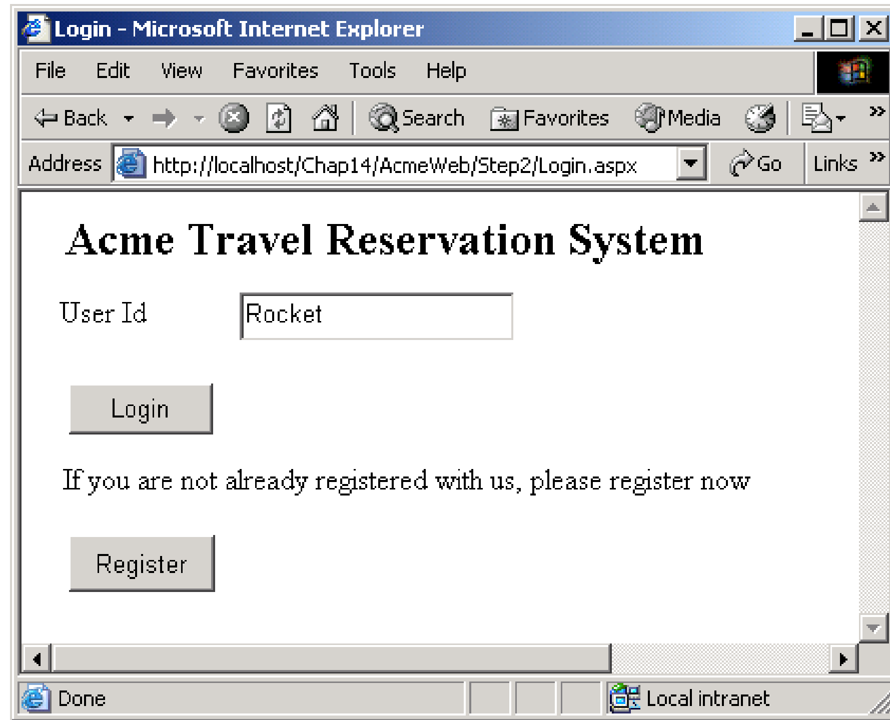


FIGURE 14–28 *Login page of Acme Web site.*

RegisterNewUser.aspx

The “Register New User” page allows the user to pick a User ID and enter some identifying information (first name, last name, and email address). Figure 14–29 shows this page after “John Smith” has entered information for himself. When done entering information, the user should click Register, which will directly bring up the Acme Travel Agency home page, bypassing a need for a separate login.

Main.aspx

The home page of the Acme Web Site is **Main.aspx**. Figure 14–30 shows this home page for the user John Smith, who has just registered. A link is provided to “Login” as a different user, if desired. There are links for “Make a Hotel Reservation” and “Manage Your Reservations.” These pages are the same as shown previously for Step 1.

ch14.fm Page 633 Wednesday, May 22, 2002 1:38 PM

FIGURE 14–29 *Register new user page of Acme Web site.*

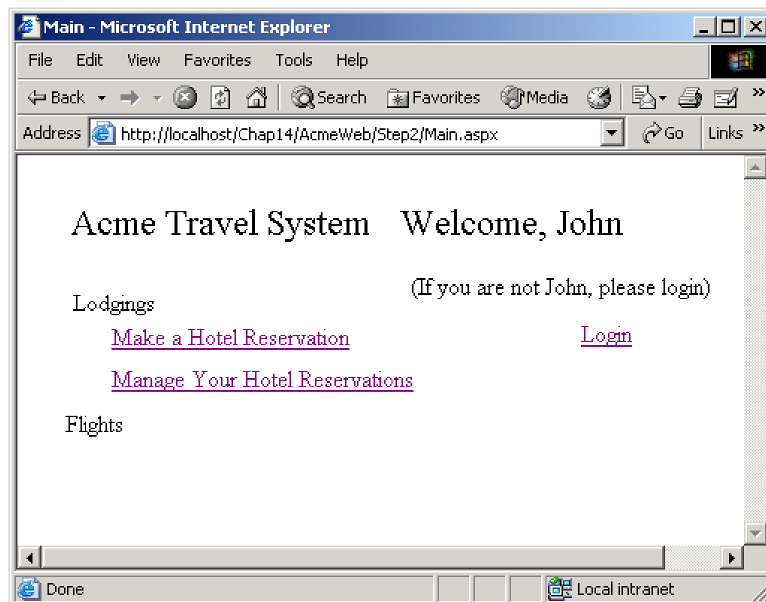


FIGURE 14–30 *Home page of the Acme Web site.*

ASP.NET Applications

An ASP.NET application consists of all the Web pages and code files that can be invoked from a virtual directory and its subdirectories on a Web server. Besides **.aspx** files and code-behind files such as those we have already examined, an application can also have a **global.asax** file and a configuration file **config.web**. In this section we examine the features of ASP.NET applications. We then investigate the mechanisms for working with application state and session state and for configuring Web applications. Our illustration will be our Acme Case Study (Step 2).

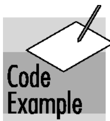
Sessions

To appreciate the Web application support provided by ASP.NET, we need to understand the concept of a Web *session*. HTTP is a stateless protocol. This means that there is no direct way for a Web browser to know whether a sequence of requests is from the same client or from different clients. A Web server such as IIS can provide a mechanism to classify requests coming from a single client into a logical session. ASP.NET makes it very easy to work with sessions.

Global.asax

An ASP.NET application can optionally contain a file **Global.asax**, which contains code for responding to application-level events raised by ASP.NET. This file resides in the root directory of the application. Visual Studio will automatically create a **Global.asax** file for you when you create an ASP.NET Web Application project. If you do not have a **Global.asax** file in your application, ASP.NET will assume you have not defined any handlers for application-level events.

Global.asax is compiled into a dynamically generated .NET Framework class derived from **HttpApplication**.



Here is the **Global.asax** file for our AcmeWeb case study, Step 2.

```
' Global.asax

Imports System.Web
Imports System.Web.SessionState
Imports OI.NetVb.Acme

Public Class Global
   Inherits System.Web.HttpApplication

#Region " Component Designer Generated Code "
...
```

ch14.fm Page 635 Wednesday, May 22, 2002 1:38 PM


```
   Public Shared acmedat As Acme

   Sub Application_Start(ByVal sender As Object, _
    ByVal e As EventArgs)
      ' Fires when the application is started
      acmedat = New Acme()
   End Sub

   Sub Session_Start(ByVal sender As Object, _
    ByVal e As EventArgs)
      ' Fires when the session is started
      Session("UserId") = ""
   End Sub

   Sub Application_BeginRequest(ByVal sender As Object, _
    ByVal e As EventArgs)
      ' Fires at the beginning of each request
   End Sub

   Sub Application_AuthenticateRequest( _
    ByVal sender As Object, ByVal e As EventArgs)
      ' Fires upon attempting to authenticate the use
   End Sub

   Sub Application_Error(ByVal sender As Object, _
    ByVal e As EventArgs)
      ' Fires when an error occurs
   End Sub

   Sub Session_End(ByVal sender As Object, _
    ByVal e As EventArgs)
      ' Fires when the session ends
   End Sub

   Sub Application_End(ByVal sender As Object, _
    ByVal e As EventArgs)
      ' Fires when the application ends
   End Sub

End Class
```

The most common application-level events are shown in this code. The typical life cycle of a Web application would consist of these events:

- **Application_Start** is raised only once during an application's lifetime, on the first instance of **HttpApplication**. An application starts the first time it is run by IIS for the first user. In your event handler you can initialize a state that is shared by the entire application.
- **Session_Start** is raised at the start of each session. Here you can initialize session variables.

- **Application_BeginRequest** is raised at the start of an individual request. Normally you can do your request processing in the **Page** class.
- **Application_EndRequest** is raised at the end of a request.
- **Session_End** is raised at the end of each session. Normally you do not need to do cleanup of data initialized in **Session_Start**, because garbage collection will take care of normal cleanup for you. However, if you have opened an expensive resource, such as a database connection, you may wish to call the **Dispose** method here.
- **Application_End** is raised at the very end of an application's lifetime, when the last instance of **HttpApplication** is torn down.

In addition to these events, there are other events concerned with security, such as **AuthenticateRequest** and **AuthorizeRequest**. We will discuss ASP.NET security in Chapter 16.

In the case study, we instantiate a single global **Acme** object instance **acmedat** in **Application_OnStart**. This single instance is stored as a shared data member of **Global**.

In the **Session_Start** event handler we initialize the session variable **UserId** to be a blank string. We discuss session variables later in this section.

State in ASP.NET Applications

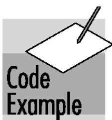
Preserving state across HTTP requests is a major problem in Web programming, and ASP.NET provides several facilities that are convenient to use. There are two main types of state to be preserved.

- **Application state** is global information that is shared across all users of a Web application.
- **Session state** is used to store data for a particular user across multiple requests to a Web application.

Shared Data Members

Shared data members of a class are shared across all instances of a class. Hence shared data members can be used to hold application state.

In our case study the class **Global** has a single shared member **acmedat** of the class **Acme**.



Thus the **broker** and **custs** objects within **Acme** will hold shared data that is the same for all users of the application. Each user will see the same list of hotels. You can view the source code for the **Acme** class in the **AcmeLib** project.

```
Public Class Acme
   Implements IAcmeUser, IAcmeAdmin

   Public broker As HotelBroker
   Private custs As Customers
   Private users As ArrayList
   Private currUser As User
```

If you like, you may perform a small experiment at this stage. The directory **HotelAdmin** contains a special version of the Acme Web site that makes available the hotel administration interface **IHotelAdmin** to the special user with user ID of “admin”. When this privileged user logins, a special home page will be displayed that provides a link to “Administer Hotels,” as illustrated in Figure 14–31.

Run this Web application, either from the “Hotel Admin” link on the example programs home page or else via the URL **http://localhost/Chap14/HotelAdmin/Main.aspx**. Log in as “admin” and follow the link to “Administer Hotels.” You will be brought to a page showing a list of all the hotels. Select the first hotel (Dixie) on the list and click the Delete Selected Hotel button and then the Refresh button. You will now see an updated list of hotels, as shown in Figure 14–32.

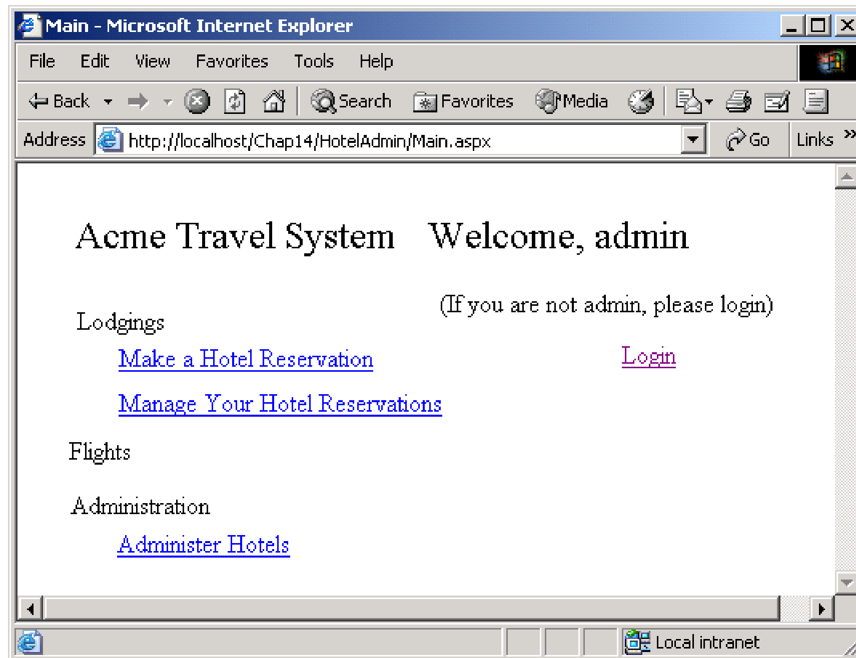


FIGURE 14–31 *Home page of the Acme Web site tailored for administrators.*

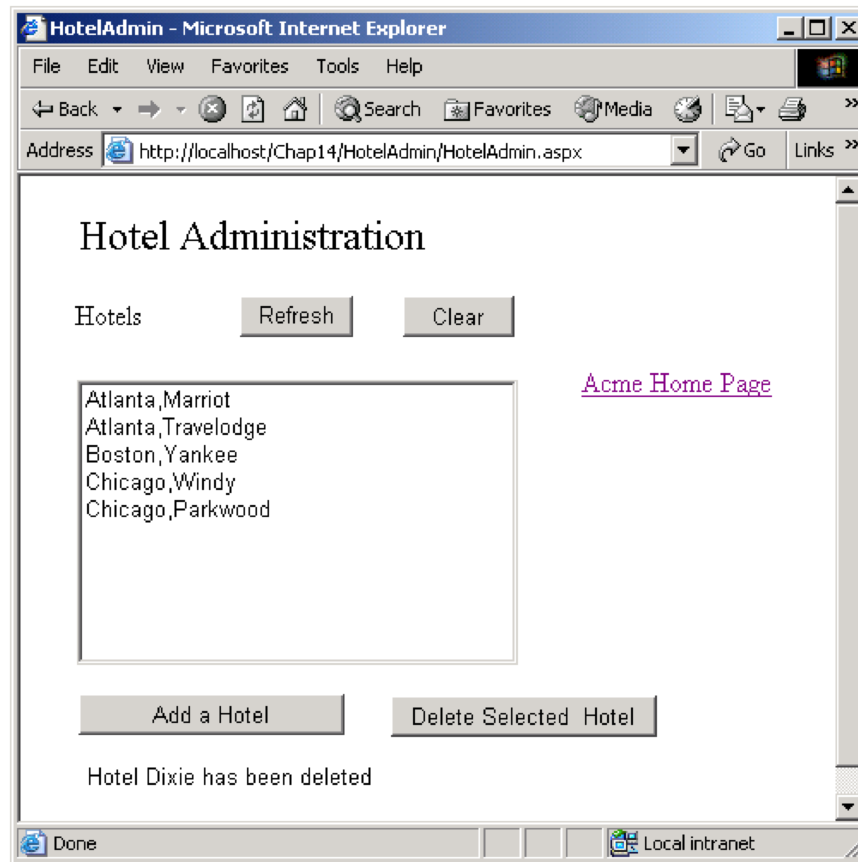


FIGURE 14–32 *Hotel administration page after deleting the hotel Dixie.*

If your Web server is on a network, you can now try running the same Web application from a different client. Use the URL

```
http://<server-name>/Chap14/HotelAdmin/Main.aspx
```

where <server-name> is the name of your server machine.[6] Again log in as "admin" and go to the "Hotel Admin" page. You should see the same list of hotels seen by the other client, with hotel Dixie not on the list.[7] You can also experiment with different browser sessions on the same machine, adding and deleting hotels, and using the Refresh button.

---

6. On a local machine you can use either the machine name or "localhost."
7. Remember that at this point we are not using a database. Thus our example illustrates application state preserved in memory.

Application Object

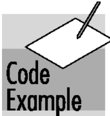
You can store global application information in the built-in **Application** object, an instance of the class **HttpApplicationState**. You can conveniently access this object through the **Application** property of the **Page** class. The **HttpApplicationState** class provides a key-value dictionary that you can use for storing both objects and scalar values.

For example, as an alternative to using the class **Global** with the shared member **acmedat** that we previously used, we could instead use the **Application** object. We make up a string name for the key—for example, "HotelState." In **Global.asax** we can then instantiate an **Acme** object and store it in the **Application** object using the following code.

```
Sub Application_Start(ByVal sender As Object, _
 ByVal e As EventArgs)
   Application("HotelState") = New Acme()
End Sub
```

You can then retrieve the **Acme** object associated with "HotelState" by using the index expression on the right-hand side and casting to **Acme**, as illustrated in the code,

```
Dim acmedat As Acme = _
   CType(Application("HotelState"), Acme)
Dim name As String = acmedat.CurrentUserInfo.FirstName
```



As a little exercise in employing this technique, you may wish to modify Step 2 of **AcmeWeb** to use the **Application** object in place of a shared data member. The solution to this exercise can be found in the directory **ApplicationObject**.[8]

Session Object

You can store session information for individual users in the built-in **Session** object, an instance of the class **HttpSessionState**. You can conveniently access this object through the **Session** property of the **Page** class. The **HttpSessionState** class provides a key-value dictionary that you can use for storing both objects and scalar values, in exactly the same manner employed by **HttpApplicationState**.

---

8. In our current example of a Web application that is precompiled by Visual Studio, it is quite feasible to use a static variable that can be shared across pages. But if your application is not precompiled, each page will be compiled individually at runtime, and sharing a static variable is no longer feasible. Hence you will have to use the **Application** object to share data.

ch14.fm Page 640 Wednesday, May 22, 2002 1:38 PM

Our **AcmeWeb** case study provides an example of the use of a session variable UserId for storing a string representing the user ID. The session variable is created and initialized in **Global.asax**.

```
Sub Session_Start(ByVal sender As Object, _
 ByVal e As EventArgs)
   ' Fires when the session is started
   Session("UserId") = ""
End Sub
```

We use this session variable in the **Page_Load** event of our home page **Main.aspx** to detect whether we have a returning user or a new user. A new user is redirected to the login page. (Note that “returning” means coming back to the home page during the same session.)

```
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
   Dim userid As String = CStr(Session("UserId"))
   If userid = "" Then
      Response.Redirect("Login.aspx")
   End If
   If Not IsPostBack Then
      Dim name As String = _
      Global.acmedat.CurrentUserInfo.FirstName
      lblUserName.Text = "Welcome, " & name
      lblLogin.Text = "(If you are not " & name & _
        ", please login)"
   End If
End Sub
```

There are some interesting issues in the implementation of session variables.

- Typically, cookies are used to identify which requests belong to a particular session. What if the browser does not support cookies, or the user has disabled cookies?
- There is overhead in maintaining session state for many users. Will session state “expire” after a certain time period?
- A common scenario in high-performance Web sites is to use a server farm. How can your application access its data if a second request for a page is serviced on a different machine from that on which the first request was serviced?

SESSION STATE AND COOKIES

Although by default ASP.NET uses cookies to identify which requests belong to a particular session, it is easy to configure ASP.NET to run cookieless. In this mode the Session ID, normally stored within a cookie, is instead embedded within the URL. We will discuss cookieless configuration in the next section.

ch14.fm Page 641 Wednesday, May 22, 2002 1:38 PM

SESSION STATE TIMEOUT

By default session state times out after 20 minutes. This means that if a given user is idle for that period of time, the session is torn down; a request from the client will now be treated as a request from a new user, and a new session will be created. Again, it is easy to configure the timeout period, as we will discuss in the section on Configuration.

SESSION STATE STORE

ASP.NET cleanly solves the Web farm problem, and many other issues, through a session state model that separates storage from the application's use of the stored information. Thus different storage scenarios can be implemented without affecting application code. The .NET state server does not maintain "live" objects across requests. Instead, at the end of each Web request, all objects in the Session collection are serialized to the session state store. When the same client returns to the page, the session objects are deserialized.

By default, the session state store is an in-memory cache. It can be configured to be memory on a specific machine, or to be stored in an SQL Server database. In these cases the data is not tied to a specific server, and so session data can be safely used with Web farms.

ASP.NET Configuration

In our discussion of session state we have seen a number of cases where it is desirable to be able to configure ASP.NET. There are two types of configurations:

- **Server configuration** specifies default settings that apply to all ASP.NET applications.
- **Application configuration** specifies settings specific to a particular ASP.NET application.

Configuration Files

Configuration is specified in files with an XML format, which are easy to read and to modify.

SERVER CONFIGURATION FILE

The configuration file is **machine.config**. This file is located within a version-specific folder under **\WINNT\Microsoft..NET\Framework**. Because there are separate files for each version of .NET, it is perfectly possible to run

different versions of ASP.NET side-by-side. Thus if you have working Web applications running under one version of .NET, you can continue to run them, while you develop new applications using a later version.

APPLICATION CONFIGURATION FILES

Optionally, you may provide a file **web.config** at the root of the virtual directory for a Web application. If the file is absent, the default configuration settings in **machine.config** will be used. If the file is present, any settings in **web.config** will override the default settings.

CONFIGURATION FILE FORMAT

Both **machine.config** and **web.config** files have the same XML-based format. There are sections that group related configuration items together, and individual items within the sections. As an easy way to get a feel both for the format of **web.config** and also for some of the important settings you may wish to adjust, just look at the **web.config** file that is created by Visual Studio when you create a new ASP.NET Web Application project.

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

  <system.web>

    <!--  DYNAMIC DEBUG COMPILATION
          Set compilation debug="true" to insert debugging
          symbols (.pdb information) into the compiled
          page. Because this creates a larger file that
          executes more slowly, you should set this value
          to true only when debugging and to false at all
          other times. For more information, refer to the
          documentation about debugging ASP.NET files.
          ...
    -->
    <compilation
         defaultLanguage="vb"
         debug="true"
    />

    <!--  CUSTOM ERROR MESSAGES
          Set customErrors mode="On" or "RemoteOnly" to
          enable custom error messages, "Off" to disable.
          Add <error> tags for each of the errors you want
          to handle.
    -->
    <customErrors
    mode="Off"
    />
```

ch14.fm Page 643 Wednesday, May 22, 2002 1:38 PM


```
    <!--  AUTHENTICATION
          This section sets the authentication policies of
          the application. Possible modes are "Windows",
          "Forms", "Passport" and "None"
    -->
    <authentication mode= "Windows" />

    ...

</system.web>
</configuration>
```

Application Tracing

Earlier in the chapter we examined page-level tracing, which can be enabled with the **Trace="true"** attribute in the Page directive. Page-level tracing is useful during development but is rather intrusive, because the page trace is sent back to the browser along with the regular response. Application tracing, which is specified in **web.config**, writes the trace information to a log file, which can be viewed via a special URL.

As a demonstration of the use of **web.config**, let's add application tracing to our original **Hello.aspx** application. The folder **HelloConfig** contains **Hello.aspx** and **web.config**. We have added a trace statement in **Hello.aspx**.

```
<!-- Hello.aspx -->
<%@ Page Language="VB" %>
<HTML>
<HEAD>
    <SCRIPT RUNAT="SERVER">
   Sub cmdEcho_Click(Source As Object, e As EventArgs)
      lblGreeting.Text="Hello, " & txtName.Text
      Trace.Write("cmdEcho_Click called")
   End Sub
    </SCRIPT>
</HEAD>
<BODY>
<FORM RUNAT="SERVER">Your name: 
<asp:textbox id=txtName Runat="server"></asp:textbox>
<p><asp:button id=cmdEcho onclick=cmdEcho_Click Text="Echo"
runat="server" tooltip="Click to echo your name">
</asp:button></p>
<asp:label id=lblGreeting runat="server"></asp:label>
<P></P>
</FORM>
</BODY>
</HTML>
```

We have provided a trace section in **web.config** to enable tracing.

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

```
    <system.web>
      <trace
          enabled="true"
      />
   </system.web>
</configuration>
```

You can run this application from Internet Explorer by simply providing the URL **http://localhost/Chap14/HelloConfig/Hello.aspx**.[9] Enter a name and click the Echo button. The application should run normally, without any trace information included in the normal page returned to the browser.

Now enter the following URL: **http://localhost/Chap14/HelloConfig/trace.axd** (specifying **trace.axd** in place of **hello.aspx**), and you will see top-level trace information, with a line for each trip to the server, as shown in Figure 14–33. If you click on the "View Details" link, you will see a detailed page trace, as we saw earlier in the chapter. The detailed trace corresponding to the POST will contain the trace output "cmdEcho_Click called" provided by our own code.

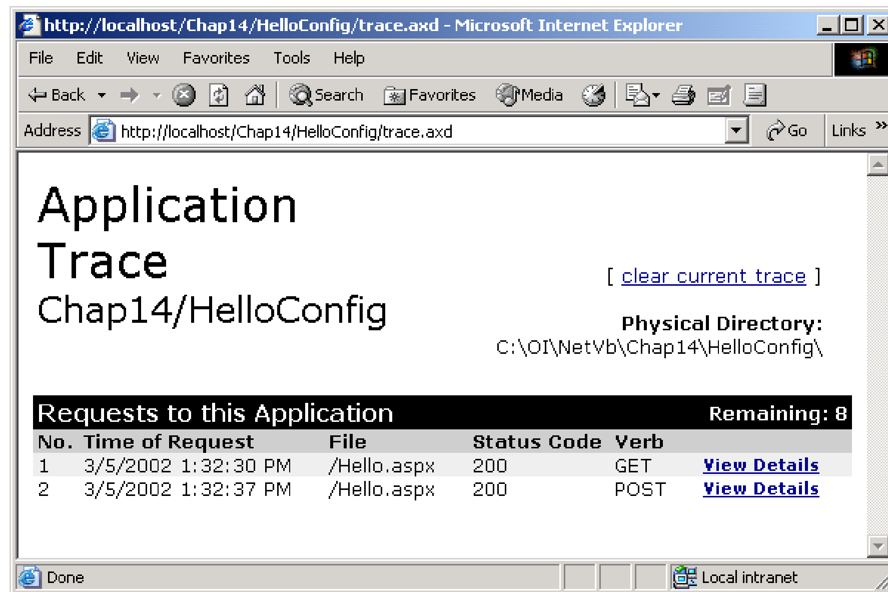


FIGURE 14–33 *Viewing the application trace log through the browser.*

9. If you get a configuration error, try configuring the directory in IIS as an application. See "Configuring a Virtual Directory as an Application" in the section "Deploying a Web Application Created Using Visual Studio."

Session Configuration

As another example of configuration, modify the **web.config** file for Step 2 of the case study to change the timeout value to be 1 minute.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
...
    <!--  SESSION STATE SETTINGS
          By default ASP.NET uses cookies to identify which
           requests belong to a particular session. If
           cookies are not available, a session can be
           tracked by adding a session identifier to the
           URL. To disable cookies, set sessionState
           cookieless="true".
    -->
    <sessionState
            mode="InProc"
            stateConnectionString="tcpip=127.0.0.1:42424"
            sqlConnectionString=
               "data source=127.0.0.1;user id=sa;password="
            cookieless="false"
            timeout="1"
    />
...
 </system.web>
</configuration>
```

Now run the application, log in, do some work, and return to the home page. You should be welcomed by your name without having to log in again. Now do some more work, wait more than a minute, and return to the home page. Now the session will have timed out, and you will be redirected to log in again.

Server Controls

An important innovation in ASP.NET is server controls. They provide an event model that is startlingly similar to Windows GUI programming, and they encapsulate browser dependencies. They integrate seamlessly into the Visual Studio development environment. The end result is an extremely powerful tool for Web development.

We have been using server controls from the very beginning of the chapter, where we presented our “Hello” program. In this section we will look at server controls more systematically, and we will see a number of examples of interesting controls.

ch14.fm Page 646 Wednesday, May 22, 2002 1:38 PM

Web Controls

The most important kind of control in ASP.NET is the *Web Forms server control* or just *Web control*. These are new controls provided by the .NET Framework, with special tags such as <asp:textbox>. These controls run at the server, and they generate HTML code that is sent back to the browser. They are easy to work with, because they behave consistently. For example, you can determine the value returned by a control by using simple property notation.

```
Dim name As String = txtName.Text
```

All of our previous examples of server controls in this chapter have been Web controls. In this section, we will look at several additional kinds of Web controls, including validation controls, list controls, and rich controls such as the Calendar control. But first we will look at HTML server controls.

HTML Server Controls

HTML server controls provide equivalent functionality to standard HTML controls, except that they run on the server, not on the client. In fact, the only way to distinguish an HTML server control from an ordinary HTML control on a Web page is the presence of the **runat="server"** attribute.

Here are two controls. Both are INPUT controls. The first is a server control. The second is of type password and is a regular HTML control.

```
<INPUT id=txtUserId
style="WIDTH: 135px; HEIGHT: 22px" type=text size=17
runat="server"></P>
<INPUT id=""
style="WIDTH: 138px; HEIGHT: 22px" type=password size=17
name=txtPassword>
```

Working with HTML server controls is much like working with the Web Forms server controls we've used already. In server-side code you access the control through a control variable that has the same name as the **id** attribute. However, we are dealing with HTML controls, so there are some differences. You access the string value of the control not through the **Text** property but through the **Value** property. Here is some code that uses the value entered by the user for the **txtUserId** control.

```
lblMessage.Text = "Welcome, " & txtUserId.Value
```

The advantage of HTML server controls for the experienced Web programmer is that they match ordinary HTML controls exactly, so that your knowledge of the details of HTML control properties and behavior carries over to the ASP.NET world. However, this similarity means they carry over all the quirks and inconsistencies of HTML. For example, rather than having two different controls for the somewhat different behaviors of a textbox and a

password control, HTML uses in both cases the INPUT control, distinguishing between the two by the **type=password** attribute. Web Forms controls, in contrast, are a fresh design and have an internal consistency. Also, as we shall soon see, there is a much greater variety to Web Forms controls.

HTML CONTROLS EXAMPLE

Let's look at an example of HTML controls. All of our server control examples in this section can be accessed from the page **ServerControls\WebForms1.aspx**. (As usual, you should use IIS to configure the folder **ServerControls** as an application.) The top-level page gives you a choice of three examples,

- HTML Controls
- Validation
- Calendar

Follow the link to HTML Controls, and you will come to a login page, as illustrated in Figure 14–34.

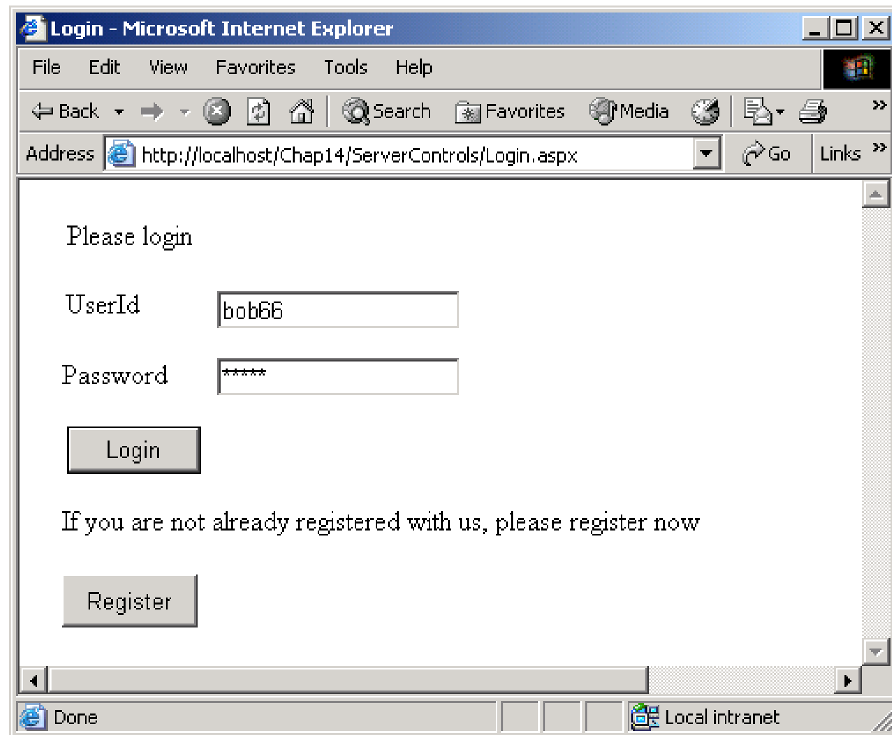


FIGURE 14–34 *A login page illustrating HTML server controls.*

ch14.fm Page 648 Wednesday, May 22, 2002 1:38 PM

There is a textbox for entering a user ID and a password control for entering a password. Both of these controls are HTML INPUT controls, as shown previously. The textbox runs at the server, and the password is an ordinary HTML control. Clicking the Login button (implemented as a Windows Forms Button control) results in very simple action. There is one legal password, hardcoded at "77." The button event handler checks for this password. If legal, it displays a welcome message; otherwise, an error message.

```
Private Sub btnLogin_Click(ByVal sender As Object, _
 ByVal e As EventArgs) Handles btnLogin.Click
   If Request.Params("txtPassword") = "77" Then
      lblMessage.Text = "Welcome, " & txtUserId.Value
   Else
      lblMessage.Text = "Illegal password"
   End If
End Sub
```

Since the password control is *not* a server control, no server control variable is available for accessing the value. Instead, we must rely on a more fundamental technique, such as using the **Params** collection.[10]

HTML CONTROLS IN VISUAL STUDIO

It is easy to work with HTML controls in Visual Studio.[11] The Toolbox has a palette of HTML controls, which you can access through the HTML tab. Figure 14–35 shows some of the HTML controls in the Visual Studio Toolbox.

You can drag HTML controls onto a form, just as we have done with Web Forms controls. You have the option of using FlowLayout or GridLayout. The default is GridLayout, which enables absolute positioning of controls on a form. FlowLayout is the simplest layout, resulting in elements positioned in a linear fashion. You can set the layout mode through the **pageLayout** property of the form. In our example we used FlowLayout for the two INPUT controls and their associated labels.

The default choice for HTML controls is not to run at the server. To make an HTML control into a server control, right-click on it in the Form Designer. Clicking on Run As Server Control toggles back and forth between running on the server and not running on the server. You can inspect the **runat** property in the Properties panel, but you cannot change it there.

---

10. We described the various collections earlier in the chapter in the section "Request/Response Programming." The collections are included in Table 14–1.
11. But it is also confusing, because there is only *one* palette for HTML controls, and you distinguish between classical HTML controls and server HTML controls by **runat="server"**. The Forms Designer UI for setting this attribute is described below.

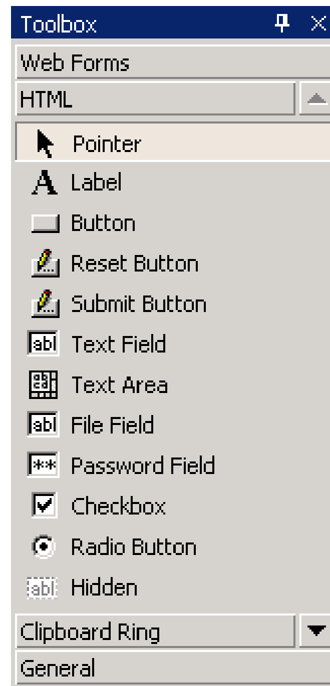


FIGURE 14–35 *HTML controls in the Visual Studio Toolbox.*

Validation Controls

The rest of our discussion of server controls will focus on Web controls. A very convenient category of control is the group of validation controls. The basic idea of a validation control is very simple. You associate a validation control with a server control whose input you want to validate. Various kinds of validations can be performed by different kinds of validation controls. The validation control can display an error message if the validation is not passed. Alternatively, you can check the **IsValid** property of the validation control. If one of the standard validation controls does not do the job for you, you can implement a custom validation control. The following validation controls are available:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator

There is also a **ValidationSummaryControl** that can give a summary of all the validation results in one place.

ch14.fm Page 650 Wednesday, May 22, 2002 1:38 PM

An interesting feature of validation controls is that they can run on either the client or the server, depending on the capabilities of the browser. With an upscale browser such as Internet Explorer, ASP.NET will emit HTML code containing JavaScript to do validation on the client.[12] If the browser does not support client-side validation, the validation will be done only on the server.

REQUIRED FIELD VALIDATION

A very simple and useful kind of validation is to check that the user has entered information in required fields. Our second server control demonstration page provides an illustration. Back on the top-level **ServerControls\WebForms1.aspx** page, follow the link to "Validation" (or click the Register button from the Login page). You will be brought to the page **RegisterNewUser.aspx**, as illustrated in Figure 14–36. The screenshot shows the result of clicking the Register button after entering a UserId, a Password, and a First Name, but leaving Last Name blank. You will see an error message displayed next to the Last Name textbox, because that is where the validator control is on the form.

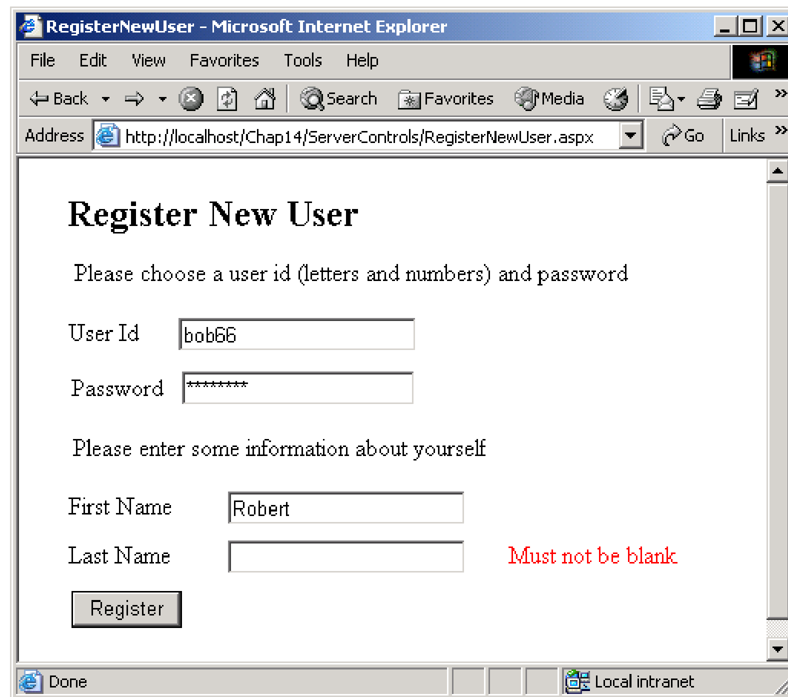


FIGURE 14–36 *Register New User page illustrates ASP.NET validation controls.*

12. Validation will also be done on the server, to prevent "spoofing."

ch14.fm Page 651 Wednesday, May 22, 2002 1:38 PM

The textboxes for First Name and Last Name both have an associated **RequiredFieldValidator** control. In Visual Studio you can simply drag the control to a position next to the associated control. You have to set two properties of the validator control:

- **ControlToValidate** must be set to the ID of the control that is to be validated.
- **ErrorMessage** must be specified.

Then, when you try to submit the form, the validator control will check whether information has been entered in its associated control. If there is no data in the control, the designated error message will be displayed.

Internet Explorer supports client-side validation using JavaScript. You can verify that ASP.NET generates suitable JavaScript by looking at the generated source code in the browser (View | Source).

This form also requires that the UserId field not be blank. Since the primary validation of this field is done by a regular expression validator, as discussed shortly, we will use another technique for the required field validation. Figure 14–37 shows the location of the various validator controls in the Visual Studio Form Designer.

FIGURE 14–37 *Layout of validation controls for Register New User page.*

ch14.fm Page 652 Wednesday, May 22, 2002 1:38 PM

We assign the ID **vldUserId** to the required field validator control associated with the UserId control, and we clear the error message. We also set the **EnableClientScript** property to **False**, to force a postback to the server for the validation. The event handler for the Register button then checks the **IsValid** property of **vldUserId**.

```
private void cmdRegister_Click(object sender,
                               System.EventArgs e)
{
   if (vldUserId.IsValid)
      lblMessage.Text = "Welcome, " + txtFirstName.Text;
   else
      lblMessage.Text = "UserId must not be blank";
}
```

If the control is valid, we display the welcome message; otherwise, an error message. Note that we won't even reach this handler if other validation is false.

REGULAR EXPRESSION VALIDATION

The **RegularExpressionValidator** control provides a very flexible mechanism for validating string input. It checks whether the string is a legal match against a designated regular expression. Our example illustrates performing a regular expression validation of UserId. The requirement is that the ID consist only of letters and digits, which can be specified by the regular expression

```
[A-Za-z0-9]+
```

The following properties should normally be assigned for a **RegularExpressionValidator** control:

- **ValidationExpression** (the regular expression, not surrounded by quotes)
- **ControlToValidate**
- **ErrorMessage**

You can try this validation out on our Register New User page by entering a string for UserId that contains a non-alphanumeric character.

Rich Controls

Another category of Web Forms controls consists of "rich controls," which can have quite elaborate functionality. The Calendar control provides an easy-to-use mechanism for entering dates on a Web page. Our third sample server control page provides an illustration, as shown in Figure 14–38.

The user can select a date on the Calendar control. The **SelectedDate** property then contains the selected date as an instance of the **DateTime** structure. You can work with this date by handling the **SelectionChanged**

ch14.fm Page 653 Wednesday, May 22, 2002 1:38 PM

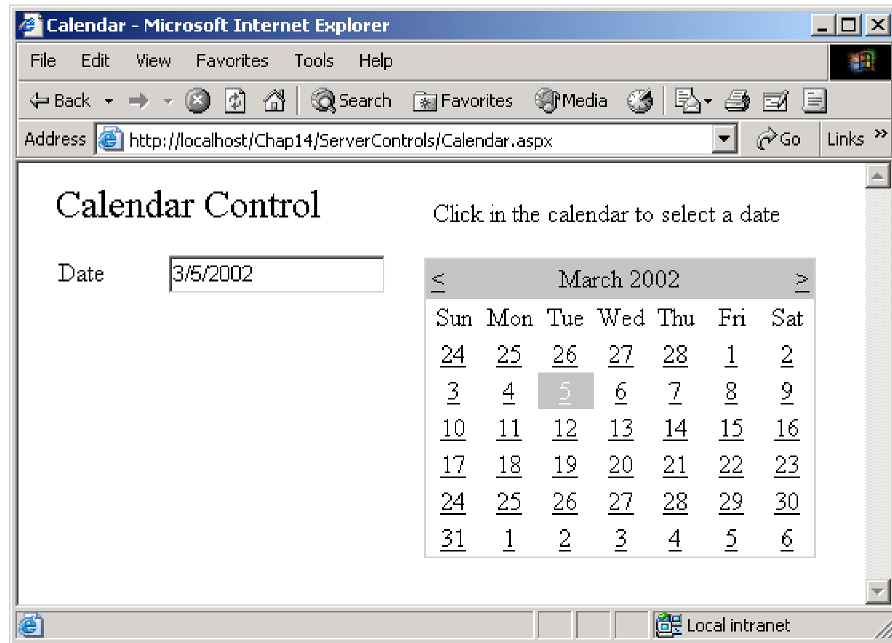


FIGURE 14–38 *Using the Calendar control to select a date.*

event. In our example page, the event handler displays the date as a string in a textbox.

```
Private Sub Calendar1_SelectionChanged( _
 ByVal sender As Object, ByVal e As EventArgs) _
 Handles Calendar1.SelectionChanged
   txtDate.Text = _
      Calendar1.SelectedDate.ToShortDateString()
End Sub
```

Database Access in ASP.NET

A great deal of practical Web application development involves accessing data in various kinds of databases. A great thing about the .NET Framework is that it is very easy to encapsulate a database, allowing the rest of the program to work with data in a very generic way, without worrying about where it came from. In this section we discuss data binding in Web Forms controls, and we then present a database version of our Acme Travel Agency Web site.

Data Binding in ASP.NET

ASP.NET makes it easy to display data from various data sources by permitting a Web Forms control to be bound to data source. The data source can be specified in a variety of ways—for example, by directly giving a connection string to a database. This form of data binding is quite convenient in a two-tier type of application, where the presentation layer talks directly to the database. In three-tier applications it is more convenient to bind to some data structure that is returned by a middle-tier component, which does the actual connection to the database. Our Acme case study illustrates this approach. The **Hotel.dll** and **Customer.dll** components encapsulate access to a SQL Server database through the **HotelBroker** and **Customers** classes. Methods such as **GetCities** return an **ArrayList**, and the array list can be bound to a Web Forms control.[13]

We will look at two examples of data binding. The first, mentioned earlier in the chapter, illustrates binding to an **ArrayList**. The second illustrates binding to a **DataTable** through a **DataView**.

BINDING TO AN ARRAYLIST

Code Example

It is extremely simple to bind to an array list. The **AcmeWeb** example, beginning with Step 0, provides an illustration. You may wish to bring up Step 0 and examine the code in **AcmeWeb\Step0\WebForm1.aspx.vb**. When the page is loaded, the **DropDownList** control **listCities** is initialized to display all the cities in the database of the hotel broker. The **GetCities** method returns the cities as strings in an array list. The following code will then cause the cities to be displayed in the dropdown.

```
broker = Global.broker
Dim cities As ArrayList = broker.GetCities()
listCities.DataSource = cities
...
DataBind()
```

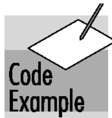
The **DataBind** method of the **Page** class causes all the Web Forms controls on the page to be bound to their data sources, which will cause the controls to be populated with data from the data sources. You could also call the **DataBind** method of a particular control.

---

13. The component could be hidden behind a Web service, which will be illustrated in Chapter 15. We can still use data binding in such a scenario by binding to an array list.

BINDING TO A DATATABLE

As we saw in Chapter 13, ADO.NET defines a very useful class, the **DataTable**, which can be used to hold data from a variety of data sources. Once created, a data table can be passed around and used in a variety of contexts. One very useful thing you can do with a data table is to bind it to a Web Forms control. Since a data table is self-describing, the control can automatically display additional information, such as the names of the columns. We illustrate with the **DataGrid** control.



To run this example, you need to have SQL Server or MSDE installed on your system, and you should also have set up the Acme database, as described in Chapter 13. The example Web page is **DataGridControl/ShowHotels.aspx**. As usual, you should use IIS to configure the folder **DataGridControl** as an application. This page will display all the hotels in the Acme database in a data grid, with appropriate headings, as illustrated in Figure 14–39. When you work with Web Forms controls you can easily change styles, such as fonts and colors, by setting properties appropriately.

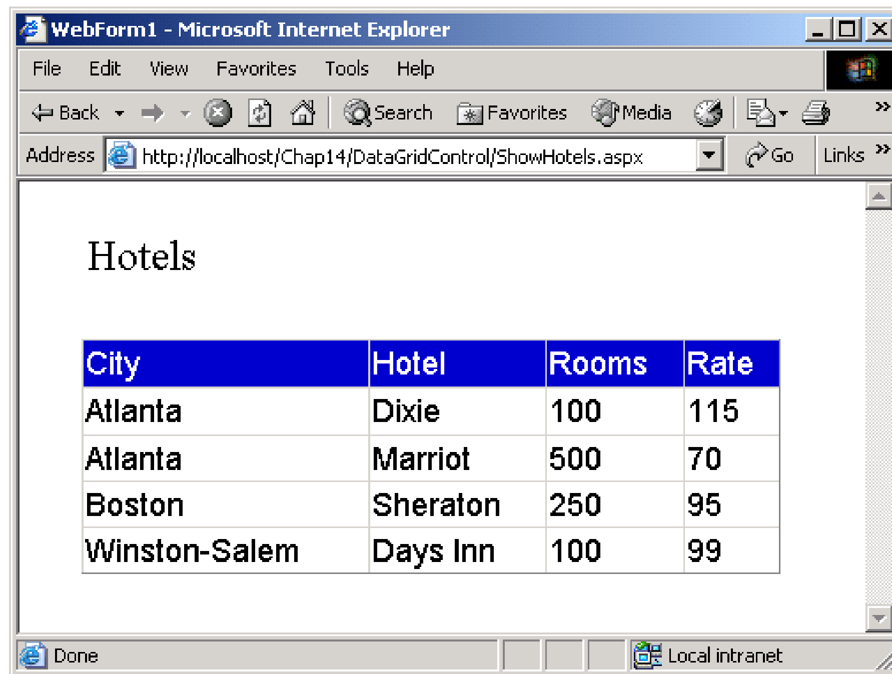


FIGURE 14–39 *Displaying hotels in the Acme database using a DataGrid control.*

ch14.fm Page 656 Wednesday, May 22, 2002 1:38 PM

The relevant VB.NET code is in the files **Global.asax.vb** and **ShowHotels.aspx.vb**. The first thing we need to do is to create an instance of the **HotelBroker** class. We create a single instance, once, when the application starts up. We save this instance as a public shared variable.

```
' Global.asax.vb

Imports System.Web
Imports System.Web.SessionState
Imports OI.NetVb.Acme

Public Class Global
    Inherits System.Web.HttpApplication

#Region " Component Designer Generated Code "
...

   Public Shared broker As HotelBroker

   Sub Application_Start(ByVal sender As Object, _
    ByVal e As EventArgs)
      broker = New HotelBroker()
   End Sub
   ...
```

In the **Page_Load** method (in file **ShowHotels.aspx.vb**) we get the hotels from the Hotel Broker, call a helper method to obtain the data source, assign the data source, and bind. We are using the **DataTable** to hold data obtained from the middle-tier component.

```
Private Sub Page_Load(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load
   If Not IsPostBack Then
      broker = Global.broker
      Dim arr As ArrayList = broker.GetHotels()
      dgHotels.DataSource = CreateDataSource(arr)
      dgHotels.DataBind()
   End If
End Sub
```

It is in the helper method **CreateDataSource** that the interesting work is done. A data table is created and populated with hotel data obtained from the Hotel Broker.

```
Private Function CreateDataSource( _
 ByVal list As ArrayList) As ICollection
   If list Is Nothing Then
      Return Nothing
   End If
```

```
   Dim dt As New DataTable()
   Dim dr As DataRow

   dt.Columns.Add(New DataColumn("City", GetType(String)))
   dt.Columns.Add(New DataColumn("Hotel", GetType(String)))
   dt.Columns.Add(New DataColumn("Rooms", _
      GetType(Integer)))
   dt.Columns.Add(New DataColumn("Rate", GetType(Decimal)))

   Dim hi As HotelListItem
   For Each hi In list
      dr = dt.NewRow()

      dr(0) = hi.City.Trim()
      dr(1) = hi.HotelName.Trim()
      dr(2) = hi.NumberRooms
      dr(3) = hi.Rate

      dt.Rows.Add(dr)
   Next

   Dim dv As New DataView(dt)
   Return dv
End Function
```

Acme Travel Agency Case Study (Database Version)

We have illustrated many concepts of ASP.NET with our Acme Travel Agency case study. For simplicity we used a version of the case study that stored all data as collections in memory. This way you did not have to worry about having a database set up properly on your system, so you could focus on just ASP.NET. Also, the results are always deterministic, since sample data is hardcoded.

Now, however, we would like to look at the “real” case study, based upon our HotelBroker database, and the database version of the **Hotel.dll** and **Customer.dll** components created in Chapter 13.

ACMECUSTOMER DATABASE

The Acme Travel Agency maintains its own database of customers. Customers register with Acme through the Web site. The following information is stored in Acme’s database:

- LoginName
- Password
- HotelBrokerCustomerId
- AirlineBrokerCustomerId

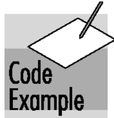
Currently we use LoginName (corresponding to what we called “UserId” earlier in the chapter) and HotelBrokerCustomerId. The AirlineBrokerCusto-

ch14.fm Page 658 Wednesday, May 22, 2002 1:38 PM

merId field will facilitate Acme adding an airplane reservation system later. A Password field is also provided for possible future use.

The **AcmeCustomer** database should have been set up as part of the database setup from Chapter 13. To set up the **AcmeCustomer** database (or restore it to its original state), all you need to do is to run the script **acmedb.sql**, which is located in the **Databases** directory from Chapter 13. This script assumes you have SQL Server installed on partition **c:**. If your installation is in a different partition, edit the script accordingly.

ACME WEB SITE (CASE STUDY)



The Case Study version of the Acme Web site is in the **CaseStudy** folder for this chapter. As usual, you will need to use IIS to configure this directory as an application. You can start from the home page for this chapter, or directly from the URL

```
http://localhost/netcs/CaseStudy/Main.aspx
```

You should find the code very easy to understand, because it relies on the same interfaces as the implementation we used earlier based on collections. The database code for accessing the **AcmeCustomer** database is in the file **Acme.vb**.

Summary

ASP.NET is a unified Web development platform that greatly simplifies the implementation of sophisticated Web applications. In this chapter we introduced the fundamentals of ASP.NET and Web Forms, which make it easy to develop interactive Web sites. Server controls present the programmer with an event model similar to what is provided by controls in ordinary Windows programming. This high-level programming model rests on a lower-level request/response programming model that is common to earlier approaches to Web programming and is still accessible to the ASP.NET programmer.

The Visual Studio .NET development environment includes a Form Designer, which makes it very easy to visually lay out Web forms, and with a click you can add event handlers. ASP.NET makes it very easy to handle state management. Configuration is based on XML files and is very flexible. There are a great variety of server controls, including wrappers around HTML controls, validation controls, and rich controls such as a Calendar. Data binding makes it easy to display data from a variety of data sources.

In the next chapter we cover Web Services, which enable the development of collaborative Web applications that span heterogeneous systems.