Introduction to ASP.NET

Objectives

- Review the features and shortcomings of classic Active Server Pages.
- Understand the advantages of an ASP.NET application.
- Learn about server controls and events in ASP.NET.
- Create a simple Web Service in ASP.NET.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

A Review of Classic ASP

You may be an experienced developer of Web applications that use a previous version of Microsoft Active Server Pages, or "classic" ASP. Or, you may be just getting started as a Web developer. In either case, it is useful to review the Active Server Page technology that preceded ASP.NET.

Dynamically Creating Web Pages

ASP is a technology that Microsoft created to ease the development of interactive Web applications. When the Internet began, it was used to transmit static HTML pages in response to HTTP requests from a browser. These Web pages included hyperlinks that allowed users to navigate easily from page to page, but that was about the extent of their interactivity.

As the Internet evolved, Web publishers developed ever more sophisticated ways of allowing users to have more control over the pages they received. HTML forms allowed users to enter information using simple Windows-like controls such as text boxes, list boxes, and check boxes. The data from these controls is embedded in the HTTP request that is issued when the user clicks a Submit button.

Web server software execution technologies like CGI (Common Gateway Interface) evolved to meet the need for more interactivity by intercepting certain Web requests and running programs that created and returned custom Web pages, usually based on data retrieved from a database.

After a couple of false starts (anyone remember ADC/HTX files?), Microsoft settled on Active Server Pages as their way of supporting dynamic creation of HTML pages based on user input.

An ASP Example

*See **Products-ASP.asp***

Figure 1 shows the results of running a typical ASP application. Microsoft installs an ISAPI application with IIS that intercepts all requests for pages with an extension of .asp.

These requests and the pages they point to are then handled within the ASP run-time environment. This execution environment allows pages to contain code in special script blocks delimited by <% and %> characters or contained within <script> elements that include the attribute, runat=server. This script performs whatever processing tasks are necessary to generate a custom HTML page in response to the user's request.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

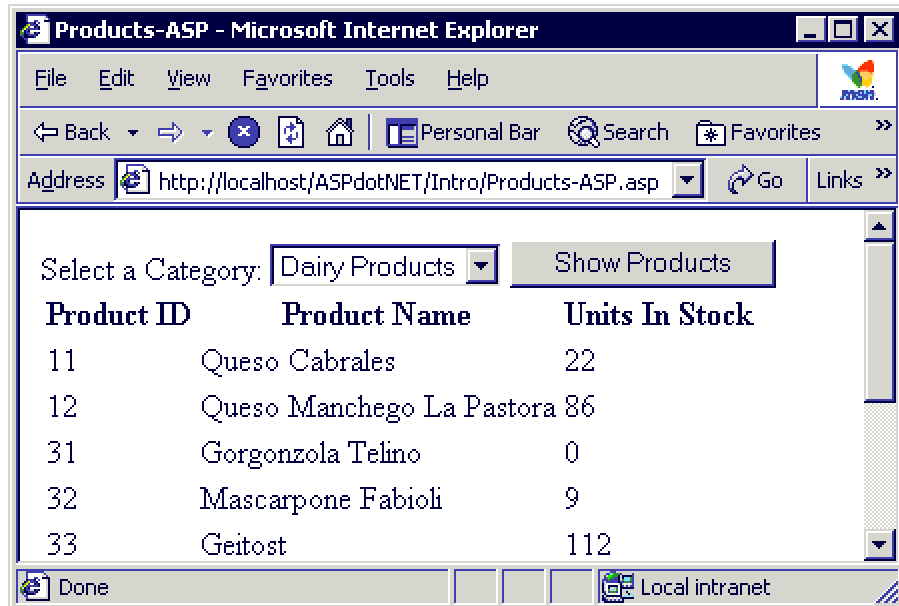

Figure 1. A typical "classic" Active Server Page.

The ASP page in this example begins with a couple of script-related lines that identify the language and specify that all variables will be explicitly declared:

```
<%@ Language=VBScript %>
<%Option Explicit%>
```

The next lines are typical HTML that creates a title, defines an HTML form, adds some text to the page, and begins a drop-down list box:

```
<html>
<head>
<title>Products-ASP</title>
</head>
<body>
  <form action="Products-ASP.asp" method="post">
    Select a Category:
    <select name="Category">
```


Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Embedding Data Access Code in the Page

Now, the page gets more interesting (and more complicated!), using ADO objects to retrieve a list of product categories from the Northwind database on a local SQL Server:

```
<%
  Dim cnn, cmd, rst
  Set cnn=Server.CreateObject("ADODB.Connection")
  ' Adjust user name and password, if necessary
  cnn.Open "Provider=SQLOLEDB.1;Data Source=(local);" _
    & "Initial Catalog=Northwind;User ID=sa;Password=;"
  ' Open a read-only, forward-only, server-side recordset.
  Set cmd=Server.CreateObject("ADODB.Command")
  cmd.CommandText= _
    "SELECT CategoryID, CategoryName FROM Categories" _
    & " ORDER BY CategoryName;"
  cmd.ActiveConnection=cnn
  Set rst=cmd.Execute
```

The ASP Object Model

The code above uses the CreateObject method of the ASP Server object to instantiate the ADO objects used for data access. In addition to providing a script execution engine, ASP also provides a set of six objects, including Server, to facilitate the development of Web applications. Here is a brief summary of these objects:

- The **Request** object is used to read data that was packaged inside the HTTP request for the page.
- The **Response** object allows you to inject data, including HTML, cookies, or redirection headers into the response stream that is sent back to the client's browser.
- A **Session** object is created when the first request from a particular client is processed, and it stays in scope until a timeout period expires following the last request from that user, allowing you to store data and objects that span multiple requests from one user.
- The **Application** object is similar to the Session object, but its data is shared across all client requests over the lifetime of the application, and it also allows you to write code that runs automatically when the applications starts or ends.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

- The **ObjectContext** object is used to commit or abort transactions managed by MTS or COM+.

- The **Server** object provides a set of generic utility methods for creating COM objects, encoding data as HTML or URL strings that can be embedded within the HTML sent back to a browser, and finding the actual file locations that correspond to virtual paths.

Getting back to our sample, the next lines of code use the ASP Response object to inject HTML into the HTTP response being sent back to the browser. The category ID from each row in the recordset is assigned as the value for each row of the drop-down list box:

```
' Use the ADO recordset to populate the dropdown list.
Do Until rst.EOF
    Response.Write( _
      "<option value=""" & rst("CategoryID") & """")
```

Handling ASP Postbacks

A *postback* is what happens when an Active Server Page creates HTML that allows the user to call the same ASP all over again. In the example, the page Products-ASP.asp is reloaded every time the user clicks the Show Products button. This happens because the page's address appeared in the action attribute of the opening tag for the HTML form:

```
<form action="Products-ASP.asp" method="post">
```

When the user clicks the button a postback occurs, and the drop-down list is populated all over again. But you want the users to see the category when they get the new page, which will also now show them the products in that category. The remainder of this section of code ensures that the category matching the one in the HTTP request will be selected in the new page, and it also adds the category name as the text for each row in the drop-down list box:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
' Preserve the selected category during postbacks
    If cstr(rst("CategoryID")) = Request("Category") then
      Response.Write(" selected>")
    Else
      Response.Write(">")
    End If
    Response.Write(rst("CategoryName") & _
      "</option>" )
    rst.MoveNext
  Loop
  rst.Close
  Set rst=Nothing
  Set cmd = Nothing
%>
```

Mixing Code and HTML

The page then returns to HTML for a couple of lines for the closing select tag (ending the rendering of the drop-down list box) and the creation of the submit button:

```
    </select>
    <input type="submit" value="Show Products">
```

The rest of the page runs only during a postback, after the user selects a category. When users first call up the page, only the drop-down box and the button appear. After they pick a category and click the button, they get back a list of all the products in that category, formatted as an HTML table. To accomplish this, the ASP alternates between sections of code and sections of literal HTML, building the table dynamically in response to the user's request:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
<%
  ' Check if a category was selected.
  If Len(Request("Category"))>0 Then
    ' Create client-side, disconnected recordset
    '   of products having the selected category.
    Set rst = Server.CreateObject("ADODB.Recordset")
    rst.CursorLocation=3 'adUseClient
    rst.Open _
      "SELECT ProductID, ProductName, UnitsInStock" _
      & " FROM Products WHERE CategoryID=" _
      & Request("Category"), _
      cnn, 3, 1 'adOpenStatic, adLockReadOnly
    Set rst.ActiveConnection = Nothing
    cnn.Close
    Set cnn = Nothing
%>
    <table>
      <tr>
        <th>Product ID</th>
        <th>Product Name</th>
        <th>Units In Stock</th>
      </tr>
<%
  ' Add a table row for each recordset row.
    Do Until rst.EOF
%>
      <tr>
        <td> <% =rst("ProductID") %> </td>
        <td> <% =rst("ProductName") %> </td>
        <td> <% =rst("UnitsInStock") %> </td>
      </tr>
<%
       rst.MoveNext
    Loop
    rst.Close
    Set rst = Nothing
%>
    </table>
```


Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
<%
  End If
  If Not cnn Is Nothing Then
    cnn.Close
    Set cnn=Nothing
  End If
%>
  </form>
</body>
</html>
```

This code sample shows an alternative to using Response.Write for building up HTML with dynamically embedded data. Snippets of code can be intermixed with snippets of literal HTML to generate the final stream that is sent to the client.

What the Client Sees

The client that calls an Active Server Page never sees the code that runs on the server. The browser only sees the HTTP response that is returned. Here's the client-side source for the page shown in Figure 1, which was created by the code you just reviewed:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
<html>
<head>
<title>Products-ASP</title>
</head>
<body>
  <form action="Products-ASP.asp" method="post">
    Select a Category:
    <select name="Category">
<option value="1">Beverages</option<option
value="2">Condiments</option><option
value="3">Confections</option><option value="4"
selected>Dairy Products</option><option
value="5">Grains/Cereals</option><option
value="6">Meat/Poultry</option><option
value="7">Produce</option><option
value="8">Seafood</option>
    </select>
    <input type="submit" value="Show Products">
    <table>
      <tr>
        <th>Product ID</th>
        <th>Product Name</th>
        <th>Units In Stock</th>
      </tr>
      <tr>
        <td> 11 </td>
        <td> Queso Cabrales </td>
        <td> 22 </td>
      </tr>
(subsequent rows in the table omitted for brevity)
          </table>
  </form>
</body>
</html>
```


Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

In this simple example, the HTML delivered to the client is very simple, but your ASP application can embed styles, graphics, hidden form controls, and even client-side script, to create complex and full-featured Web pages.

ASP Shortcomings

As useful and successful as ASP has been, it suffers from several important limitations that motivated the development of ASP.NET.

Interpreted and Loosely-Typed Code

The script-execution engine that Active Server Pages relies on interprets code line by line, every time the page is called. In addition, although variables are supported, they are all loosely typed as variants and bound to particular types only when the code is run. Both these factors impede performance, and late binding of types makes it harder to catch errors when you are writing code. The lack of Microsoft IntelliSense support in scripting environments also hinders programmer productivity.

To overcome this limitation, many ASP developers have tried moving as much logic as possible into COM automation components, such as ActiveX DLLs created by Microsoft Visual Basic. Unfortunately, this practice results in one very undesirable side effect: Once the DLL is loaded by ASP, it remains in memory until the Web server is shut down, making it very hard to maintain ASP applications without periodically bringing down the server.

Collaboration Is Difficult

Most Web development teams include two separate camps, which have jokingly been referred to as the “black socks” and the “pony tails.” The nerdy programmers in their black socks are responsible for writing code to fetch and massage data, while the hip, pony-tailed designers are busy making the site beautiful and interesting.

The way that ASP encourages the intermixing of code and HTML makes it difficult for programmers and designers to collaborate on a page without messing up each other’s work. To some extent, code can be segregated into script blocks or encapsulated in COM objects, but some intermixing is inevitable in most ASP projects, and this makes team development difficult.

Limited Development and Debugging Tools

Microsoft Visual InterDev, Macromedia Visual UltraDev, and other tools have attempted to increase the productivity of ASP programmers by providing graphical development environments. However, these tools never achieved the ease of use or the level of acceptance achieved by Microsoft Windows application development tools, such as Visual Basic or Microsoft Access. Most

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

ASP developers still rely heavily or exclusively on Notepad. This occurs in part because the typical ASP page combines elements from a variety of different technologies, including VBScript, JavaScript, Cascading Style Sheets, Dynamic HTML, ADO, and lately even XML and XSL style sheets. Handling all these technologies well would be a tough assignment for any single development tool.

Debugging is an unavoidable part of any software development process, and the debugging tools for ASP have been minimal. Most ASP programmers resort to embedding temporary Response.Write statements in their code to trace the progress of its execution.

Server Affinity when Maintaining Session State

As traffic to a Web site grows, one of the best ways to handle the growth is to build a “Web farm” of multiple, clustered Web servers. Recent hardware and software innovations have made this a very affordable and effective way to add capacity and to dynamically balance the load. This also increases reliability and availability, because one or even multiple servers can fail without bringing down the entire site.

ASP was created before this architecture became common, and its way of maintaining session state makes it very hard to achieve optimal load balancing across a cluster of servers. The reason for this is that session data is stored on the server that processed the first request of the session. As multiple requests come back from that user, those requests need to be routed to the server that holds that session’s state.

Enterprising developers have developed clever ways to work around this problem. For example, you can encapsulate session data in a hidden control on the page that is returned to the user, and then retrieve it when the next request comes in. However, ASP’s inability to make optimum use of load balancing across clusters is still an important limitation.

Obscure Configuration Settings

To store configuration settings, ASP uses the metabase, which is a hierarchical, registry-like repository that is part of IIS. Many developers find the structure of the metabase hard to understand and hard to navigate. In addition, it is difficult to transfer metabase settings from one Web server to another.

Code Required for Postbacks and Multibrowser Support

As the Products-ASP sample page demonstrated, it is common for pages to be reloaded repeatedly as users enter data and make selections. To keep the users from losing their previous entries each time this happens, you must write code

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

to capture and reload their existing entries. In complex data entry pages, this code can become quite extensive.

For an intranet site, you may be able to assume that all users have recent versions of Internet Explorer, but for a public Internet site, you can't make this assumption. Unless you limit your pages to using only the basic HTML features, you must write code to detect the user's browser and alter the HTML output accordingly. This creates a lot of extra work for ASP programmers who need to support multiple browsers. Similarly, there is no support for mobile devices, such as PDAs and phones.

ASP.NET to the Rescue

ASP.NET was developed in direct response to the problems that developers had with classic ASP. Although the .NET Framework has come to include much more than just ASP.NET, the original impetus for its creation was the need for a new way of efficiently building modern, scalable Web applications. So, it's not surprising that ASP.NET directly and handily addresses all of the shortcomings of ASP.

Compiled and Maintainable Code

ASP.NET makes use of the .NET Common Language Runtime (CLR), rather than a scripting engine. This means that the full power of all .NET languages is available to ASP.NET developers, and the code is compiled using the standard .NET just-in-time compilation process, offering optimum performance and type safety.

Furthermore, ASP.NET takes advantage of the efficiency and maintainability of object-oriented programming. Every ASP.NET page that you create results in the creation of a class derived from the generic System.Web.UI.Page class. An instance of this class is what generates your ASP.NET page each time it is called.

When you need to replace a component of your ASP.NET application, you can do so without bringing down the server. In fact, you can simply copy the new file over the old one—ASP.NET will continue to service any requests that were using the old component until they are finished, and it will use the new version to process all new requests. This is possible because the CLR allows multiple versions of a component to coexist, even within the same process.

Separation of Code from HTML

The ASP.NET architecture creates a clear separation between the HTML and code elements of each page, and it allows you to store these elements in separate files. This makes it much easier for teams of programmers and designers to collaborate efficiently.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Graphical Development Environment

Visual Studio .NET provides a very rich development environment for Web developers. You can drag and drop controls and set properties the way you do in Visual Basic. And you have full IntelliSense support, not only for your code, but also for HTML and XML.

In addition, the Visual Studio .NET development environment is programmable and extensible, so you can expect a steady stream of new development aids to become available, both from Microsoft and from independent software vendors.

Cluster-Friendly State Management

ASP.NET does not force you to store session state on the server that processes a user's initial request in a session. Instead, you can store state out-of-process, either in memory or in a SQL Server database. This session management scheme supports load balancing in server farms, and it even allows your session data to survive a crash of the server that created it.

XML-Based Configuration Files

Configuration settings in ASP.NET are stored in XML files that you can easily read and edit. You can also easily copy these to another server, along with the other files that comprise your application.

Automatic Handling of Postbacks and Multiple Browsers

ASP.NET automatically includes a hidden control on Web pages, holding in compressed form the state of all the controls on the form that have enabled this feature. ASP.NET uses this *viewstate* data to fill in the user-entered data automatically and efficiently.

ASP.NET is also capable of automatically detecting the browser or device that was used to issue a request and tailoring the output to match the capabilities of the client.

And It's Fun!

Many experienced programmers have shied away from Web development, despite its growing importance, because the process of building a Web application has been so ugly and inelegant. Unlike software development that revolves around a single language, such as Visual Basic or C++, the Web world forces you to mix and match a number of unrelated languages and technologies. In short, it's been a mess!

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

ASP.NET doesn't completely eliminate the need for you to understand Web-standard technologies, including HTML, XML, and CSS, in addition to your programming language of choice. But you will find that ASP.NET does an impressive job of bringing order to the world of Web development, with a well-thought-out framework that encompasses all these technologies and integrates them within a development environment that feels much more productive—and, yes, more fun—than what was previously available.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

ASP.NET Web Applications

ASP.NET is not simply a new improved version of ASP—it is a completely new product that relies on a completely new architecture for building Web applications. A good way to get a feel for this new architecture is to create a new ASP.NET Web application in Microsoft Visual Studio.

Creating a New Web Application

In this presentation, we are using Visual Studio .NET and the Visual Basic .NET language. However, Visual Studio is not required for building ASP.NET applications. You can create these files in Notepad or with your favorite editor, and you can compile them into an application using command-line tools that are available free with the .NET Framework. Once you have used Visual Studio, however, you probably won't want to give it up.

You can also use other .NET languages besides Visual Basic. The most common alternative to Visual Basic is C#, a new language with C-based syntax that is very similar to Java. All the features of ASP.NET are available regardless of the language you use.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Try It Out!

1. Open Visual Studio and select **File|New|Project**, or press **CTRL+SHIFT+N**. You'll see the New Project dialog box shown in Figure 2.

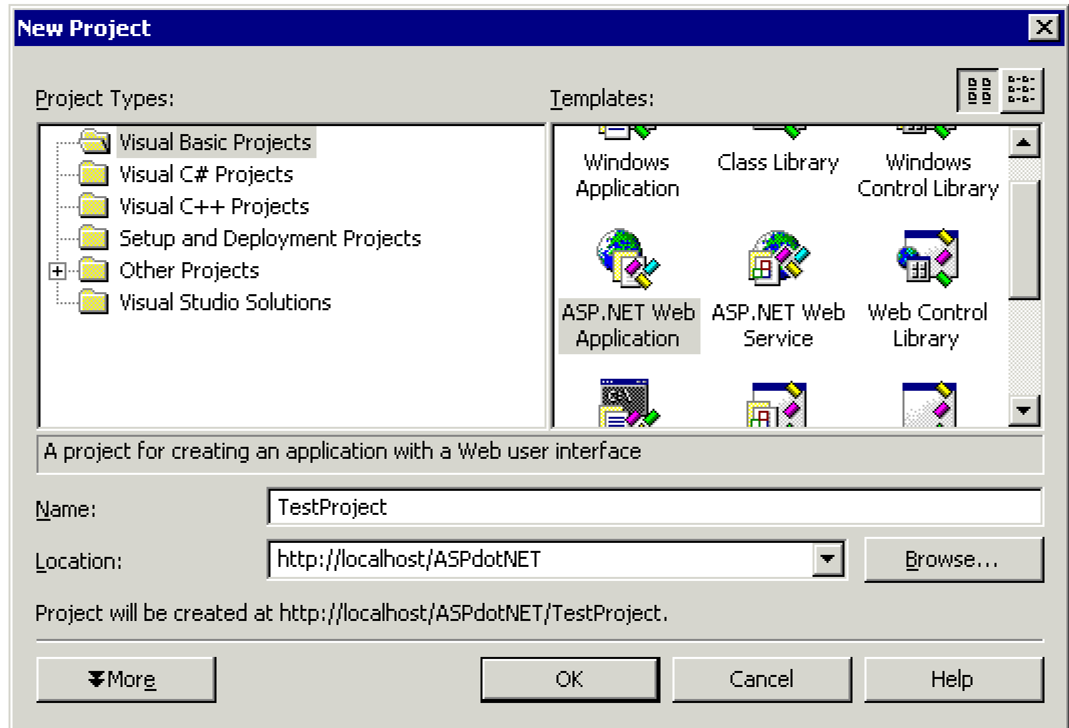

Figure 2. Creating a new Web application in Visual Studio .NET.

2. Name your new project **TestProject** and locate it in the virtual root directory, **http://localhost/ASPdotNET**. A new subdirectory will automatically be created for your new application.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

3. A new ASP.NET page, or Web Form, called WebForm1.aspx, is automatically created for you. The designer for this page is shown in Figure 3. The text in the designer informs you that you are working in grid layout mode, with absolute positioning. If your form defaults to FlowLayout mode, then use the Properties window to switch the pageLayout property to GridLayout.

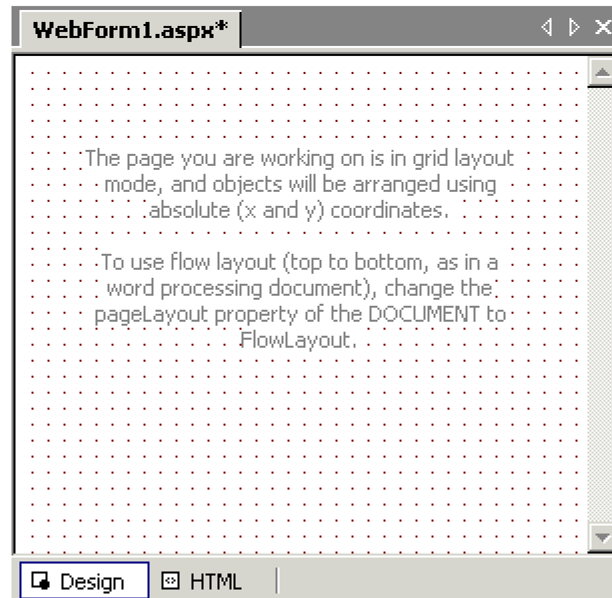

Figure 3. The Web form designer in Visual Studio .NET.

4. Select **Format|SnapToGrid**, to make it easy to line up controls on the form.

5. If necessary, use the View menu to bring up the Toolbox. Select the **Web Forms** category, and drag a **TextBox**, a **Button**, and a **Label** onto your form.

6. Select the three controls by holding the **SHIFT** key down and clicking on them, or by using the mouse pointer to click and drag a bounding rectangle around them. Using the Format menu, select **Align|Lefts**, **Make Same Size|Width**, and **Vertical Spacing|Make Equal**. Your form should look like Figure 4.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

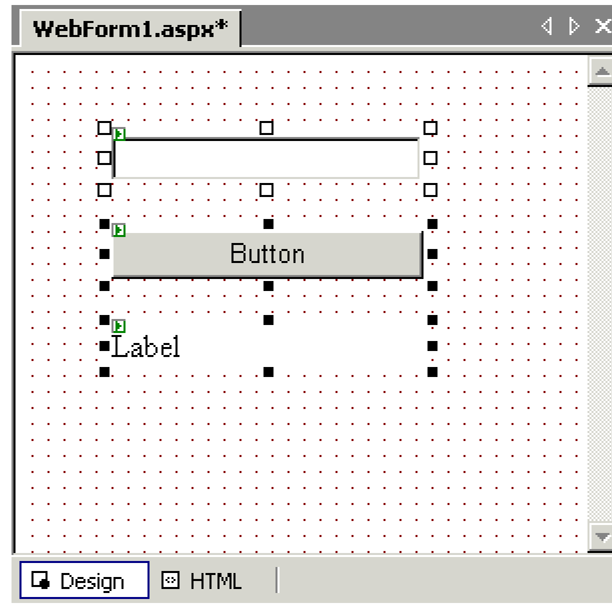

Figure 4. Laying out ASP.NET server controls.

7. Now, click on the HTML tab at the bottom of the form designer. You'll see the HTML that corresponds to the controls you created. Add an instructional heading by entering this below the opening form tag:

```
<h3>Enter your name and click the button.</h3>
```

Notice that the closing tag is added for you automatically, and you get IntelliSense support when you edit HTML, as shown in Figure 5.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

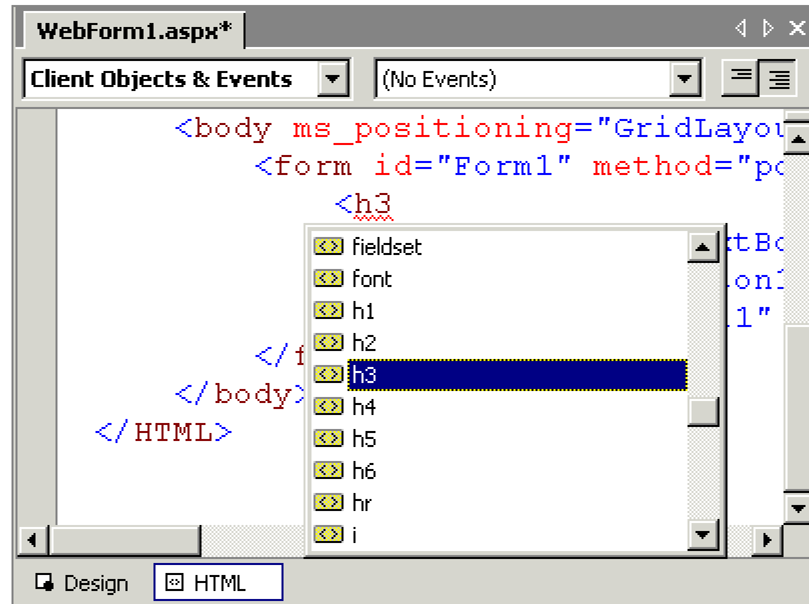

Figure 5. The HTML editor supports IntelliSense.

8. Switch back to the **Design** pane, select the label, and use the Properties window to delete the text. Select the button and change the Text property to “Click Here.” You can switch back to the HTML pane to see how these changes are immediately reflected in the HTML—you could have made the changes there initially, if you wanted to.

9. In the **Design** pane, double-click on your button control. This brings up the code window. In the **Button1_Click** procedure, add this line, as shown in Figure 6:

```
Me.Label1.Text = "Hello, " & Me.TextBox1.Text & "!"
```


Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

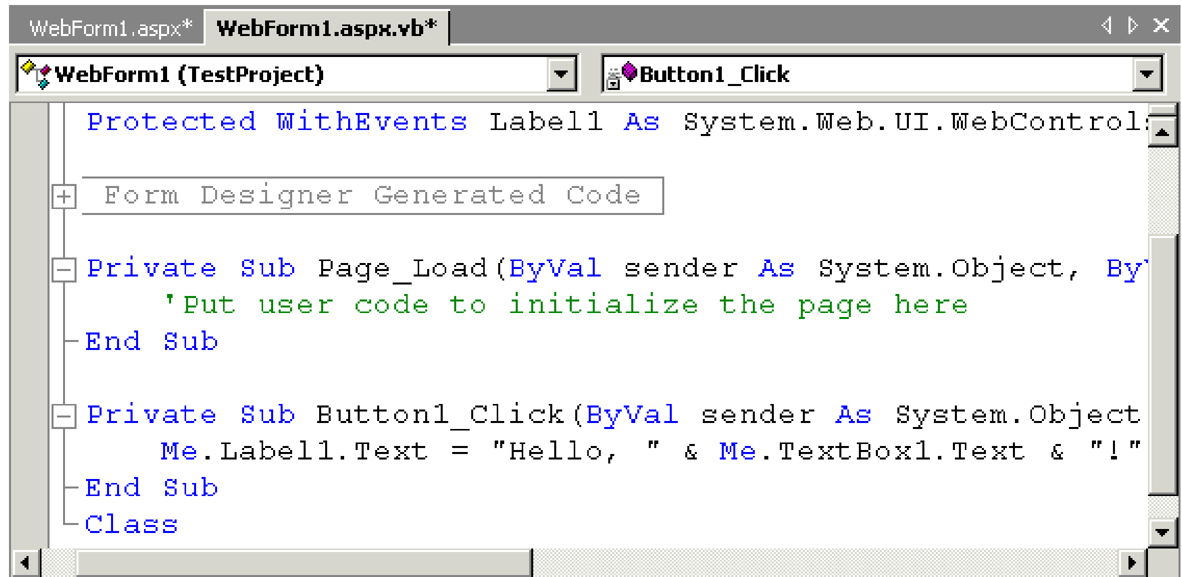

Figure 6. Visual Studio automatically creates a separate file for your code.

The code window shows you the contents of the code-behind file, WebForm1.aspx.vb. Every Web form you create in ASP.NET has a corresponding code file, which creates the class that ASP.NET uses to render the page. To see the code files listed separately in the Solution Explorer, click the **Show All Files** button at the top of the Solution Explorer window, as shown in Figure 7. You can also choose **Project|Show All Files** from the menu.

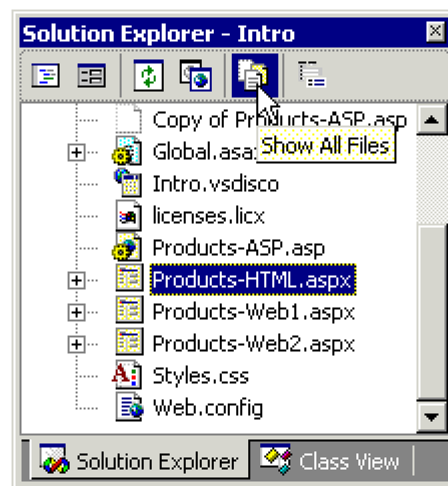

Figure 7. Click the Show All Files button to include code files in the Solution Explorer.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

10. Select **File|Save All**. Then, in the Solution Explorer window, right-click on WebForm1.aspx, and choose **Build and Browse**. Type in your name, and click the button. The completed page is shown in Figure 8. You can also try using Internet Explorer to navigate to

http://localhost/ASPdotNET/TestProject/WebForm1.aspx.

Figure 8. The finished Web page, running inside Visual Studio.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Rendering HTML with Server Controls

Right-click in the browser window where your finished page is displayed, and select **View Source** to see the HTML that was generated. You'll notice that the HTML has changed quite a bit from what you saw in the HTML editor in Visual Studio.

In Visual Studio, the label appeared as an element like the following:

```
<asp:label id="Label1" style="LEFT: 48px; POSITION:
 absolute; TOP: 136px" runat="server"
 Width="156px"></asp:label>
```

In the browser source, you'll see something like this:

```
<span id="Label1" style="width:156px;Z-INDEX: 103; LEFT:
 48px; POSITION: absolute; TOP: 136px">Hello, Andy!</span>
```

The label is an example of an ASP.NET *server control*. Note that the HTML in the Design pane included the attribute runat="server". This attribute marks the HTML element as a server control that ASP.NET renders into custom HTML when the page is called. In this case, it became an HTML span with an ID corresponding to the name of the control and a style attribute that defines its position and size.

If a user calls this page from an older browser that doesn't support absolute positioning, ASP.NET detects this and renders the HTML differently. The following steps let you test this without installing an older browser:

Try It Out!

1. View the Design pane for your page in Visual Studio, and click in a blank area of the page. This selects the page as a whole. In the Property window, you should see the properties for the Document. You can also achieve this by selecting **Document** from the drop-down list at the top of the Properties window. Set the **targetSchema** property to **Internet Explorer 3.02/Navigator 3.0**, as shown in Figure 9.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

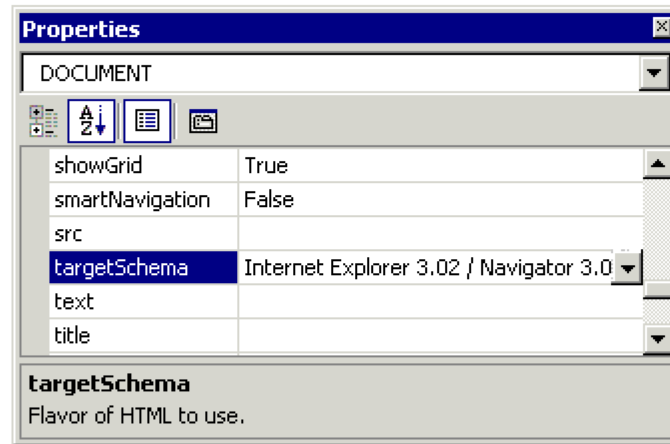

Figure 9. The targetSchema property affects the HTML that is rendered.

2. Save the project, right-click on the page in the Solution Explorer, and select **Build and Browse**.

3. In the Browser window, right-click and select **View Source**. The older browsers corresponding to this targetSchema do not support absolute positioning. So, instead of using positioning values in style attributes, ASP.NET accomplishes the positioning by creating an HTML table. Here's how the label now appears:

```
    <TR vAlign="top">
       <TD colSpan="2" height="20">
       </TD>
       <TD>
          <span id="Label1" style="width:156px;">
           Hello, Andy!</span>
       </TD>
    </TR>
</TABLE>
```

This type of HTML would be generated automatically if an older browser accesses your page, even with the original targetSchema setting.

Types of Server Controls

ASP.NET makes use of two broad categories of server controls that appear in separate sections of the Toolbox: HTML controls and Web Forms controls, often called simply Web controls.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

The *HTML controls* are very similar to the standard controls that you use when writing HTML. Their properties correspond to the HTML attributes you set when formatting the standard controls. However, unlike standard HTML controls, these controls are rendered differently by ASP.NET, depending on the client browser.

Web controls are distinguished by an *asp:* prefix in the HTML editor. Like HTML controls, these controls must send standard HTML to the browser, since that is, of course, all that browsers understand. But these controls support properties, methods, and events that go far beyond what is available in standard HTML. This gives you much more freedom and power when creating Web pages, and you can let ASP.NET handle the messy details of rendering the resulting HTML.

ASP.NET also allows you to create your own custom controls. This feature is somewhat analogous to the way that Visual Basic 6 allows you to create your own ActiveX controls.

In most cases, you will only use HTML controls for backwards compatibility to facilitate migration of existing pages. Given a choice, you will probably want to use the more full-featured Web controls when you can.

Remember that you are not limited to using server controls in ASP.NET. You can edit the HTML and include standard HTML tags wherever you want, as you did when you added the <h3> text at the top of the sample form.

At the start of this chapter, you saw an example of a classic ASP application that retrieved product data from the Northwind database. Now, you'll see examples of how to build that application in ASP.NET, first using HTML controls and then using Web controls.

Using HTML Controls

*See **Products-HTML.aspx.vb***

The Intro example project includes a Web form named Products-HTML.aspx. This page behaves very much like the Products-ASP.asp page, but it achieves that behavior very differently. The HTML for this page is very simple:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
<%@ Page Language="vb" CodeBehind="Products-HTML.aspx.vb"
AutoEventWireup="false" Inherits="Intro.Products_HTML" %>
<HTML>
   <HEAD>
      <title>Products-HTML</title>
      <META http-equiv="Content-Type" content="text/html;
       charset=windows-1252">
   </HEAD>
   <body>
      <form runat="server">
         Select a Category:
         <select ID="Category" runat="server"> </select>
         <input type="submit" value="Show Products"
          runat="server">
         <table ID="Products" runat="server">
         </table>
      </form>
   </body>
</HTML>
```

All the work to retrieve data from the database and to fill the list box and table occurs in the code contained in Products-HTML.aspx.vb, which is referenced in the directive at the top of the HTML.

Working with HTML Server Controls in Code

The HTML server controls provide object models that you can program against in the code-behind class that the HTML page “inherits.” Visual Studio does all the work for you of creating this code file, complete with a class for your page that is derived from the System.Web.UI.Page class in the .NET Framework. You’ll also automatically get a variable for each server control you place on the form:

```
Public Class Products_HTML
    Inherits System.Web.UI.Page
    Protected WithEvents Products As HtmlTable
    Protected WithEvents Category As HtmlSelect
```


Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

The hyphen in the name of the form is automatically converted to an underscore in the class name, because hyphens are not legal in Visual Basic class names. The control variables automatically match the ID attributes of the controls on the form. Code in the Page_Load procedure uses these variables to populate the drop-down list box and the table on the form.

The Page_Load Event Procedure

Visual Studio also automatically adds a Page_Load procedure to the class, which handles the Load event of the Page class that this class is derived from:

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
```

> TIP: As with all .NET event procedures, you can give this sub any legal name—you don't have to use Object_Event. The Page_Load name is just a convention that makes the purpose of the procedure immediately obvious and it is a naming pattern that is familiar to Visual Basic programmers because it was required in previous versions of Visual Basic, so Visual Basic .NET uses this naming pattern by default.

This procedure is very important and very useful, because it runs every time a client accesses your page, including postbacks. In this example, all the functionality of the page is defined within this procedure.

Using ADO.NET to Retrieve Data for Server Controls

In this page, the primary task of the Page_Load event is to use ADO.NET objects to retrieve data from the SQL Server Northwind database, and to use that data to populate the HTML server controls on the page.

At the very top of the Visual Basic file, before the Products_HTML class is created, we inserted three Imports statements for the ADO.NET namespaces we would be using:

```
Imports System.Data
Imports System.Data.SqlClient
Imports System.Web.UI.HtmlControls
```

These statements are there just to save some typing. They allow you to refer to ADO.NET classes using only their names, without having to prefix the names

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

with the full namespace. For example, without the Imports statement for System.Data.SqlClient, you would type:

```
Dim cmd As System.Data.SqlClient.SqlCommand
```

With the Imports statement at the top of the file, you can simplify this to:

```
Dim cmd As SqlCommand
```

To use ADO.NET, the project also needs a reference to System.Data, which you can find under the References tab in the Solution Explorer window.

Here are the ADO.NET object variable declarations that appear at the top of the Page_Load procedure. The SqlConnection declaration also includes a constructor that defines the connection:

```
Dim cnn As New SqlConnection( _
  "Data Source=(local);Initial Catalog=Northwind;" & _
  "User ID=sa;Password=")
Dim cmd As SqlCommand
Dim rdr As SqlDataReader
```

Checking the Postback Property

It is common in a page's load event procedure to distinguish between the first time the page is loaded by a user and subsequent postback loadings of the page. The Page class in ASP.NET has a Postback property you use to determine this.

In this page, you only need to populate the drop-down list once, when the form is first loaded. ASP.NET will automatically preserve the contents of the list on subsequent postback. The code checks the postback property and if it isn't true—meaning that this is a fresh hit, not a postback—the code creates an ADO.NET SqlDataReader object to retrieve a list of categories from the database:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
If Not Page.IsPostBack Then
    cmd = New SqlCommand( _
      "SELECT CategoryID, CategoryName " & _
      "FROM Categories ORDER BY CategoryName;", _
      cnn)
    cnn.Open()
    rdr = cmd.ExecuteReader( _
      CommandBehavior.CloseConnection)
```

Binding Data to an HTMLSelect List

The HTMLSelect class has a DataSource property that makes it easy to populate the list. You can also use the DataTextField and DataValueField properties to populate both the text that appears in the list and the internally stored values of the list items. After setting these properties, you must call the DataBind method to perform the action of filling the list:

```
    With Category
        .DataSource = rdr
        .DataTextField = "CategoryName"
        .DataValueField = "CategoryID"
        .DataBind()
    End With
    rdr.Close()
```

The Hidden ViewState Control

When you inspect the HTML that an ASP.NET application generates—by selecting View Source in the browser—you will find a hidden input control named __VIEWSTATE. For example, here is a portion of the HTML produced when a user first runs the Products-HTML page:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
<form name="ctrl0" method="post"
  action="Products-HTML.aspx" id="ctrl0">
<input type="hidden" name="__VIEWSTATE"
value="dDw5OTM2ODcwMTE7dDw7bDxpPDE+Oz47bDx0PDtsPGk8MT47Pjt
sPHQ8dDxwPGw8RGF0YVRleHRGaWVsZDtEYXRhVmFsdWVGaWVsZDs+O2w8Q
2F0ZWdvcnlOYW1lO0NhdGVnb3J5SUQ7Pj47dDxpPDk+O0A8QmV2ZXJhZ2V
zO0NhbmR5O0NvbmRpbWVudHM7Q29uZmVjdGlvbnM7RGFpcnkgUHJvZHVjd
HM7R3JhaW5zL0NlcmVhbHM7TWVhdC9Qb3VsdHJ5O1Byb2R1Y2U7U2VhZm9
vZDs+O0A8MTs5OzI7Mzs0OzU7Njs3Ozg7Pj47Pjs7Pjs+Pjs+Pjs+" />
```

This control is part of the HTML form, so its value is posted back to the server when the Submit button is clicked. The cryptic contents of this control are a compressed representation of the property values of all the controls that have their EnableViewState property set to True.

During a postback, ASP.NET can use the ViewState data to reconstruct the state that the page had the last time it was sent out. This allows ASP.NET to detect changes when necessary, and to recreate sections of the form that don't need to change. In this example, the Page_Load event only fills the drop-down list box once for each session, when the page is first called. During a postback, the list is automatically recreated from the ViewState data.

Populating an HTML Table

The table on this page is only populated if the page request is a postback. Presumably the user has selected a category and clicked the Show Products button.

Populating a table using the HTML Table control is not simply a matter of setting a DataSource property and binding the data. You must add rows to the table, add cells to the rows, and add content to the cells.

Here's the first part of the code, which just adds the header row at the top of the table:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
Else 'it's a postback, so populate the table
    ' Add the header row
    Dim row As HtmlTableRow
    Dim cell As HtmlTableCell

    row = New HtmlTableRow()

    cell = New HtmlTableCell("th")
    cell.InnerText = "Product ID"
    row.Cells.Add(cell)

    cell = New HtmlTableCell("th")
    cell.InnerText = "Product Name"
    row.Cells.Add(cell)

    cell = New HtmlTableCell("th")
    cell.InnerText = "Units in Stock"
    row.Cells.Add(cell)

    Products.Rows.Add(row)
```

The final step is to use ADO.NET again to retrieve a list of products for the selected category, and to add a row of cells to the table for each row in the data:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
    'Now add the data
    cmd = New SqlCommand( _
      "SELECT ProductID, ProductName, UnitsInStock " _
      & " FROM Products" _
      & " WHERE CategoryID = " & Category.Value, _
      cnn)
    cnn.Open()
    rdr = cmd.ExecuteReader( _
      CommandBehavior.CloseConnection)
    Do While rdr.Read
        row = New HtmlTableRow()
        cell = New HtmlTableCell()
        cell.InnerText = rdr("ProductID").ToString
        row.Cells.Add(cell)
        cell = New HtmlTableCell()
        cell.InnerText = rdr("ProductName").ToString
        row.Cells.Add(cell)
        cell = New HtmlTableCell()
        cell.InnerText = rdr("UnitsInStock").ToString
        row.Cells.Add(cell)
        Products.Rows.Add(row)
     Loop
     rdr.Close()
End If
End Sub
```

This code generates a standard HTML table, which you can inspect by selecting View Source in the browser after running the page.

Setting HTML Properties

Visual Studio makes it easy to set attributes of HTML controls by using the Properties window.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Try It Out!

1. Open the HTML pane of the Products-HTML.aspx Web form, and click inside the table tag, or select the **Products** table from the drop-down list at the top of the Properties window. You can also select controls in the Design pane, but the table is hard to select there because it doesn't have any rows defined.

2. In the Properties window, click in the **bgcolor** property and use the builder button (showing three dots, called an ellipsis) to open the Color Picker dialog box.

3. Click on the Named Colors tab, as shown in Figure 10, and select a color you like for the table's background.

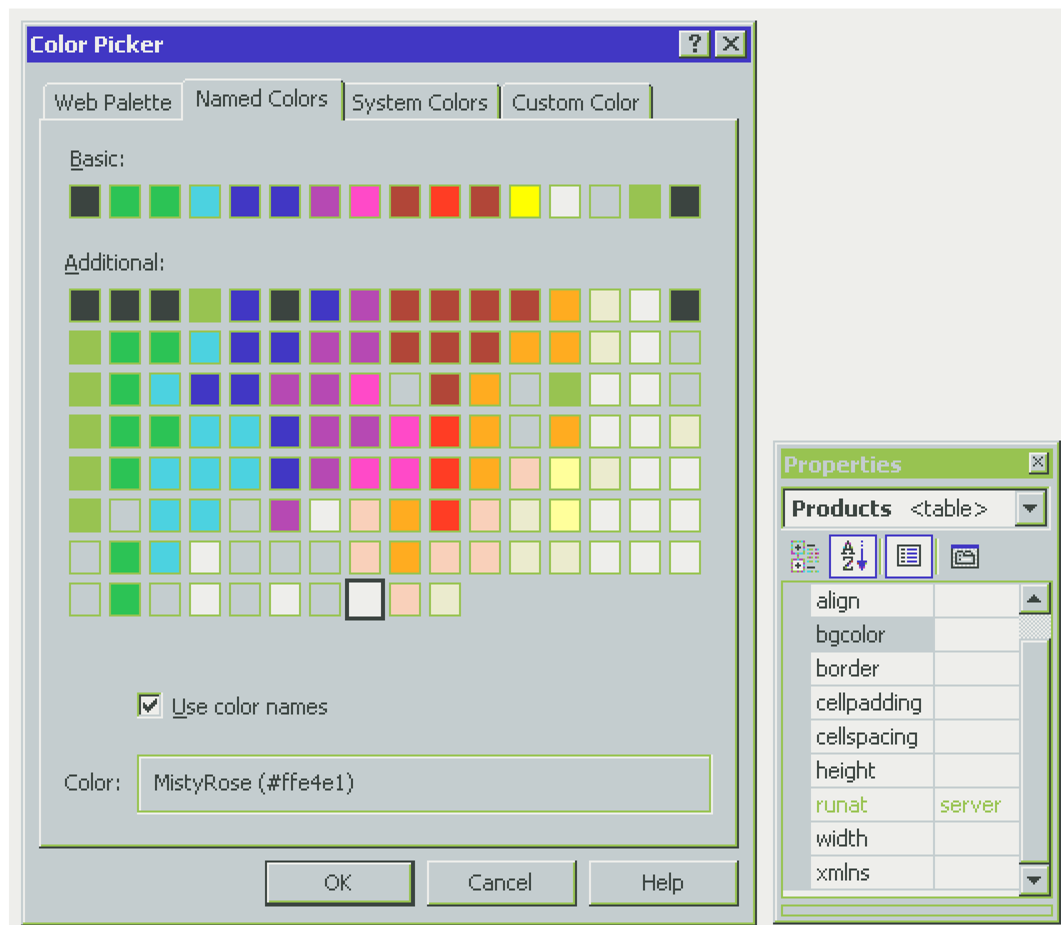

Figure 10. The ASP.NET Color Picker dialog box supports all the standard HTML options for specifying colors.

4. Notice that the setting is immediately added to the HTML attributes of the table. Test the new setting by right-clicking on the form in the Solution Explorer and selecting **Build and Browse**.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Using Web Controls

*See **Products-Web1.aspx***

The Intro project also includes a couple of example forms that use Web controls, rather than HTML controls.

Here is the HTML for the body of the Products-Web1.aspx page:

```
<body>
   <form id="Form1" method="post" runat="server">
      <asp:Label id="Label1" runat="server">
       Select a Category:</asp:Label>
      <asp:DropDownList id="cboCategory" runat="server">
      </asp:DropDownList>
      <asp:Button id="btnShow" runat="server"
       Text="Show Products"></asp:Button>
      <br>
      <asp:DataGrid id="grdProducts" runat="server"
       Visible="False" GridLines="None">
         <HeaderStyle Font-Bold="True"
          HorizontalAlign="Center"></HeaderStyle>
      </asp:DataGrid>
   </form>
</body>
```

The control tags on this page all have the prefix *asp***:**, signaling that they are Web controls. These Web controls support properties, methods, and events that go beyond what is available from ASP.NET HTML controls.

The most interesting difference between this page and the Products-HTML page is the use of a DataGrid control instead of an HTML table control.

Working with DataGrid Properties

The DataGrid Control is extremely full-featured. This page barely scratches the surface of its capabilities, and you will learn much more about using DataGrid controls later in the course.

The DataGrid, like many Web controls, contains object-based properties that can be nested several object layers deep. For example, to make the header font bold, using Visual Basic .NET, you would use syntax like this:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
grdProducts.HeaderStyle.Font.Bold = True
```

In the HTML, this complex property is represented using a HeaderStyle tag that contains a Font-Bold attribute:

```
<HeaderStyle Font-Bold="True"
```

In this example, we set the Datagrid's GridLines property to None and we made the header text bold and centered, so that the grid would look like the table in our original ASP example. You can see these settings in the HTML shown above, or in the Properties window in Visual Studio, shown in Figure 11. Notice that property values you changed appear in bold type in the Properties window.

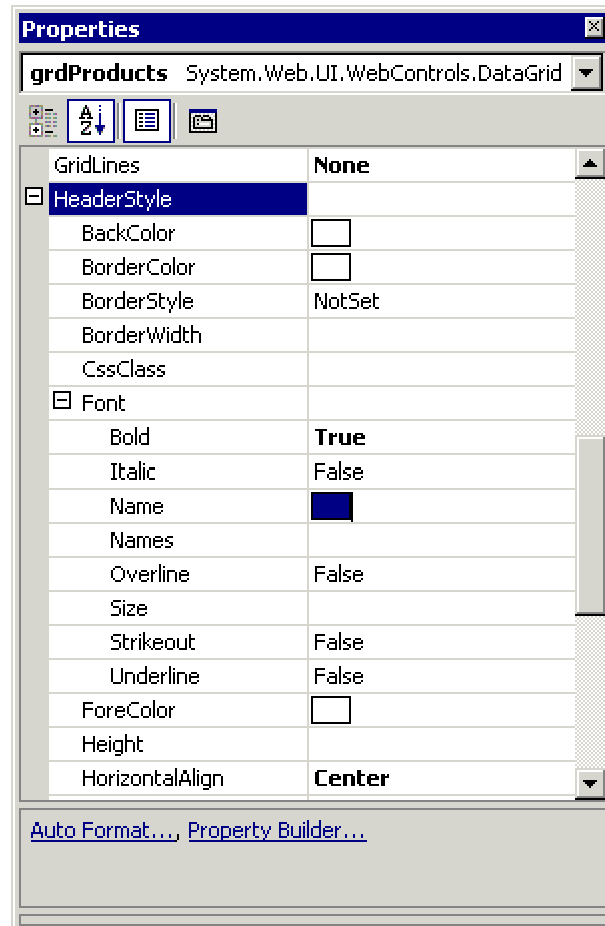

Figure 11. Properties you have changed appear in bold type in the Properties window in Visual Studio.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

In this example, we also set the Visible property of the data grid to False. We only show the grid during a postback, after the user has selected a category and clicked the button.

Binding Data to a DataGrid

*See **Products-Web1.aspx***

In contrast to the HTML table control, the DataGrid Web control is easy to fill with data. It automatically creates columns to match the columns in your data, and it creates headings that match the field names in your data.

To take advantage of this, the Products-Web1 example uses a SQL statement that gives the columns friendly names, with spaces between the words. Here is the section of code from the Page_Load event that makes the DataGrid visible and fills it with data. This code runs only when the page's Postback property is True:

```
grdProducts.Visible = True
cmd = New SqlCommand("SELECT ProductID [Product ID]," _
 & " ProductName [Product Name]," _
 & " UnitsInStock [Units in Stock] FROM Products" _
 & " WHERE CategoryID = " _
 & cboCategory.SelectedItem.Value, _
 cnn)
cnn.Open()
rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection)
grdProducts.DataSource = rdr
grdProducts.DataBind()
rdr.Close()
```

So, the same task that required a couple dozen lines of code using the HTML table control takes just a couple of lines using the DataGrid—not bad!

Using Control Events

*See **Products-Web2.aspx***

So far, the example pages have forced the user to click a button after selecting a product category. But what if you wanted to fetch the list of products immediately when the users select a category, without requiring them to click a button?

To see an example of how to accomplish this, test out Products-Web2.aspx, which is shown in a browser window in Figure 12. This page has no button—a new product listing appears automatically as soon as the user selects a category.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

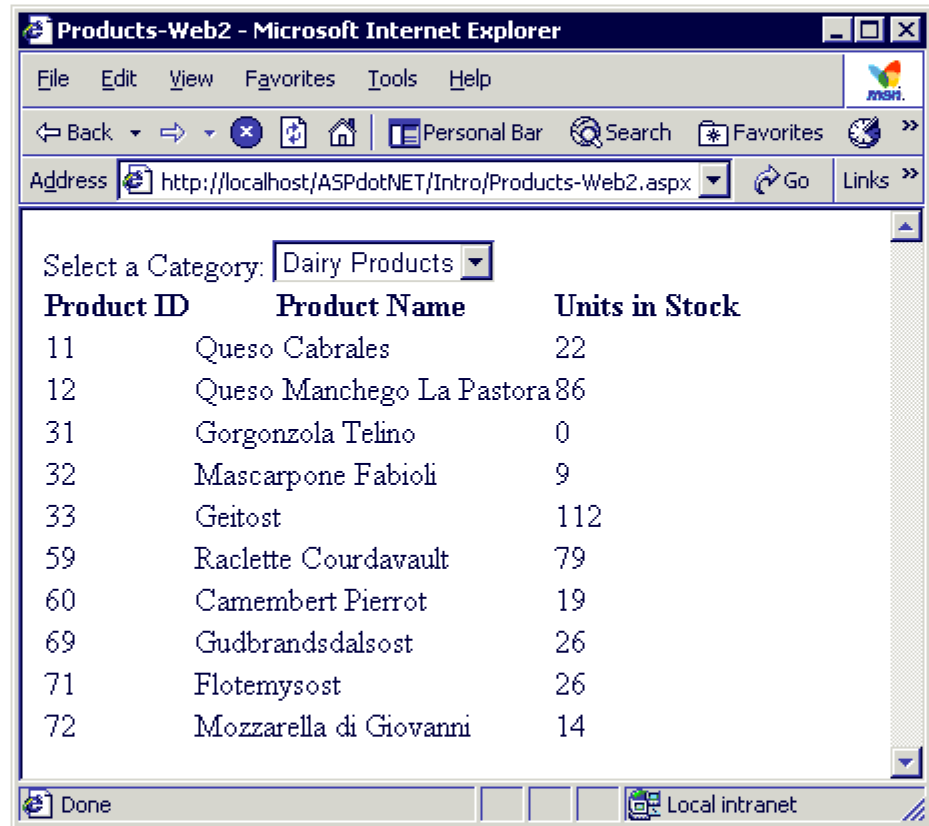

Figure 12. Using the drop-down list box to trigger a postback.

The only difference between Products-Web1 and Products-Web2 is that the button was deleted and a single property of the drop-down list control was changed—the AutoPostBack property, which is False by default, is set to True in Products-Web2, as shown in Figure 13.

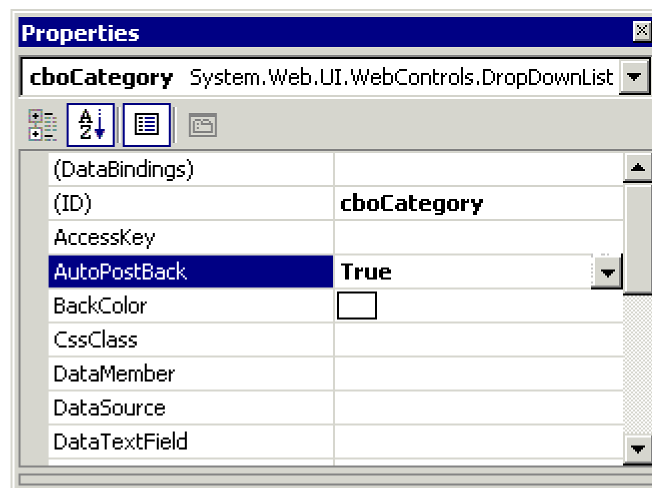

Figure 13. The AutoPostBack property causes control events to trigger a postback.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

The AutoPostBack Property

By default, data input controls do not trigger a postback. You can create server-side event procedures for these controls, but they don't run until some other event, such as a button click, causes a postback. By setting AutoPostBack to True, however, you ensure that a postback will occur when the list selection changes.

In this example, we could have moved the code that populates the data grid to an event handler for the drop-down list's SelectedIndexChanged event. But that isn't really necessary, because the Page_Load event will still run after every postback.

This behavior is something that Visual Basic programmers have a hard time getting used to. Server-side control events never run in isolation—the page's load event handler always runs first and then any control event handlers run. Events based on changes to data "accumulate" unless the control's AutoPostBack property is set to True, and then all these events run during the next postback.

The Client-Side Code that Triggers a Postback

Run **Products-Web2** and then right-click in the browser window and select **View Source**. You will see that ASP.NET has created client-side JavaScript code to trigger the postback, and a hidden input control that keeps track of which control triggered the event. A second hidden control holds any argument data from the event, which in this case is just an empty string. Here is the section of HTML that creates the drop-down list box and triggers a postback when its value changes:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
<select name="cboCategory" id="cboCategory"
 onchange="__doPostBack('cboCategory','')"
 language="javascript">
   <option value="1">Beverages</option>
   <option value="2">Condiments</option>
   (remaining list items omitted for brevity)
</select>
         <br>
<input type="hidden" name="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" value="" />
<script language="javascript">
<!--
   function __doPostBack(eventTarget, eventArgument) {
      var theform = document.Form1;
      theform.__EVENTTARGET.value = eventTarget;
      theform.__EVENTARGUMENT.value = eventArgument;
      theform.submit();
   }
// -->
</script>
</form>
   </body>
</HTML>
```


Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Using ASP.NET to Deliver XML Web Services

No introduction to ASP.NET would be complete without mentioning XML Web Services. Developers are usually first attracted to the great support ASP.NET provides for easily creating interactive Web pages, but an equally important mission of ASP.NET is providing a productive and supportive development environment for creating XML Web Services.

The Evolution of XML Web Services

Originally, the Internet was used to allow people to communicate with servers that dispensed documents in response to requests using the Hypertext Transfer Protocol (HTTP). As the Internet evolved, the servers became more sophisticated by employing technologies like ASP to perform processing in response to requests and to deliver dynamic content rather than static documents.

In the meantime, the markup language HTML, which was standardized as the page-description language for Internet documents, evolved into XML, which uses the same basic structure to represent any type of data, not just page descriptions. Using the Internet to transmit XML means that any kind of data can be communicated, taking advantage of the fact that virtually every type of computing device can now connect to the Internet.

XML Web Services are provided over the Internet to support communication between software applications. Using the HTTP and XML Internet standards, XML Web Services allow disparate types of software running on disparate platforms to send questions and answers to each other.

These communications use a standard protocol named SOAP (Simple Object Access Protocol). A more recently developed protocol that is equally important is WSDL (XML Web Services Description Language). WSDL provides the “type libraries” for XML Web Services, allowing programs to dynamically discover exactly what types of requests they can send to a particular Web Service, and what data they should expect to get back.

A third standard, UDDI (Universal Description and Discovery and Integration), allows companies to publish standardized summaries of the XML Web Services that they offer and to search among published service descriptions for the service they need.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

Support for XML Web Services in ASP.NET

ASP.NET makes it very easy for developers to take any class and turn it into a published Web Service. The developer only has to add a single attribute to the class and to each method, and ASP.NET does all the work to create the various XML files plus a Web page that describes the service and allows it to be tested.

Creating an XML Web Service in Visual Studio

To add a Web Service to an ASP.NET project in Visual Studio, you can select **Project|Add Web Service** and enter a name for your service. The service code that is created for you includes this commented-out example of how to write a simple “Hello, World” Web Service:

```
' WEB SERVICE EXAMPLE
' The HelloWorld() example service returns the string
'  Hello World.
' To build, uncomment the following lines then save and
' build the project.
' To test this web service, ensure that the .asmx file is
' the start page
' and press F5.
'
'<WebMethod()> Public Function HelloWorld() As String
'  HelloWorld = "Hello World"
' End Function
```

As this simple example code makes clear, creating a method in an XML Web Service is like writing any Visual Basic function. The only difference is that you need to preface the function with an attribute that identifies it as a WebMethod. When the compiler sees that attribute, it does all the heavy lifting for you.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

An Example Web Service

ASP.NET XML Web Services all have the extension .asmx, which allows ASP.NET to intercept and handle all requests for XML Web Services. The sample Intro project includes an XML Web Service named Products-WebService.asmx, which has one method, GetInventoryByCategory.

The GetInventoryByCategory function takes a category ID as a parameter and uses ADO.NET to query the Northwind database for all the products in that category, much as the Web pages in the project do. In this case, however, the code uses an ADO.NET DataSet object, which is automatically converted to XML by the Web Service. Here is a listing of the code:

```
<WebMethod()> Public Function GetInventoryByCategory( _
 ByVal CategoryID As Integer) As DataSet
  Dim strCnn As String
  Dim strSQL As String
  strCnn = _
   "Data Source = (local);Initial Catalog = Northwind;" _
   & "User ID=sa;Password="
  strSQL = "SELECT ProductID, ProductName, UnitsInStock" _
    & " FROM Products WHERE CategoryID =" & CategoryID
  Dim sda As New SqlDataAdapter(strSQL, strCnn)
  Dim ds As New DataSet()
  sda.Fill(ds)
  sda.Dispose()
  Return ds
End Function
```

Testing the Web Service

You can use a browser to test an XML Web Service. ASP.NET automatically creates a human-readable test page that includes a hyperlink for each of the methods supported by the service. When you click on one of those hyperlinks you get a test page for that method. Figure 14 shows the test page for the GetInventoryByCategory method. When you enter a valid category ID and click the button, the Web Service runs and returns an XML representation of the dataset returned by the method. Software running on virtually any platform can call your Web Service and process the data that is returned as XML.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

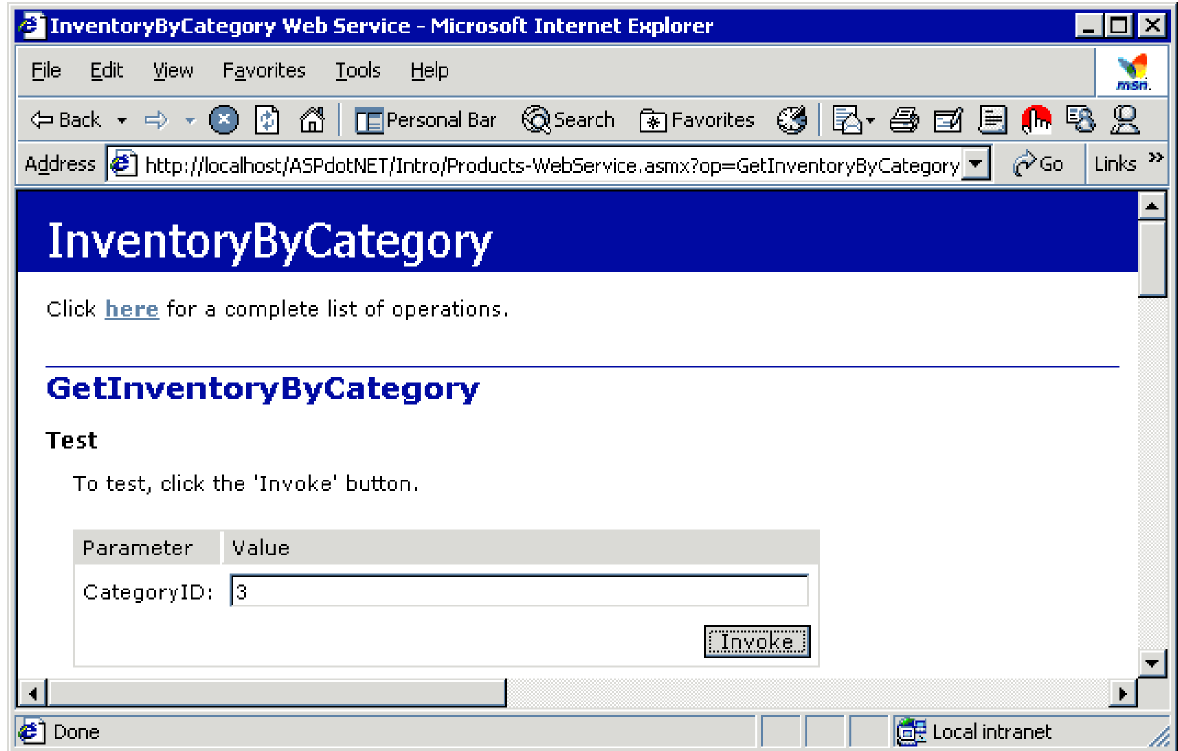

Figure 14. ASP.NET automatically creates a test page for each method in an XML Web Service.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Summary

- Classic ASP has several important shortcomings that are addressed by ASP.NET
- ASP.Net is a completely new product based on the .NET Framework.
- Server controls include the attribute runat="server".
- ASP.NET uses both HTML controls and Web Forms controls.
- The Imports statement allows you to simplify your code.
- The Postback property allows you to determine whether a page is being loaded for the first time.
- The hidden __VIEWSTATE control is used to reconstruct the state that the page had the last time it was sent out.
- Setting the AutoPostBack property determines whether data entry control events cause an immediate postback.
- ASP.NET makes it easy for developers to create XML Web Services.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

(Review questions and answers on the following pages.)

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Questions

1. Name three of ASP.NET's advantages over classic ASP.
2. What attribute do all Server controls have?
3. What does the VIEWSTATE control do?
4. Which property allows you to determine whether a page is being loaded for the first time?
5. What are XML Web Services used for?

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

Answers

1. Name three of ASP.NET's advantages over classic ASP.
   Compiled code, separation of code from HTML, better design tools, state management that works in Web farms, easier configuration based on XML files, automatic support for different browsers

2. What attribute do all Server controls have?
   runat="server"

3. What does the VIEWSTATE control do?
   It is used to reconstruct the state that the page had the last time it was sent out.

4. Which property allows you to determine whether a page is being loaded for the first time?
   The Postback property

5. What are XML Web Services used for?
   To support communication between software applications.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Lab 1: Introduction to ASP.NET

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Lab 1 Overview

In this lab you'll learn how to work with HTML and Web server controls in Visual Studio.

To complete this lab, you'll need to work through two exercises:

- Working with HTML Controls
- Working with Web Controls

Each exercise includes an "Objective" section that describes the purpose of the exercise. You are encouraged to try to complete the exercise from the information given in the Objective section. If you require more information to complete the exercise, the Objective section is followed by detailed step-by-step instructions.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

Working with HTML Controls

Objective

In this exercise, you'll create an HTML Label and Table control. You'll use the HTML editor to set properties for the Table control. You'll then write code to populate the Table control with the Country and ContactName data from the Northwind Customers table.

Things to Consider

- How do you create HTML controls in Visual Studio?
- How do you set properties for the controls?
- How do you access data from a database?
- How do you fill an HTML table?

Step-by-Step Instructions

1. Open the **HTMLControls.aspx** page and click the **Toolbox** button on the toolbar. Choose **Project|Show All Files** if that option is not already selected. These steps assume that the pageLayout property is set to GridLayout.

2. Click the **HTML** section on the Toolbox to display the HTML controls.

3. Double-click the **Label** control to add it to the page. Make sure that the label is not selected and double-click the **Table** control to add it to the page also. Reposition the table, if necessary by clicking and dragging. The page should look similar to Figure 15.

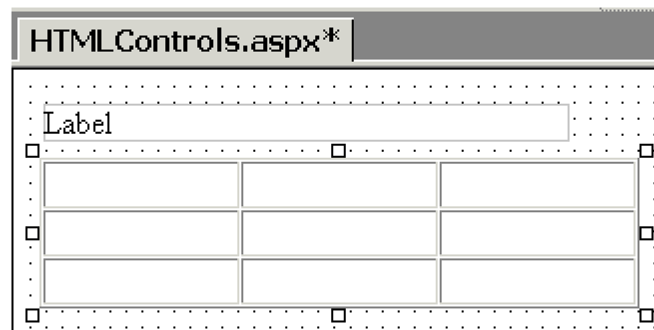

Figure 15. A Label and a Table HTML control.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

4. Click the **HTML** tab on the bottom of the page and examine the HTML that was created automatically when you dropped the controls on the page.

5. Select all of the **<TD> </TD>** and **<TR> </TR>** pairs, and delete them. Click the Design tab—the cells are no longer visible. However, the table is still there and you can continue working with it in the HTML window.

6. To ensure that you can access your table in event procedures, check that the opening table tag includes `runat="server"`. Enter this attribute, if necessary:

```
<TABLE runat="server"
```

7. In the HTML pane position your cursor just after the **<TABLE** tag. Right-click and choose **Properties** from the menu. This loads the Property Pages dialog box.

8. Click the builder button (**…**) next to the **Background color** option and choose **Named Colors**. Select a color from the list—Light Blue, for example, and click **OK**.

   Note the HTML that was created:

```
<TABLE runat="server" style="Z-INDEX: 102; LEFT: 13px;
 POSITION: absolute; TOP: 44px" cellSpacing="1"
 cellPadding="1" width="300" border="1"
 bgcolor=LightBlue>
```

9. Modify the <TABLE to give it an id property of "Customers" by typing the bolded text in the tag:

```
<TABLE id="Customers" style="Z-INDEX: 102; LEFT: 13px;
 POSITION: absolute; TOP: 44px" cellSpacing="1"
 cellPadding="1" width="300" border="1"
 bgcolor=LightBlue>
```

10. Change the text that reads **Label** (located right on top of the </DIV> tag) to **Customer Contacts**.

11. Next, open the code-behind file, **HTMLControls.aspx.vb**. Add the following Imports statements to save yourself from having to type out the fully qualified names of classes:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

```
Imports System.Data.SqlClient
Imports System.Web.UI.HTMLControls

Public Class HTMLControls
  Inherits System.Web.UI.Page
  Protected WithEvents Customers As HtmlTable
```

12. Now it's time to write code that loads the table with data. Replace the existing Page_Load event handler stub with the following procedure. You can copy and paste this code from HTMLControls.txt:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

```
Private Sub Page_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles MyBase.Load
  Dim cnn As New SqlConnection("Data Source=(local);" _
   & " Initial Catalog=Northwind;User ID=sa;Password=")
  Dim cmd As SqlCommand
  Dim rdr As SqlDataReader
  Dim row As HtmlTableRow
  Dim cell As HtmlTableCell

  row = New HtmlTableRow()

  cell = New HtmlTableCell("th")
  cell.InnerText = "Country"
  row.Cells.Add(cell)

  cell = New HtmlTableCell("th")
  cell.InnerText = "Contact Name"
  row.Cells.Add(cell)

  Customers.Rows.Add(row)

  'Now add the data
  cmd = New SqlCommand( _
    "SELECT Country, ContactName FROM Customers" _
    & "ORDER BY Country", cnn)
  cnn.Open()
  rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection)

  Do While rdr.Read
    row = New HtmlTableRow()

    cell = New HtmlTableCell()
    cell.InnerText = rdr("Country").ToString
    row.Cells.Add(cell)
```


Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
      cell = New HtmlTableCell()
      cell.InnerText = rdr("ContactName").ToString
      row.Cells.Add(cell)

      Customers.Rows.Add(row)
    Loop
    rdr.Close()
  End Sub
```

13. Click the **Save** button to save your changes. Select the **HTMLControls.aspx** page in the Solution Explorer, right-click and choose **Build and Browse**. The page should be displayed in your browser.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Working with Web Controls

Objective

In this exercise, you'll create a page using Web controls rather than HTML controls, specifically a Label and a DataGrid. You'll use the Properties window to set properties for the controls. You'll then write code to populate the DataGrid control with Country and ContactName data from the Northwind Customers table, and to modify the caption of the label.

Things to Consider

- How do you create Web controls in Visual Studio?
- How do you set properties for the controls?
- How do you fill a DataGrid contol with data?
- How do Web controls differ from HTML controls.

Step-by-Step Instructions

1. Open WebControls.aspx and click on the **Toolbox** icon to load the Toolbox. Click the **Web Forms** bar in the toolbox so that you can select Web controls.

2. Add a Label control and DataGrid control so that the page looks like Figure 16.

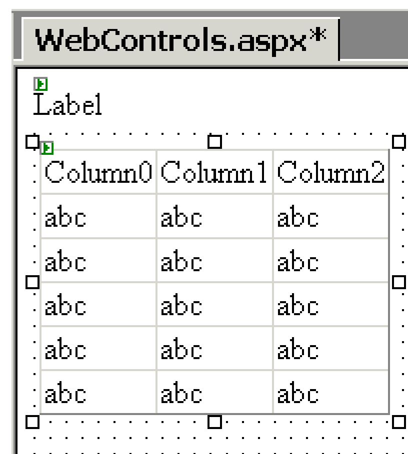

Figure 16. Add a Label and DataGrid control.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

3. Click the **HTML** tab to examine the HTML created. Note the asp: prefix for the controls.

4. Next, open the WebControls.aspx.vb page and place the following Import statements at the top of the page. Note how the WebControls namespace allows you to shorten the class names for the label and data grid:

```
Imports System.Data.SqlClient
Imports System.Web.UI.WebControls

Public Class WebControls
  Inherits System.Web.UI.Page
  Protected WithEvents Label1 As Label
  Protected WithEvents DataGrid1 As DataGrid
```

5. Next, replace the Page_Load event handler with the following procedure, which populates the DataGrid control and sets the Label control's Text property. You can copy this code from WebControls.txt:

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

```
Private Sub Page_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles MyBase.Load
  Dim cnn As New SqlConnection( _
    "Data Source=(local);Initial Catalog=" _
     & "Northwind;User ID=sa;Password=;")
  Dim cmd As SqlCommand
  Dim rdr As SqlDataReader

  DataGrid1.Visible = True
  cmd = New SqlCommand( _
   "SELECT Country, ContactName FROM Customers" _
   & " ORDER BY Country", cnn)
  cnn.Open()
  rdr = _
   cmd.ExecuteReader(CommandBehavior.CloseConnection)

  DataGrid1.DataSource = rdr
  DataGrid1.DataBind()

  rdr.Close()
  Label1.Text = "Customers by Country"
End Sub
```

6. Save your changes by clicking the **Save** button. Right-click on the **WebControls.aspx** file in the Solution Explorer and choose **Build and Browse**. The Label and DataGrid should be populated with data and displayed in the browser.

7. Close the browser window and return to the Design tab of the WebControls.aspx page.

8. Select the Label control and press **F4** to bring up the Properties window.

9. Expand the plus sign (+) next to the Font property. Set the **Name** to **Verdana**, **Bold** to **True**, and the **Size** to **Medium**. Widen the label so that it can accommodate the larger text size when displayed in a browser.

10. With the Properties window still open, click on the **DataGrid** control. Set the Font **Name** to **Verdana**.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.
AppDev

11. Expand the plus sign (+) next to the **HeaderStyle** option. Set the **BackColor** property to **Aqua**.

12. Click the **HTML** tab and examine the changes that have been made by changing properties in the designer.

13. Save your changes by clicking the **Save** button. Right-click on the **WebControls.aspx** file in the Solution Explorer and choose **Build and Browse**.

Your changes will be displayed in the browser. For the most part, you'll find it easier to work with Web controls than HTML controls—you can do more and there is usually less code to write.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.

Copyright © by Application Developers Training Company and AppDev Products Company, LLC
All rights reserved. Reproduction is strictly prohibited.