CHAPTER 1

ASP.NET Language Structure

IN THIS CHAPTER:
Page Structure
Defining Your Own Procedures
Page
Compiler Directives
Quiz Sample Page
ASP.NET presents a whole new approach to developing dynamic content for the Internet or your intranet. With ASP, the process was very linear. A page was requested, and your ASP code returned HTML tags either directly or indirectly through the Response object. You then received the visitor’s input through the Request object.

With ASP.NET, the approach is much more object driven. The process is more like developing a standard Windows-based application instead of a Web page. With ASP.NET, you place controls such as Label controls and TextBox controls on your page. You then assign values to the properties of these controls, which allows you to affect how the controls are rendered in the visitor’s browser.

Your controls have methods that allow you to take some action against them. You know that the visitor has taken some action because events fire, which you can write code for.

In this chapter, we will look at the structure of a very simple ASP.NET page and how that page is rendered in the visitor’s browser. Then we will look at the Page class, which is the .NET class that all your ASP.NET pages are based on. After that, we will review basic compiler directives that you can place on your page. At the end of the chapter, we will review a simple Quiz ASP.NET page that demonstrates the basics of an ASP.NET page.

Page Structure

In this section of the chapter, we will review the structure of ASP.NET through a simple ASP.NET page. The page simply displays some text to the visitor through their browser. Figure 1-1 displays the output of this page.

HelloWeb.aspx

Here is the entire code for this simple ASP.NET page:

```xml
<%@ Page Language=VB Debug=true %>
<script runat=server>
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    lblMessage.Text = "Hello Web!"
End Sub
</script>
<html>
<head>
<title>Hello Web Sample Page</title>
</head>
```
First, notice the general structure of the contents of the ASP.NET page. Like an ASP page, the page does contain standard HTML tags. But notice that the controls placed on this page are defined like some specialized HTML tags. This is very different from ASP, where you would have found Response.Write statements.

Also notice that all the code on the page is contained within a procedure. In ASP, you could see code anywhere and the code was simply contained within code tags.

The first line on the page is called a compiler directive. This directive tells the compiler that the language you are using on this page is Visual Basic.NET. It also tells the compiler that you want the page to be in debug mode. The following directive will be discussed further later in this chapter:

```html
<%@ Page Language=VB Debug=true %>
```
Next, we define a script block. Notice that we indicate that the code should run on the server side, unlike a script block that you want to have passed through to the client and run on their browser. Typically, you will find all the code for the page within a single code block at the top of the page. But you could place code in many different blocks throughout the page. Or, as you will see in Chapter 16, code can even be placed in a separate file:

```xml
<script runat=server>
```

Within the opening and closing script tags, all the code is contained within procedures. Here, a sub procedure is defined as opposed to a function. A sub does not return a value, whereas a function does. In this case, the procedure is called Page_Load. This special procedure runs every time the page is loaded, so many of your pages will contain this procedure:

```xml
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
```
Within the procedure, we assign a value to the property of one of the controls defined on our ASP.NET page. In this case, the control is a Label control called lblMessage and the property we are setting is the Text property. This control will be discussed further in Chapter 3, but take note of the fact that we can assign values to the properties of our controls in code or in a control’s definition:

```csharp
lblMessage.Text = "Hello Web!"
```

We then close the sub’s definition

```csharp
End Sub
```

and close the script tag.

```html
</script>
```

Next, notice that, like an ASP page, our ASP.NET page contains standard HTML elements. Here, we open the HTML page and define its Title and Body properties:

```html
<html>
<head>
<title>Hello Web Sample Page</title>
</head>
<body
    background="./bg.gif"
    text="black"
    link="darkred"
    vlink="darkred"
    alink="red"
    leftmargin="40"
    topmargin="30"
>
```

After that, you will find that most of your ASP.NET pages contain a special Form tag. The tag is different from a standard HTML Form tag in that it contains the runat="server" parameter. This tells the .NET compiler that you wish to maintain the state of the controls on this page. That means that if you had a control on the page that the visitor supplied some input into, such as a TextBox control, its value and the values of the properties of the Label controls on this page would be maintained between calls to this page:

```html
<form runat="server">
```
Next, we use a standard HTML Font tag to indicate the font face that should be used on this page. This is another way that ASP.NET interacts with HTML tags. If you don’t supply a font name for your controls, they will inherit whatever font is used on the rest of the page outside the control’s definition:

```html
<font face="Tahoma">
```

Next, we define the first of the controls on this page. Notice how a control is defined. We start with an opening tag character “<” that is followed by the name of the library that the control is from and the name of the control type itself, in the form library:control:

```html
<asp:Label
```

That is followed by the ID property. The ID property stores the name of the control. You don’t have to give all your controls a name, but this is how you will refer to the control in code. If you wish to refer to the control in code, you need to give it a name:

```html
id="lblTitle"
```

Next, we assign other properties for the control. The order that we define the properties is irrelevant. You should probably pick an order and be consistent with it to make your code easier to read. As you saw in Figure 1-1, the first Label control was defined with a border. Here, we supply the values for that border:

```html
BorderWidth="7px"
BorderStyle=9
```

We also indicate the width of the control on the page relative to the width of the page in the visitor’s browser:

```html
Width="90%"
```

Next, we indicate the font type and size:

```html
Font-Size="25pt"
Font-Name="Arial"
```

After that, we supply the text that should appear in the Label control:

```html
Text="Test Page"
```
We also indicate that the server should run the control. We must add this property to all of our control definitions:

```html
runat="server"
```

After that, we close the control’s definition:

```html
/>
```

Next, we have more HTML tags:

```html
<br><br><br>
```

Then we define the other Label control. Note the name of this Label control, indicated in the ID property. This control is the one that we referred to in the Page_Load procedure:

```html
<asp:Label
   id="lblMessage"
   runat="Server"
   Font-Bold="True"
/>
```

After that, we end the page by closing the HTML tags and the ASP.NET Form tag:

```html
</font>
</form>
</body>
</html>
```

When the visitor requests this page, the .NET compiler runs the page and returns HTML only to the visitor. None of the code or controls defined and indicated with the runat="Server" property are returned in that form to the visitor’s browser. Instead, the browser would receive this HTML only:

```html
<html>
<head>
<title>Hello Web Sample Page</title>
</head>
<body
   background="./bg.gif"
   text="black"
   link="darkred"
```
At the top of the page, the opening standard HTML tags were untouched by the compiler:

```html
<html>
<head>
<title>Hello Web Sample Page</title>
</head>
<body>
<form name="ctrl0" method="post" action="helloweb.aspx" id="ctrl0">
<input type="hidden" name="__VIEWSTATE" value="dDwxNzIzNzUyNTg1O3Q8O2w8aTwxPjs+O2w8dDw7bDxpPDQ+Oz47bDx0PHA8cDxsPFRleHQ7PjtsPEhlbGxvIFd1YiE7Pj47Pjs7PjzPjs+Pjs+Pjs+" />
<font face="Tahoma">
<span id="lblTitle" style="border-width:7px;border-style:Outset; font-family:Arial;font-size:25pt;width:90%;">Test Page</span>
<br><br><br>
<span id="lblMessage" style="font-weight:bold;">Hello Web!</span>
</font>
</form>
</body>
</html>
```
But notice that the form tag has changed. The runat property is gone and the compiler added some properties of its own, which indicate the action to take if the page was submitted back by the visitor:

```html
<form
    name="ctrl0"
    method="post"
    action="helloweb.aspx"
    id="ctrl0"
>

Next, an Input control is defined that was not at all part of our page’s code. This control is ASP.NET’s way of maintaining the state of the call of this page by this visitor. This unique value will be used by ASP.NET if the page is resubmitted so that the values of variables, controls, and properties can be retrieved:

```html
<input
type="hidden"
    name="__VIEWSTATE"
    value="dDwxNzIzNzUyNTg1O3Q8O2w8aTwxPjs+O2w8dDw7bDxpPDQ+Oz47bDx0PHA8cDxsFPR1eHQ7PjtsPEhlbGvIFd1YiE7Pj47Pjs7Pjs+Pjs+Pjs+"
/>
```

Next, we have the unchanged HTML Font tag:

```html
<font face="Tahoma">```

Next, we have an HTML Span tag. But we didn’t define a Span tag. What the compiler did was to take our first Label control and render it through the properties
of an HTML Span tag. Notice that the Style property in the tag contains many of the values that we set for the control when it was defined on the ASP.NET page:

```html
<span id="lblTitle"
     style="border-width:7px;border-style:Outset;
             font-family:Arial;font-size:25pt;width:90%;">
     Test Page
</span>
```

Next comes more HTML, just as we placed it on the ASP.NET page:

```html
<br><br><br>
```

That is followed by another Span tag rendered through our second Label control,

```html
<span id="lblMessage"
     style="font-weight:bold;">
     Hello Web!
</span>
```

And we end the output page with the closing HTML tags:

```html
</font>
</form>
</body>
</html>
```

### Defining Your Own Procedures

OddOrEven.aspx

In the past section, you saw how to define a procedure that runs when a page is loaded. In this section, through a sample page, we will look at how you can create your own procedure that you then call from one of the event procedures. The output of the sample page is displayed in Figure 1-2.
The page allows the visitor to enter a number. When they click OK, they see whether the number they enter is an odd or even number.

As with the sample page we looked at in the last section, this sample page has a compiler directive at the top that tells the compiler the language to use and that we are in debug mode:

```vbscript
<%@ Page Language=VB Debug=true %>
```

Within the body of the page, all the controls on the page are placed within the ASP.NET Form tag:

```html
<form runat="server">
```
Those controls include a Label control for the title of the page:

```html
<asp:Label
    id="lblTitle"
    BorderWidth="7px"
    BorderStyle=9
    Width="90%"
    Font-Size="25pt"
    Font-Name="Arial"
    runat="server"
/>
```

That is followed by another Label control that displays the page’s instructions:

```html
<asp:Label
    id="lblMessage1"
    runat="server"
    Font-Bold="True"
    Text="Enter a number"
/>
```

Next, we define a TextBox control. This control allows the visitor to enter a number. This control and the next are discussed further in Chapter 3:

```html
<asp:TextBox
    id="txtNumber"
    runat="server"
/>
```

After the TextBox control is the Button control. This control allows the visitor to submit the number that they enter. When they take that action, the procedure indicated in the OnClick property will run:

```html
<asp:button
    id="butOK"
    text="OK"
    Type="Submit"
    OnClick="SubmitBtn_Click"
    runat="server"
/>
```
One other control is defined on this page. That control is another Label control and is used to display the message whether the number is odd or even:

```html
<asp:Label
    id="lblMessage2"
    runat="server"
    Font-Bold="True"
/>
```

The code on this page is defined at the top within Script tags and is defined within three procedures:

```html
<script runat=server>
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    lblTitle.Text = "Odd or Even Page"
End Sub
Sub SubmitBtn_Click(Sender As Object, E As EventArgs)
    lblMessage2.Text = "The number you entered is " & OddOrEven(txtNumber.Text)
End Sub
Function OddOrEven(TheNumber as Long) as String
    If TheNumber Mod 2 = 0 Then
        OddOrEven = "even."
    Else
        OddOrEven = "odd."
    End If
End Function
</script>
```

The first procedure is called Page_Load. Therefore, it will run every time the page is loaded:

```html
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    lblTitle.Text = "Odd or Even Page"
End Sub
```

It simply places text in the Text property of the title Label control:
The next procedure matches the name of the procedure that we specified in the OnClick property of the Button control. Therefore, this code will run when that Button control is clicked:

Sub SubmitBtn_Click(Sender As Object, E As EventArgs)

    The procedure places text into the bottom Label control corresponding to the number passed in. Notice that it calls our own procedure that we will define on this page:

    lblMessage2.Text = "The number you entered is " & OddOrEven(txtNumber.Text)

End Sub

That procedure is defined next. It is a function, which means that it will return a value. The value will be returned in the form of a String data type. The function is defined to have one parameter passed into it. The parameter must be a number of type Long:

Function OddOrEven(TheNumber as Long) as String

    Within our procedure, we check to see whether the number is even by seeing if it is evenly divisible by the number 2:

    If TheNumber Mod 2 = 0 Then

        If it is, the number is even and we return this text from our procedure:

        OddOrEven = "even."

    Else

        We return this text from our procedure:

        OddOrEven = "odd."

    End If

End Function
Typically, you would want to create your own procedure when you found that you were using the same code over and over again throughout your page. You could optionally define the procedure in a separate file so that it is available to more than one page. That topic is discussed in Chapter 16.

Page

Every ASP.NET page you create inherits the Page class. That means that the structure of your ASP.NET page is based on the structure of an object called Page. That Page object contains its own properties, methods, and events that control how the page is rendered. Some of the properties, methods, and events are important enough that you will use them on most of your pages. In this section, we will discuss those items.

Page_Load Event

Every time your page loads, either initially when the visitor browses to it or when the page is called additionally through a button being clicked or because of some other event, the code in the Page_Load event fires.

The structure of the Page_Load event must match this:

```
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    'Code goes here.
End Sub
```

The procedure is defined as a sub with the name Page_Load. The procedure must be defined with two parameters, the first of type Object and the second of type EventArgs.

Within the procedure, you would typically place initialization code. For example, maybe you have a DropDownList control that needs to display the names of sections at your site. Maybe that list comes from a database table. You would place code in this event procedure that would populate the DropDownList control from the database. Or you could place code in this event that makes sure the visitor has entered the site first by logging in. Your code would check to see whether they have, and if they haven’t, you would send them to do so.
Page_UnLoad Event

When the compiler is totally done with running your page, you have one last place to put clean-up code or any terminating code, and that is in the Page_UnLoad event. This event fires at the end of every rendering of your page. A procedure defined to run in this event must have this structure:

```
Sub Page_UnLoad(ByVal Sender as Object, ByVal E as EventArgs)
    'Code goes here.
End Sub
```

The procedure is defined as a sub and contains two parameters. The first is of type Object, and the second is of type EventArgs. You can’t write any output to the visitor’s browser when this event procedure fires. But you can place code that closes database connections or that records somewhere that the page completed successfully.

IsPostBack Property

As already explained, the Page_Load event fires every time a page is loaded. But sometimes you want code to run only the first time a page is loaded. For example, if you were to populate a DropDownList when a page was loaded with the names of the sections of your site, you would likely need to do that only the first time the page was loaded. If the visitor clicked a Button control on the page, which caused the page to load again, you would not want this code to run a second time.

You can check to see whether this is the initial loading of the page by querying the IsPostBack property of the Page object. The property returns False the first time the page is loaded:

```
If IsPostBack = False Then
```

If the page has been called because one of the events on the page has fired, for instance, because a button has been clicked, the property returns True.

IsValid Property

Another important property of the Page object is the IsValid property. This property is used in conjunction with validation controls. Validation controls, discussed in Chapter 5, allow you to make sure that the input entered by the visitor falls within your requirements. For example, a quantity field needs to be a number.
The IsValid property indicates to you whether all the controls on the page meet your input requirements. The property returns True if IsValid = True if all the controls on the page are valid. Otherwise, the property returns False.

Compiler Directives

*Compiler directives* are special tags that you add to your ASP.NET page that indicate how the page should run or be compiled. In this section, we will review some of those directives that are pertinent to the page as a whole.

**@Page Directive**

You have already seen the @Page directive used in the two samples cited in the previous sections like this:

```csharp
<%@ Page
    Language=VB
    Debug=true
%>
```

The directive specifies instructions to the compiler. Two parameters are frequently used. The first is Language:

```csharp
Language=VB
```

Here, it is set to VB, but it could be set to C# or any other .NET language. Next, we set the Debug parameter to True:

```csharp
Debug=true
```

The default for this parameter is False. If the parameter is set to True, the page will be compiled with debug symbols. This means that when an error occurs on the page, you often will get a better idea of what went wrong. But note that setting this parameter to True hurts performance. Therefore, it is great to use for development but should be changed in deployment.

Another parameter you can set is Buffer:

```csharp
Buffer=False
```
The default is True. When the parameter is set to True, the response is buffered; that is, the response is held until complete. If the property is set to False, buffering does not occur.

The Description parameter

Description="This is my test page."

is ignored by the compiler. The parameter can be set to any string you like, but it typically is set to a description of the page.

Two compiler options you can set are Option Explicit and Option Strict:

Explicit=True
Strict=True

Option Explicit controls whether variable declaration is required. If you set the option to True, you must declare all your variables; otherwise, you do not have to declare your variables before using them.

Option Strict controls whether you allow variables to automatically be typed to different types as needed. If you set this parameter to False, variable types are not strict. If you set the parameter to True, variable typing is strict.

@Import Directive

The @Import Directive allows you to include additional libraries or namespaces in your page so that you can instantiate controls or objects from that namespace. You will frequently see this directive used like this on pages that access data from a database:

<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

In this code, two namespaces are imported. The first is the System.Data namespace. As you will see in Chapters 10 and 11, this namespace enables you to access data that is returned from a database. The Other Import directive enables us to instantiate objects that allow us to connect to and manipulate data in a SQL Server database.

@OutputCache Directive

Say that you have an ASP.NET page that displays the weather for your company’s location. And say that the weather information comes from a database and that the weather data is updated in that database only every hour. That means that your page will run over and over again displaying the exact same data for an hour. You really
only need your page to run once every hour and then show that run of the page until
the hour has been reached. The @OutputCache allows you to do that:

```html
<%@ OutputCache Duration=3600 VaryByParam="none" %>
```

Here, we set the cache duration to 3600 seconds, or one hour. Notice the inclusion
of the VaryByParam parameter and that it is set to none. This indicates that we want
this page to be cached in all conditions.

### Quiz Sample Page

**Quiz.aspx**

In this section of the chapter, a more complex ASP.NET page is presented. The page
allows the visitor to take a three-question ASP.NET quiz. When the visitor first enters
the page, they see the Quiz that is displayed in Figure 1-3.

The visitor selects the answer that they think is correct for each question. They
then click OK and see the results of their quiz displayed, as in Figure 1-4.
The Results view is actually the same ASP.NET page as the Quiz page. Code is used to toggle the controls that the visitor sees on the page. On the Results view, the visitor sees the number of questions that they answered correctly.

The page uses Panel, Label, DropDownList, and Button controls to accomplish its functionality. Also defined on the page are two procedures that fire on different events.

The first control defined on the page is a Label control:

```<asp:Label
    id="lblTitle"
    BorderWidth="7px"
    BorderStyle=9
    Width="90%"
    Font-Size="25pt"
    Font-Name="Arial"
    runat="server"
/>```
This control will display the title of the page. After that, a Panel control is defined:

```xml
<asp:Panel
  id="Panel1"
  runat="server"
>

A Panel control is simply an element for you to place other controls on. The reason for doing this is that it makes it much easier to hide and show groups of controls. On this page, there are two Panel controls. One displays the questions, and the other, the quiz results. In code, we toggle which controls are visible by setting the Visible property of these panels.

The first control on this first Panel control is a Label control that displays the first question:

```xml
<asp:Label
  id="lblQuestion1"
  runat="server"
  Font-Bold="True"
  Text="What is the unit of measure in the Duration parameter of the OutputCache directive?"
/>
```

Next, we define a DropDownList control. A DropDownList control is rendered as an HTML Select form element with Option tags. Using this control, the visitor will select the answer that they think is the correct answer for the first question:

```xml
<asp:DropDownList
  id="DDLAnswer1"
  runat=server>
  <asp:ListItem>Hour</asp:ListItem>
  <asp:ListItem>Minute</asp:ListItem>
  <asp:ListItem>Second</asp:ListItem>
</asp:DropDownList>
```

The DropDownList control will be discussed in Chapter 4.

Next, we define another Label and DropDownList control for the second question:

```xml
<asp:Label
  id="lblQuestion2"
  runat="server"
>
Setting Option Explicit to True does not require variable declaration.

We also define another pair of controls for the third question:

That is followed by a Button control, which the visitor clicks to have their quiz scored:

We then end the definition of controls on this Panel control
and define the second Panel control:

```csharp
<asp:Panel
    id="Panel2"
    runat="server"
>
    This Panel control contains just a single Label control that is used to display the visitor's quiz score:

    <asp:Label
        id="lblResult"
        runat="server"
        Font-Bold="True"
    />
</asp:Panel>
```

The first procedure on this page fires whenever the page loads, since it is called Page_Load:

```csharp
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    If Not IsPostBack Then
        Panel1.Visible = True
        Panel2.Visible = False
        lblTitle.Text = "Quiz Page"
    End If
End Sub
```

First, we check to see whether this is the initial view of this page by checking the IsPostBack property of the Page object:

```csharp
If Not IsPostBack Then
    If the code flows into this If statement, that means that this is the initial viewing of the page. In that case, we show the Quiz panel

    Panel1.Visible = True
```

and hide the Results panel:

```csharp
Panel2.Visible = False
```
We also place this text into the title of the page:

```vbnet
lblTitle.Text = "Quiz Page"
End If
```

The other procedure on this page fires when the Button control is clicked. This procedure scores the quiz:

```vbnet
Sub SubmitBtn_Click(Sender As Object, E As EventArgs)
    Dim TotalCorrect as Integer
    Panel1.Visible = False
    Panel2.Visible = True
    lblTitle.Text = "Results Page"
    If DDLAnswer1.SelectedItem.Text = "Second" Then
        TotalCorrect = TotalCorrect + 1
    End If
    If DDLAnswer2.SelectedItem.Text = "False" Then
        TotalCorrect = TotalCorrect + 1
    End If
    If DDLAnswer3.SelectedItem.Text = "False" Then
        TotalCorrect = TotalCorrect + 1
    End If
    lblResult.Text = "You scored " & TotalCorrect " out 3 correct."
End Sub
```

Within this procedure, we declare a variable that will store the total number of questions that the visitor answered correctly:

```vbnet
Dim TotalCorrect as Integer
```

Since we will be displaying the quiz results, we hide the quiz

```vbnet
Panel1.Visible = False
```

and display the results:

```vbnet
Panel2.Visible = True
```

We also need to display different text in the title Label control:

```vbnet
lblTitle.Text = "Results Page"
```
We then check to see whether the visitor answered the first question correctly:

```vbnet
If DDLAnswer1.SelectedItem.Text = "Second" Then

If they did, we increment the total correct counter:

TotalCorrect = TotalCorrect + 1

Do the same for the second question

If DDLAnswer2.SelectedItem.Text = "False" Then
    TotalCorrect = TotalCorrect + 1
End If

and the third question

If DDLAnswer3.SelectedItem.Text = "False" Then
    TotalCorrect = TotalCorrect + 1
End If

Then display the number of correct answers back to the visitor through the Label control:

lblResult.Text = "You scored " & TotalCorrect & " out 3 correct."
```

As you can see, ASP.NET makes it relatively easy to create dynamic Web pages. In the next few chapters, we will look more deeply at the basic controls that you will use on numerous pages to create extensive dynamic content.