CHAPTER 1

Basic ASP.NET Page Issues

TIPS IN THIS CHAPTER

When you create an ASP.NET page it can contain certain items and those items need to be defined in an expected way. To get the most out of your ASP.NET development work, you need to be familiar with this structure. This chapter presents tips and techniques that show you how to effectively define an ASP.NET page. In addition, tips and techniques are presented with regards to the Page object, which is the underlying class that all ASP.NET pages are based on.

Coding the Basic ASP.NET Page Structure

When you create an ASP.NET page, that page needs to include certain tags and have an expected structure to it. This tip defines the basic structure of an ASP.NET page.

USE IT The page created for this tip contains the basic structure of an ASP.NET page. At the top of that page, you will find these directives:

```
<%@ Page Language=VB Debug=true %>
<%@ Import Namespace="System.Web.Mail" %>
```

Directives are instructions to the compiler that tell it how the page should be run. In this case, the Page directive is used to indicate the code language on the page. The Import directive indicates a namespace that should be compiled with this page.

That is followed by your page code, which is placed within Script tags:

```
<script runat=server>
Private A as Integer
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)

End Sub
Sub SubmitBtn_Click(Sender As Object, E As EventArgs)

End Sub
</script>
```

Notice that the Script tag includes the parameter RunAt with a value of Server. That means that this code is server-side code and should be used by the compiler when the page loads or is posted back for processing.

Within the Script tags, you define the procedures for the page. These procedures can be made up of event procedures that fire when something happens on the page, such as when a button is clicked or the page loads. These procedures can also be made up of your own procedures that are called from other event-driven procedures.

After defining the code for your page, you can start including the HTML tags for the page structure:

```
<HTML>
<HEAD>
<TITLE>Coding the Basic ASP.NET Page Structure</TITLE>
</HEAD>
<Body LEFTMARGIN="40">
```

That is typically followed by an opening ASP.NET Form tag:

```
<form runat="server">
```

Within the Form tag, you define your ASP.NET controls mixed in with standard HTML elements:

```
<asp:label
    id="lblDataEntered"
    runat="server"
/>
<BR><BR>
Enter your Name:<BR>
<asp:textbox
    id="txtName"
    columns="25"
    maxlength="30"
    runat=server
/>
<BR><BR>
<asp:button
    id="butOK"
    text="  OK  "
    onclick="SubmitBtn_Click"
    runat="server"
/>
```

Here, a Label control, a TextBox control, and a Button control are defined. Notice that between the control definitions, HTML elements are defined.

Once you define the controls and other content on the page, you need to close the ASP.NET Form control:

```
</Form>
</BODY>
</HTML>
```

Notice that the HTML tags are also closed.

Using the Load Event of the Page Object

Frequently, you need to include code on your page that fires when the page loads. This type of code is used to set up the items on your page. For example, if you wanted to display a list of products, you would need to retrieve those products from a database when visitors view the page. You do this by creating a Load event procedure for the Page object. This technique shows you how to create that procedure.

USE IT The page created for this technique displays to visitors text through two Label controls. The text in these controls is set in code when the page loads.

The Label controls are defined within the form on the ASP.NET page. The first Label control has this definition:

```
<asp:label
    id="lblFixed"
    runat="server"
/>
```

The other Label control has this definition:

```
<asp:label
    id="lblStatic"
    runat="server"
/>
```

When the page loads, the following code runs:

```
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    lblFixed.Text = "Welcome to the page!"
    lblStatic.Text = "The current time on the server is " _
        & TimeOfDay()
End Sub
```

Notice that the name of the procedure is Page_Load:

```
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
```

If you want the code to run whenever the page loads, you must give the procedure this name. Also, notice that two parameters are passed into this procedure. If you do not supply these two parameters, your page will not run and an error will be returned.

Within the procedure, you define code that you want to run every time the page loads:

```
lblFixed.Text = "Welcome to the page!"
lblStatic.Text = "The current time on the server is " _
    & TimeOfDay()
```

In this case, fixed text is displayed in the first Label control and the current system time is displayed in the other Label control.

You then need to close the procedure's definition:

```
End Sub
```

Using the Unload Event of the Page Object

Another event procedure of the Page object that you can provide code for is the Unload event. This event fires when the page is finished processing and is being removed from memory. You can use this procedure to write any final clean-up code such as closing connections to databases or storing any closing statistical values. Note though that you cannot change the controls on the page during this event because they already will have been rendered. This technique shows you how to use the Unload event.

USE IT The page created for this technique displays text to visitors when the page loads. Then, when the page is done processing, the time that the page finished is stored in a persistent variable.

Defined within the ASP.NET page are these three Label controls:

```
<asp:label
    id="lblFixed"
    runat="server"
/>
<BR><BR>
<asp:label
    id="lblStatic"
    runat="server"
/>
<BR><BR>
<asp:label
    id="lblDone"
    runat="server"
/>
```

When the page loads, this code fires:

```
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    lblFixed.Text = "Welcome to the page!"
    lblStatic.Text = "The current time on the server is " _
        & TimeOfDay()
    lblDone.Text = Session("FinalText")
End Sub
```

Notice that the text in the third Label control is set to a value in a Session variable. That variable gets its value in the Unload event, so the first time the page loads, this Label will not display any text. But on subsequent calls, the Label control will display the time the page finished processing.

That Session variable has its value set in the Unload event:

```
Sub Page_Unload(ByVal Sender as Object, ByVal E as EventArgs)
    Session("FinalText")  = "During last load, the " _
        & "page finished processing at: " & TimeOfDay()
End Sub
```

Therefore, when the page is done processing, the Session variable is set to the system time.

Redirecting Visitors When an Unhandled Error Occurs

Your code may, on occasion, produce results that cause an error to occur. When that happens, you can choose to simply let visitors see the error message on the page. But you could choose to redirect visitors to another page when an unhandled error occurs. On such a page, you could display friendly help telling them to retry at a later time or to contact your company for support. You can do this on an ASP.NET page through the use of the Error event. This tip shows you how to use that event.

USE IT The page created for this technique produces an error. The error occurs in the Load event of the Page object:

```
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    Dim I as Boolean
    I = "hello"
End Sub
```

Notice that the variable “I” is declared as a Boolean. Therefore, it should be assigned a True/False value. But in this procedure, it is set to a string. That will cause an error.

Typically, visitors to this page would see the standard error page. But this page includes an Error event procedure:

```
Sub Page_Error(Sender as Object, e as EventArgs)
    Response.Redirect("./errorshappen.aspx")
End Sub
```

The effect of this procedure is that when an error occurs on this page that is not handled, the code in this procedure fires. In this case, visitors are sent to another page. You would then supply that other page and let visitors see a friendlier message with regards to the problem that occurred.

Using the IsPostBack Property of the Page Object

Frequently, you will have code on your ASP.NET page that you want to run when the page loads. But you will only want the code to run the first time the page loads. When visitors submit the page for processing, you would not want the code to run again. This technique shows you how to do that.

USE IT The page created for this technique defines two Label controls. One Label control is assigned the current time every time the page loads. The other Label control is assigned the current time only the first time the page loads.

The Label controls have this definition:

```
<asp:label
    id="lblFirstTime"
    runat="server"
/>
<BR><BR>
<asp:label
    id="lblOtherTimes"
    runat="server"
/>
```

Also defined on the page is a Button control that allows subsequent calls to the page to be performed:

```
<asp:button
    id="butOK"
    text="  OK  "
    onclick="SubmitBtn_Click"
    runat="server"
/>
```

The following code fires every time the page loads:

```
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    If Not Page.IsPostBack Then
        lblFirstTime.Text = "The page first loaded at: " _
            & TimeOfDay()
    End If
    lblOtherTimes.Text = "The last page load fired at " _
        & TimeOfDay()
End Sub
```

Notice the use of the IsPostBack property of the Page object:

```
If Not Page.IsPostBack Then
```

That property returns True if the page has been returned for processing. In other words, the property is set to True when visitors click the Button control. But it is set to False the first time the page loads.

When it is the first load of the page, one of the Label controls is assigned the current system time:

```
lblFirstTime.Text = "The page first loaded at: " _
    & TimeOfDay()
```

The other Label control will have its value assigned regardless of whether the page is initially loading or not:

```
lblOtherTimes.Text = "The last page load fired at " _
    & TimeOfDay()
```

Creating Your Own Procedures

In addition to writing event procedures on your page that fire when an event such as a page loading occurs, you can create your own procedures. You call these procedures from the event procedures. You will find doing this is helpful as your ASP.NET pages become more complex and you are repeating code within event procedures. This technique shows you how to create your own procedures.

USE IT The page created for this technique prompts the visitor for two numbers. Those numbers are added when the page is submitted. The page includes two user-defined procedures that are called from two event procedures. One of the procedures is a Function, so it returns a value. The other procedure does not return a value, since it is a Sub.

Defined on the page is this Label control, which will display page instructions to visitors:

```
<asp:label
    id="lblInstructions"
    runat="server"
/>
```

Next, two TextBox controls are defined for entering the two numbers that will be added:

```
<asp:textbox
    id="txtNumber1"
    runat=server
/>
<asp:textbox
    id="txtNumber2"
    runat=server
/>
```

After those controls a Button control is defined:

```
<asp:button
    id="butOK"
    text="Add"
    onclick="SubmitBtn_Click"
    runat="server"
/>
```

A second Label control is defined that will display the result of adding the numbers:

```
<asp:label
    id="lblResult"
    runat="server"
/>
```

The first procedure defined on the page is called DisplayInstructions:

```
Sub DisplayInstructions(Mode as String)
    If Mode = "Initial" Then
      lblInstructions.Text = "Enter two numbers that " _
          & "you want to add."
    Else
      lblInstructions.Text = "The results are " _
          & "below."
    End If
End Sub
```

This procedure displays different information into the Label control, based on whether the page is in initial load mode or a subsequent posting. The procedure is called from the Load event:

```
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    If IsPostBack Then
        DisplayInstructions("Final")
    Else
        DisplayInstructions("Initial")
    End If
End Sub
```

Notice that the IsPostBack property is used to determine whether the page is in its initial load or whether the page is loading because the form on the page was posted.

The other procedure defined on this page is called AddNums:

```
Function AddNums(Num1 as Single, Num2 as Single) as Single
   AddNums = Num1 + Num2
End Function
```

This procedure requires two parameters to be passed in. Those are the numbers to be added. The procedure returns the result of adding those two numbers.

That procedure is called when the Button control is clicked:

```
Sub SubmitBtn_Click(Sender As Object, E As EventArgs)
    lblResult.Text = "Result: " & AddNums( _
        txtNumber1.Text, txtNumber2.Text)
End Sub
```

Notice that the two numbers entered into the TextBox controls by visitors is passed into the procedure call.

Using the Page Directive

Directives are instructions that you supply to the compiler to indicate how a page should be compiled. One of the directives that you are likely to include on every page is the Page directive. This technique shows you how to define that directive.

USE IT The ASP.NET page created for this technique uses the Page directive to set a variety of options. The Page directive, used on that page, has this syntax:

```
<%@ Page
    Language=VB
    EnableSessionState=false
    Debug=true
    Buffer=true
    Explicit=true
    Strict=false
    Description="Sample Page directive page."
%>
```

You start by indicating the name of the directive:

```
<%@ Page
```

Notice the specific characters used to open a directive.

Then within the Page directive you can start setting its parameters. One of those is the Language parameter:

```
Language=VB
```

This parameter is set to the name of the programming language you are using on the page.

Next, the EnableSessionState parameter is set:

```
EnableSessionState=false
```

A value of True in this parameter means that Session variables can be created and they will persist. A value of False means that these variables cannot be used on this page. The parameter can also be set to ReadOnly, which means that Session variables can be read but not changed.

The next parameter set is the Debug parameter:

```
Debug=true
```

Setting this parameter to True sometimes provides you with more information when an error occurs.

QUICK TIP

Set the Debug parameter to False when you move your page to a production machine. This will reduce page overhead.

Next, the Buffer parameter is set:

```
Buffer=true
```

If this parameter is set to True, the page will be sent after all the processing is complete. If the parameter is set to False, the page is sent as it is processed.

The next parameter, Explicit, is only used with Visual Basic.NET:

```
Explicit=true
```

When you set this parameter to True, you must declare all your variables.

The Strict parameter also is only used by Visual Basic.NET:

```
Strict=false
```

Setting this parameter to False means that the compiler will allow you to place a value of one data type into another data type without throwing an error.

The last parameter set is the Description parameter:

```
    Description="Sample Page directive page."
```

This parameter is just for you. It provides a place for you to define the description of the page.

Using the Import Directive

To instantiate objects of a class, the compiler needs to know where the definitions for those classes can be found. In .NET architecture those definitions can be found within a namespace. To indicate to the compiler that you want to use classes in a namespace on your ASP.NET page, you use the Import directive. This tip shows you how to do that.

USE IT The page created for this tip imports three namespaces. It does this using the Import directive. Those directives are defined towards the top of the ASP.NET page:

```
<%@ Import
    Namespace="System.Web.Mail"
%>
<%@ Import
    Namespace="System.Data"
%>
<%@ Import
    Namespace="System.Data.OLEDB"
%>
```

Only one parameter is used with the Import directive. That is the Namespace parameter. You place into that parameter the name of the namespace that you wish to include in the compilation of your page. Notice that each namespace is imported through a separate Import directive.

Using the OutputCache Directive

The OutputCache directive provides a way for you to have your page only be processed at specific time intervals. This is helpful when you have a resource-intensive page that does not change frequently. For example, if you had a page that lists all the employees at your company through a DataGrid control, you really don't need this page to grab fresh data from the database every time the page is loaded. You would use the OutputCache parameter to indicate how frequently you want the page to refresh. This tip shows you how to do that.

USE IT The page created for this tip displays the current time to visitors when the page loads. But since the OutputCache directive is used, the time displayed is only updated after 30 seconds have elapsed.

Defined within the page is this Label control:

```
<asp:label
    id="lbl1"
    runat="server"
/>
```

That control has its Text property set when the page loads:

```
Sub Page_Load(ByVal Sender as Object, ByVal E as EventArgs)
    lbl1.Text = "The current time on the server is " _
        & TimeOfDay()
End Sub
```

But the text will not be set every time the page is loaded because of this OutputCache directive:

```
<%@ OutputCache
    Duration=30
    VaryByParam=none
%>
```

The Duration parameter is set to the number of seconds that the page will be cached. After that time elapses, the page will run again.

Notice the use of the VaryByParam parameter. You must include this parameter. You can use this parameter to have caching be based on a value passed in through the Get or Post methods. You can set this parameter to None if you want caching to occur regardless of how the page loads.