WORKING WITH ASP.NET

UNIT IV

V.B Buvaneswari

What is ASP.NET?

ASP.NET is the framework you'll use to build Web applications for the .NET Framework. One of the design goals of ASP.NET was to bring to the Web the same rich application features (as close as possible) used in traditional desktop applications. ASP.NET achieves this by providing a flexible development methodology that allows the use of any .NET-capable language to code dynamic Web pages. ASP.NET also frees up the Web application developer from many mundane programming tasks by providing built-in code objects for many common Web-programming tasks.

Features of ASP.NET!

ASP.NET provides a programming model that contains many features to support rich application functionality. In this chapter, you'll learn how you can apply these features to your own Web applications.

Features

Compatibility with Any .NET Language

Many Web application development environments don't allow for many of the features that traditional programming languages provide, which is why one of the primary design goals of ASP.NET was to make Web applications as rich in features as their desktop counterparts.

Compiled Code

Many Web application development environments don't provide an environment that takes advantage of features available to desktop applications, primarily because Web development languages are generally interpreted. Prior versions of ASP are an example of this. ASP in its previous incarnations

Features

consisted of an interpreted environment based on ActiveX scripting languages such as VBScript and JScript. Interpreted languages have a distinct disadvantage—they are subject to the inefficiency of code interpretation each time the code is executed (that is, every time the page is requested from the Web server).

ASP.NET addresses the inefficiency of interpreted code. When a user first requests a Web page with ASP.NET code, ASP.NET compiles the code embedded in the Web page. This not only provides great speed enhancement but also allows for other features such as strong variable typing and early binding to objects. In addition, ASP.NET applications can cache data for faster delivery through special Application Programmer's Interfaces.

Features

Easy Application Deployment

Many Web applications are tightly integrated with the Web server, which makes deployment and maintenance of the application difficult. Before the development of ASP.NET, replacing Web application components sometimes required a restart of the Web server, and some components (like COM objects) needed to be registered with the operating system in order to be used. ASP.NET addresses these issues by removing those prohibitive requirements and making deployment and maintenance as simple as copying files to the appropriate directories on the server. In some cases, no further configuration is required. ASP.NET also solves the problem of swapping out new components or pages, not by locking them, as other systems do, but by making copies of the existing pages and components for the pending requests. Then, after all pending requests have been fulfilled, the new pages and components become active.

Features

ASP.NET addresses the inefficiency of interpreted code. When a user first requests a Web page with ASP.NET code, ASP.NET compiles the code embedded in the Web page. This not only provides great speed enhancement but also allows for other features such as strong variable typing and early binding to objects. In addition, ASP.NET applications can cache data for faster delivery through special Application Programmer's Interfaces.

Simple Application Configuration

Many Web applications have distinctive application settings that control various parts of the Web application, such as page request time-outs. Unlike many Web application development environments (including prior versions of ASP), ASP.NET stores these settings in human-readable XML configuration files. These XML configuration files are placed in common directories, making them easy for administrators to locate. New settings can be added easily to XML-based files, and the settings are stored in a hierarchical fashion for easy referencing.

Advanced Session and Application State Management

Applications, be they desktop-based or Web-based, need to retain information about current conditions in the application. Given that the Web client/Web server model is inherently stateless (no data are saved between requests for pages), it becomes the job of the application environment to track the application state. ASP.NET has several offerings for maintaining application state, and some are even compatible with older versions of ASP. And, of course, ASP.NET offers advanced state management that improves on the legacy methods of state management.

Integrated Security Features

Security is an important concern in Web programming. ASP.NET leverages the security features of server versions of Windows (.NET/XP/2000/NT), which enable you to implement security in a variety of ways. Users can be authenticated using traditional methods provided by IIS like BASIC and Windows (formerly known as challenge/response authentication) as well as newer methods like Microsoft Passport. Protecting the site's files can be done by assigning user and group access to the individual files and directories. In addition, authorized users and groups can be stored in an XML file.

Direct Low-Level HTTP Support

Modern Web applications usually consist of HTML and executable code interspersed within the HTML. But Web applications are sometimes supplemented by lower-level programs that interact with the Web server through a tight integration. Typically, these programs modify the default behavior governing how requests for documents are handled by the Web server. In the past, such low-level programs needed to be written in low-level languages such as C++. But with the advent of ASP.NET, these programs can now be written as normal ASP.NET pages in languages like VB.NET.

Backward Compatibility with ASP

If you have experience with prior versions of ASP, you'll be glad to know that ASP.NET works quite nicely with regular ASP. In fact, applications written in classic ASP can run alongside ASP.NET applications without any ill effects. This makes upgrading and transitioning your existing ASP code base to ASP.NET less difficult by allowing a gradual phasing out of old ASP application code.

ASP.NET pages begin with a plain-text file that has an `.aspx` extension. IIS uses this extension to identify particular files that contain executable code and

should be processed as ASP.NET pages (as opposed to plain HTML files or other file types).

The contents of the `.aspx` file are called the *page*. The page consists of visual markup (HTML) and application logic (executable code written in a .NET language). ASP.NET also has the ability to store application code in separate files rather than grouping it with the HTML. For organizational and structural reasons, placing application code in a separate file is usually a good idea. This concept is called *code behind*, and we'll see how it works in just a moment.

The page I referred to earlier is actually an ASP.NET class. When an ASP.NET class is compiled, a new class that derives from the `Page` class is generated automatically. In reality, this new class is an executable program. Once it is initially compiled by ASP.NET, it is run each time a request for its corresponding `.aspx` file is made. For compatibility with all Web browsers, the output of the executable can be HTML 3.2. If the `.aspx` file changes, the page is recompiled into a new class. The dynamically generated class also contains the page elements (HTML elements, controls, and so on) as class

generated automatically. In reality, this new class is an executable program. Once it is initially compiled by ASP.NET, it is run each time a request for its corresponding `.aspx` file is made. For compatibility with all Web browsers, the output of the executable can be HTML 3.2. If the `.aspx` file changes, the page is recompiled into a new class. The dynamically generated class also contains the page elements (HTML elements, controls, and so on) as class members of the generated class. The compilation occurs only during the first request of a page and after modifications have been made to the page's code.

Let's take a quick look at a very simple `.aspx` file to familiarize you with its structure. The following code listing shows a typical `.aspx` file.

Code Samples

```
<%@ Page Language="vb"
❶     AutoEventWireup="false"
%>

<html>
<body>
<form id=WebForm1
   method=post
❷     runat="server">

<h1>Example .aspx File</h1>
Enter a value:

❸ <asp:TextBox id=txtEnteredVal
   runat="server">
</asp:TextBox><br>

❹ <asp:Button id=cmdSend
   runat="server"
   Text="Send">
</asp:Button><br>

❺ <asp:Label id=lblDisplay
   runat="server"
   Width="101"
   Height="19">
</asp:Label>

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

Anatomy of ASP.NET

At first glance, this `.aspx` file looks much like an ordinary HTML file, but on closer inspection, you'll see several items that you won't recognize as standard HTML (see the numbered lines). This `.aspx` file contains three new items of interest:

1. ASP.NET page directives (shown in line ❶)
2. The `runat="server"` attribute (shown in line ❷)
3. Web Controls (shown in lines ❶, ❹, and ❺)

Page directives provide ASP.NET with information about the code contained in the file. They can be located anywhere in the file, but they are generally placed at the top. Page directives allow for customization of compiler settings.

I stated earlier that ASP.NET allows you to separate content (HTML) from application logic. Often you will want to expose some HTML elements to the ASP.NET object model. These HTML items, in a sense, are executed on the server. The `runat="server"` attribute instructs the page compiler to preprocess these HTML elements on the server rather than simply serving up

Anatomy of ASP.NET

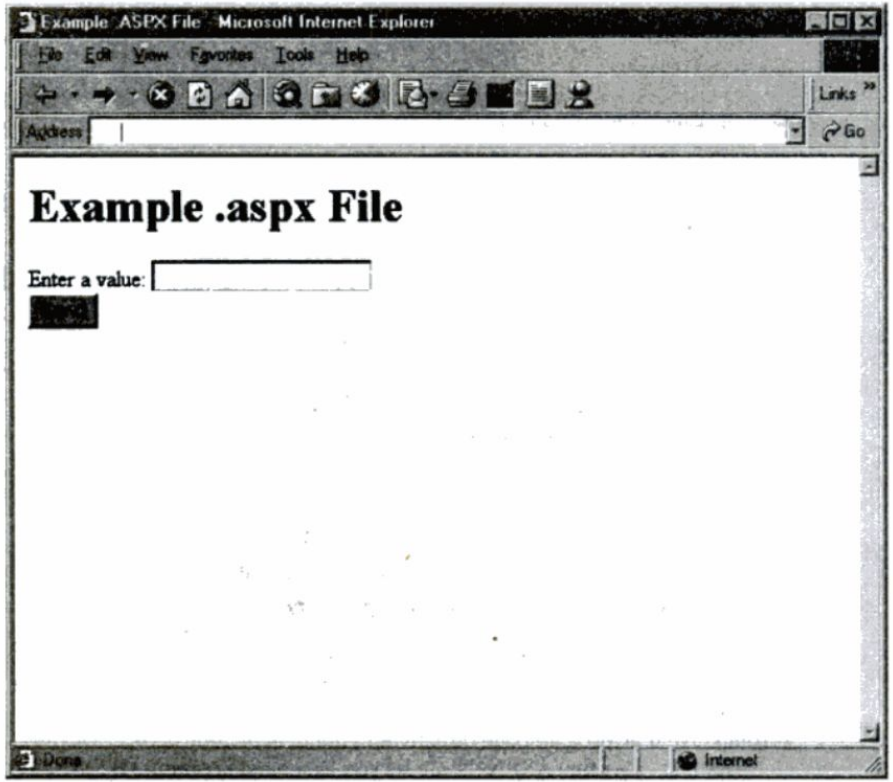
to the ASP.NET object model. These HTML items, in a sense, are executed on the server. The `runat="server"` attribute instructs the page compiler to preprocess these HTML elements on the server rather than simply serving up the HTML text back to the browser. The HTML `<form>` tag that appears just above line ❷ shows the use of the `runat="server"` attribute.

Web Controls are server-side code pieces that perform a variety of functions.

for now, let's say that the Web Controls defined in lines ❸ through ❺ are user-interface elements that interact with the server.

When a browser requests the `.aspx` file shown above, the result returned to the browser looks something like Figure

Output:


Example .ASPX File - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Address
Go
Example .aspx File
Enter a value:
Done
Internet

Anatomy of ASP.NET

The HTML delivered to the browser should look familiar since it is plain HTML 3.2 (some formatting has been added for readability).

```
<html>
<head>
<title>Example .ASPX File</title>
</head>
```

Anatomy of ASP.NET

```
<body>
<form name="WebForm1" method="post"
   action="Webform1.aspx" id="WebForm1">
<input type="hidden" name="__VIEWSTATE"
   value="YTB6LTEwNzAyOTU3NjJfX1949e1355cb" />

<h1>Example .aspx File</h1>
Enter a value:

<input name="txtEnteredVal"
   type="text" id="txtEnteredVal" /><br>

<input type="submit" name="cmdSend"
   value="Send" id="cmdSend" /><br>
```

Anatomy of ASP.NET

```
<span id="lblDisplay"
   style="height:19px;width:101px;"></span>

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

ps really occur when a Web browser requests an ASP.NET page? It's important to understand that Web applications (be they ASP.NET or another type) don't behave like traditional desktop applications. Web applications use a request–response model of communication: the Web browser requests a document from the Web server and the Web server responds with the data from that document. Web applications also contain HTML forms, which allow for user input. The Web browser must package up the data entered in the form before sending it to the server. The data in the form can then be used as input for an ASP.NET program. The program is executed using the form data as input, and the results are returned to the browser as HTML. This sequence of steps is referred to as a *round-trip*.

Round-trips are usually required when the user of the Web application expects a meaningful response to a query issued from an HTML form. In some instances, round-trips are not required. If the Web browser is capable, it

Execution stages and state Management

expects a meaningful response to a query issued from an HTML form. In some instances, round-trips are not required. If the Web browser is capable, it can provide immediate feedback from form input. This type of feedback is generally limited to simple data validation and is handled using client-side scripts. Data validation using this method can check for conditions such as form completeness, numerical ranges, and correct data formatting. Functions such as searching a database require a round-trip to the server since the data to be searched is stored on the server.

subsequent requests to other pages may want to access data values returned from other pages. Traditional desktop applications never had a problem keeping state. Because desktop applications are typically loaded into memory once and are kept in memory until the user exits the application, the program can freely store any values it needs in the memory allocated for the application. As Web applications become more advanced and function more like desktop applications, a reliable state-management system is required.

Fortunately, many state-management tasks in ASP.NET are done for you, and other state-management tasks can be accomplished in a number of ways. The ASP.NET Web Controls and HTML Controls, for instance, can maintain their associated values between round-trips without any coding effort on your part. This is called the *view state*.

View state is maintained using the `__VIEWSTATE` hidden form field, which is automatically generated by ASP.NET for every `.aspx` file that contains a server-side form. In the previous section, the source generated by ASP.NET showed the `__VIEWSTATE` hidden field with an encoded text string as its value. This encoded text string contains information about the state (values) of the server-side controls on the form. When the form is posted back to the server, the `__VIEWSTATE` will be posted along with the other form values. Form values that haven't been changed by the user are restored with values encoded in the `__VIEWSTATE` value from the previous post.

The Events Model for the Page Class

The `Page` class contains events that fire when (1) the page loads into the browser and (2) when server code execution ends and page rendering has

The Events Model for the Page Class(cont..)

You can specify event-handling code for these events in your `.aspx` file. Consider the following `.aspx` file with code in place for responding to the `Page_Load` and `Page_Unload` events.

Code Sample

```
<%@ Page Language="vb"%>
<html>
<head>

<script language="vb" runat="server">
   Sub Page_Load(Sender as Object, E as EventArgs)
      lblSampleLabel.Text = "Loaded!"
   End Sub

   Sub Page_Unload(Sender As Object, E as EventArgs)
      ' Cleanup code goes here
   End Sub
</script>
</head>
```

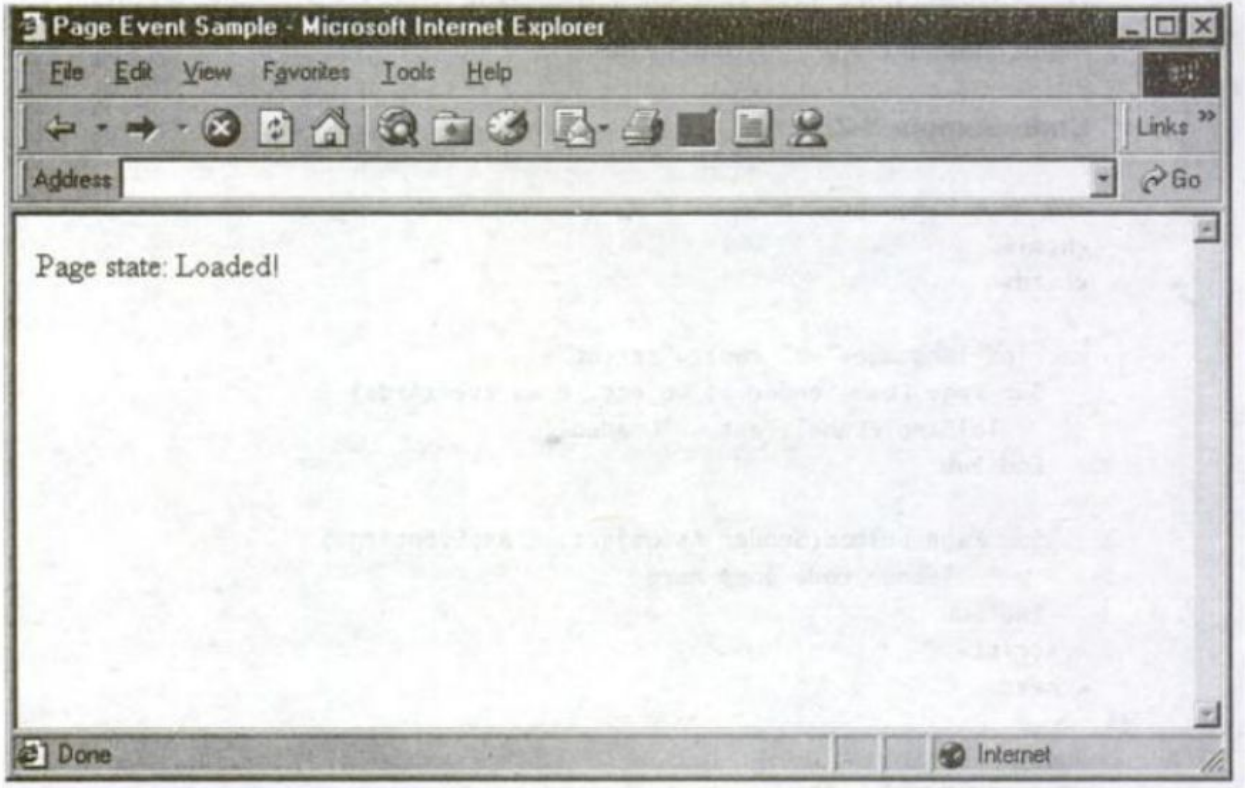
```
<body>
<form id="WebForm2"
   method="post"
   runat="server">

Page state:
<asp:Label id=lblSampleLabel
   runat="server">
</asp:Label>

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

The Events Model for the Page Class (cont..)

When we request this page, we get the output shown in Figure

The `Page_Load` event is fired when the page and all its controls are rendered. The code then displays the message "Loaded!" in the `lblSampleLabel` Control. `Page_Unload` is fired when the page is about to be discarded. This is where cleanup code is placed (freeing objects and so on). Any references made to visual elements in this event are not recognized since `Page_Unload` is fired after page processing is complete.

Output:


Page Event Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Address
Go
Page state: Loaded!
Done
Internet

Introducing Web Forms

Web Forms is the name given to the technology that provides dynamic Web pages with ASP.NET. In a sense, Web Forms allow programmers to code against the page itself in an object-oriented manner. The Web page and all the objects contained within it (controls, text, and so on) can be accessed using an object model.

Web Forms provide many distinct programming advantages. They provide an event-based programming model. This means that objects on Web Forms such as buttons, controls, and text have behaviors as well as appearances. For example, a button can exhibit a particular behavior when it is clicked. The clicking of the button is an event. Events cause certain sections of code to execute inside the program, like displaying new data to the user or submitting data to the server. Event-based programming models are common with desk-

Introducing Web Forms

top application development, but ASP.NET Web Forms deliver, for the first time, this type of functionality to the Web browser.

Web Forms allow for the separation of application logic from content. On older Web-development systems, application code (typically script code) was interspersed with HTML. The main advantage of this separation is to facilitate better team-based development. Designers who work with layout and content can create the visual look and feel of the page without interrupting the work of the Web programmers who place the application logic that drives the page's functionality.

Introducing Web Forms

Web Forms work closely with the VS.NET Integrated Development Environment (IDE) to provide an easy design-time development experience. Users of Visual Studio have long enjoyed the benefits of rapid application development (RAD) by combining event-based programming with easy-to-use form design tools. With VS.NET, you can use those same RAD features to construct ASP.NET pages with greater ease than ever before. In addition, if you've programmed applications using Visual Basic's Forms Designer in the past, you'll be pleased to find a similar interface in VS.NET.

Introducing Web Forms

you've programmed applications using Visual Basic's Forms Designer in the past, you'll be pleased to find a similar interface in VS.NET.

ASP.NET frees developers from having to write less routine code typically associated with Web applications. It accomplishes this by providing a standard set of server-based controls for many common tasks, such as data entry validation. We'll investigate these great timesaving controls in coming sections.

VS.NET Web Applications and Other IDE Basics

VS.NET offers an easy way to create new ASP.NET Web application projects. You can create a new ASP.NET Web application in the same way you create other types of applications in VS.NET: by selecting **File→New→Project** from the menu bar and clicking the ASP.NET Web Application button

Adding new Web Forms to the application is simple. In the Solution Explorer window, right-click on the icon for your Web application and select **Add→Add Web Form** . . . as shown in Figure . Then select the Web Form icon in the Templates section of the screen and give your new Web Form a file name (see Figure). The new Web Form is saved and becomes part of the current solution.

Whenever you want to view the code or when you place a Web Control on the Web Form for the first time, a new code module is added to the solution that serves as the code-behind module for the Web Form. A newly created code-behind module looks like the sample below. (Some formatting has been added for readability.)

Code Sample

```
❻Public Class MyWebForm
❼    Inherits System.Web.UI.Page

#Region " Web Form Designer Generated Code "

    ' This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()

    End Sub

❽     Private Sub Page_Init(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles MyBase.Init
          InitializeComponent()
```

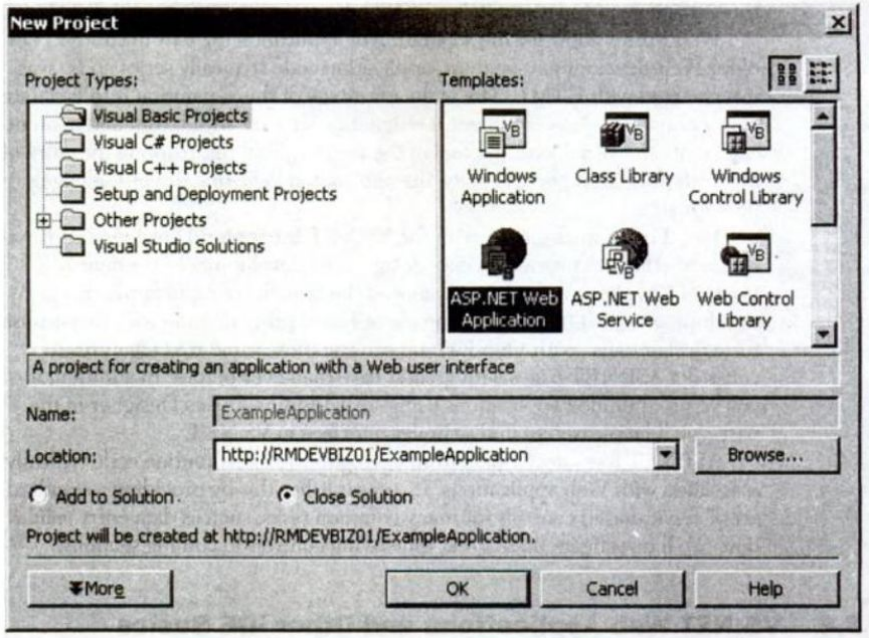
Code Sample

```
    End Sub

#End Region

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

    End Sub

End Class
```

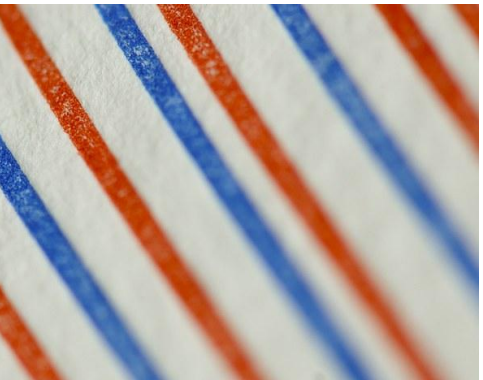
This code-behind module defines a new class for the Web Form and creates plumbing code. Notice that VS.NET assigned the class the same name

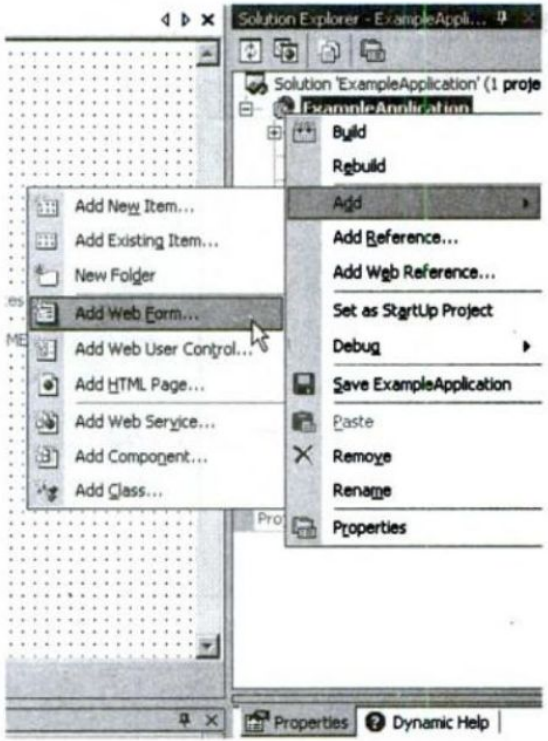
VS.NET Web Applications and Other IDE Basics

code-behind module looks like the sample below. (Some formatting has been added for readability.)

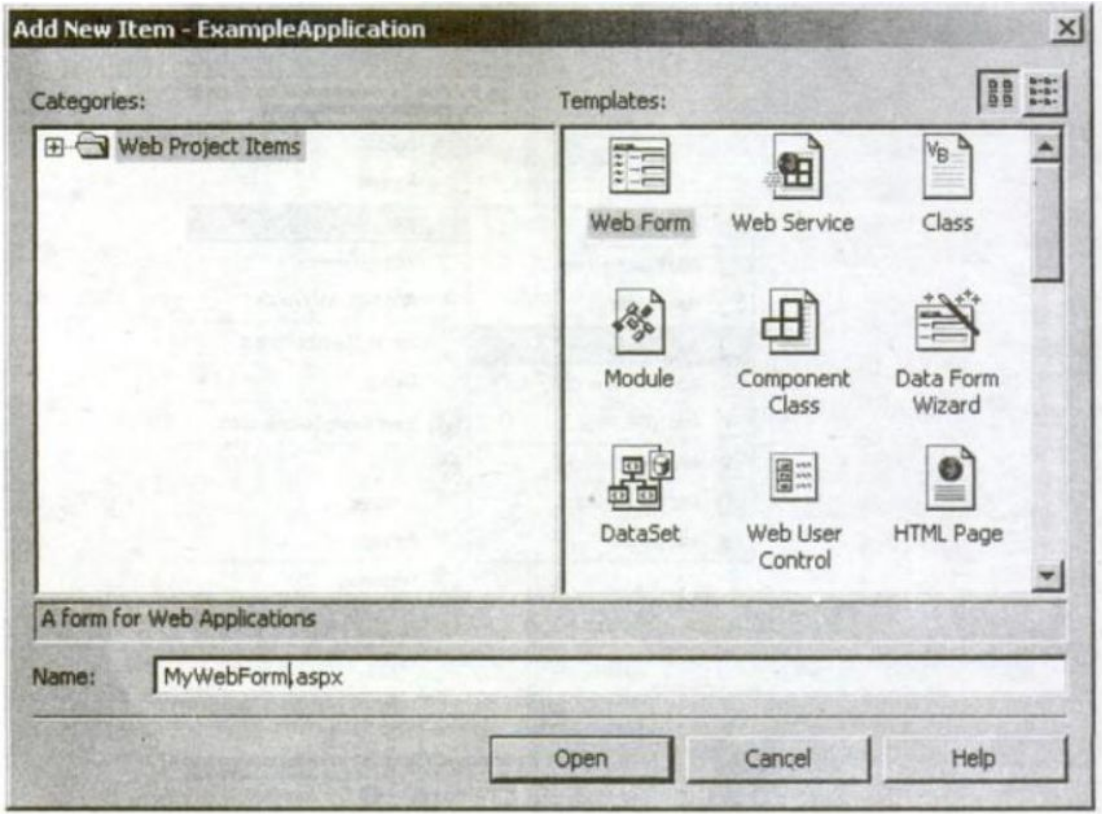


VS.NET Web Applications and Other IDE Basics





VS.NET Web Applications and Other IDE Basics



given to the Web Form in Figure `MyWebForm` (see line ❻). This class inherits from the `Page` class (see line ❼), which is a predefined ASP.NET class. This exposes the `Page` class's events and other properties and methods so they can be used within the new class. Two inherited `Page` event handlers are defined for us, `Page_Init` and `Page_Load`, in lines ❽ and ❾, respectively.

As you develop a Web Form using the VS.NET IDE, you will see code added to the code-behind module. In particular, code will be generated in response to new Web Controls placed on the Web Form. During the discussions of Web Controls that appear later in this book, we will constantly be referring back to the code-behind file, so it's important to familiarize yourself with its structure.

Separating Content and Code [the Code-Behind Feature]

When building Web applications with VS.NET, you can greatly simplify the amount of code that you'll actually have to type. VS.NET also offers many features to keep your code organized, such as the code-behind feature.

Code behind came about when design and organizational issues arose from mixing application code with HTML (that is, mixing logic with presentation). Many programmers combined the two elements within the same file (as many did with ASP 3.0). The result was a code base that was difficult to maintain and understand due to the chaotic mixture of HTML and script. The code-behind feature alleviates that problem by completely separating HTML from the program code; they are, in fact, in separate files.

The two files are linked together using a special page directive, `Codebehind`, which is defined in the `.aspx` file. `Codebehind` specifies the pathname to a code module file. If the pathname is omitted, the class is assumed to be in the `/bin` directory of the Web application's virtual Web directory. Here's a sample page directive line that contains `Codebehind`.

Separating Content and Code [the Code-Behind Feature]

```
<%@ Page Language="vb"
    AutoEventWireup="false"
    Codebehind="SampleWebButton.vb"
    Inherits="WebApplication1.SampleWebButton"
%>
```

When you use the VS.NET IDE to create Web applications, the IDE is configured to generate the above code for you. Whenever you create a new Web Form (from the **New** menu), this and other "boilerplate" Web Form code is placed in the file. The code-behind module (the `.vb` file) is also generated with some event-handling code (we'll discuss this in a moment).

You have other options when specifying a code-behind class. You may also specify an `Src` attribute, which tells ASP.NET where your class file is located. If omitted, ASP.NET assumes that the class is located in the `/bin` directory of the Web application. Looking at the previous example, we can represent the same thing using this form.

Separating Content and Code [the Code-Behind Feature]

```
<%@ Page Language="vb"
    AutoEventWireup="false"
    Src="SampleWebButton.vb"
    Inherits="WebApplication1.SampleWebButton"
%>
```

Structure and Configuration of the Global .asax file

Applications in ASP.NET are comprised of a virtual Web directory and pages, files, and assemblies contained within the directory and subdirectories. You can define settings for this application plus write event handlers for application-wide events. A special file called the `Global.asax` file controls these settings and events.

Structure and Configuration of the `Global.asax` File

exactly one `Global.asax` file per application, and it is optional. The file is always located at the root level of the virtual directory for the ASP.NET Web application. At first this may seem like a security risk since the `Global.asax` file is a file like any other in the Web application directory that users may request. The `Global.asax` file can also contain code that may compromise the security of the Web server. Because of this potential vulnerability, ASP.NET places a restriction on the `Global.asax` file so that any direct HTTP request for it is automatically denied.

Structure and Configuration of the Global .asax file

HTTP request for it is automatically denied.

When VS.NET creates a new Web application, a `Global.asax` file is created automatically. The `Global.asax` file contains a code-behind file named `Global.asax.vb`. Here's how a typical VS.NET-generated `Global.asax` file looks:

```
<% Application Codebehind="Global.asax.vb" Inherits="Lab3_1.Global" %>
```

The code-behind file, `Global.asax.vb`, looks like this:

Code Sample

```
Imports System.Web
Imports System.Web.SessionState

Public Class Global
   Inherits System.Web.HttpApplication

   Sub Application_Start(ByVal Sender As Object, _
      ByVal e As EventArgs)

   End Sub

   Sub Session_Start(ByVal Sender As Object, ByVal e As EventArgs)

   End Sub
```

Code Sample

```
    Sub Application_BeginRequest(ByVal Sender As Object, _
       ByVal e As EventArgs)

    End Sub

    Sub Application_EndRequest(ByVal Sender As Object, _
       ByVal e As EventArgs)

    End Sub

    Sub Session_End(ByVal Sender As Object, ByVal e As EventArgs)

    End Sub

    Sub Application_End(ByVal Sender As Object, _
       ByVal e As EventArgs)

    End Sub

End Class
```

Structure and Configuration of the `Global.asax` File

The `Global.asax.vb` file defines a class called `Global`, which inherits from `System.Web.HttpApplication`. This class defines all the methods, properties, and events that all application objects use within an ASP.NET application. In this file, VS.NET places overridden stubbed functions that handle events of the `HttpApplication` class. Those events are associated with the beginning and ending of user sessions and of the ASP.NET application.

Credits:

Matt J Crouch, “ASP.NET and VB.NET web programming”, Pearson Education, 2005.