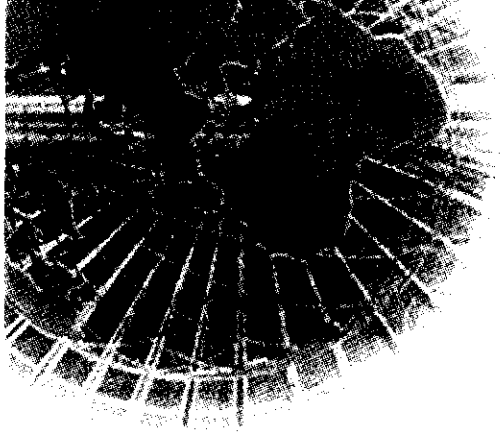


14

ASP.NET 3.5 Essentials

Since the release of .NET Framework 1.0, Microsoft has put in consistent effort in developing and improving ASP.NET, which is a part of .NET Framework for building rich Web applications. This first release brought a radical change over the older Microsoft technology to build websites using a server-side script engine, called Active Server Pages (ASP). Despite many advantages of ASP.NET over ASP, new programmers found it difficult to start with ASP.NET because of its complexity and the knowledge needed to build applications.

After the initial release in 2002, another version of .NET Framework named .NET 1.1 was released with Visual Studio 2003. Most people considered this release as a mere service pack for the initial release, though it also had new improvements in both the framework and the development environment.

Later in November 2005, Microsoft released Visual Studio 2005, which integrated ASP.NET 2.0. Again, Microsoft made drastic improvements in this version and reduced the complexity of the initial version. This new version had many new wizards and controls that reduced the amount of code required in building applications, thus increasing productivity.

Now, Microsoft's latest release is ASP.NET 3.5, which includes new features for creating Web-based applications.

New Features in ASP.NET 3.5

ASP.NET 3.5 was released along with Visual Studio 2008 on November 19, 2007. ASP.NET 3.5 uses the same engine as that of ASP.NET 2.0, with some additional features. ASP.NET 3.5 has simplified the process of building websites, making it easier than ever. Now, let's discuss the following new features added to ASP.NET 3.5:

- ASP.NET AJAX Support
- The ListView Control
- The DataPager Control
- Support for WCF Services
- WCF Support for RSS, JSON and Partial Trust
- Support for Nested Master Pages at Design Time
- Merge Tool in ASP.NET 3.5
- Improved Support for JavaScript Debugging
- Support for Microsoft Office Applications

ASP.NET AJAX Support

In ASP.NET 2.0, ASP.NET AJAX was used as an extension and you had to download the extension and install it in Visual Studio to reap the benefits of ASP.NET AJAX. However, in ASP.NET 3.5, .NET Framework comes with ASP.NET AJAX integration, making the process of building interfaces easier.

The integration between the `WebParts` and `UpdatePanel` controls has become much smoother in ASP.NET 3.5. You can also add ASP.NET AJAX control extenders to the Toolbox in Visual Studio 2008 to extend the functionalities of the different ASP.NET controls.

The ListView Control

The new data control, `ListView`, added to ASP.NET 3.5 contains features of the `GridView`, `DataGrid`, `Repeater`, and similar list controls available in ASP.NET 2.0. Using the `ListView` control, you can insert or delete a page, or edit and sort data. To do so, the `ListView` control uses the `DataPager` control. However, one of the most notable features of the `ListView` control is its great flexibility over the markup generated as you get full control on how the data is displayed. The `ListView` control is useful to display data in any repeating structure, similar to the `DataList` and `Repeater` controls. The `ListView` control also consists of a rich set of templates that you can use to develop attractive websites.

The DataPager Control

As the `ListView` control does not have a built-in support for paging, the `DataPager` control is used to support paging for this control. You can place the `DataPager` control anywhere on a Web Form.

You can do paging with controls that support the `DataPager` control, such as a `ListView` control. The `DataPager` control is used to page through the data that is displayed by the control, which implements the `IPageableItemContainer` interface.

Support for WCF Services

In Visual Studio, you can add both ASP.NET (`.asmx` files) and WCF (`.svc` files) Web services to a project. Client applications using managed code typically access these Web services through a proxy class. For example, when you use the Add Web Reference dialog box, the applications with managed code use the proxy class generated by Visual Studio. Similarly, AJAX applications use Web services from the browser by using proxy classes automatically generated in client script.

WCF Support for RSS, JSON, and Partial Trust

Using WCF, you can now build Web services and expose them with the help of Internet protocols, such as Simple Object Access Protocol (SOAP), Really Simple Syndication (RSS), Java Script Object Notation (JSON), Plain Old XML (POX), and many more. WCF makes creating endpoints easier irrespective of whether you are building an AJAX application that uses JSON, providing syndication of your data through RSS, or building a standard SOAP Web service. Now, .NET Framework 3.5 supports the building of Web services in partial-trust situations similar to a typical shared-hosting environment.

Support for Nested Master Pages at Design Time

Master page support was one of the key features introduced with ASP.NET 2.0 that enables a developer to provide consistent look and feel throughout a website.

The problem with ASP.NET 2.0 was that Web pages based on nested master pages cannot be edited at design time. However, ASP.NET 3.5 solves this problem. You can now edit nested master pages at design time using the Visual Web Designer.

Merge Tool in ASP.NET 3.5

ASP.NET 3.5 introduces a new Merge tool called `Aspnet_merge.exe`, which allows you to combine and manage assemblies created by the ASP.NET precompilation tool (`Aspnet_compiler.exe`). Visual Studio 2008 has the Merge tool as an add-on. This tool creates a single assembly for the entire website, for each website folder, or for just the files that make up the website user interface (UI), which includes Web pages and controls.

Improved Support for JavaScript Debugging

Another important JavaScript feature included in Visual Studio 2008 is the improved support for JavaScript debugging that has made the building of AJAX applications a lot easier. Earlier in Visual Studio 2005, you had to first run your ASP.NET Web forms to set JavaScript breakpoints on them. However, setting JavaScript breakpoints is much easier in Visual Studio 2008. All you need to do is to set the breakpoints directly into the server-side `.aspx` and `.master` source files, as shown in Figure 14.1:

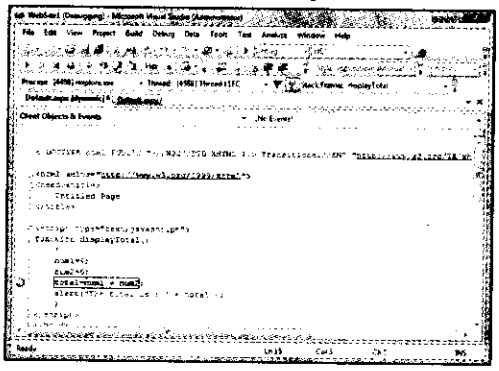


Figure 14.1: JavaScript Debugging

When you close the project, Visual Studio 2008 saves, by default, any breakpoint set in JavaScript. When you reopen the project, you will find these breakpoints still enabled.

After a quick overview of the new features of ASP.NET 3.5, let's now see the various types of ASP.NET websites.

Support for Microsoft Office 2007 Applications

ASP.NET 3.5 provides developers with support for building applications that use Microsoft Office 2003 as well as Microsoft Office 2007, as shown in Figure 14.2:

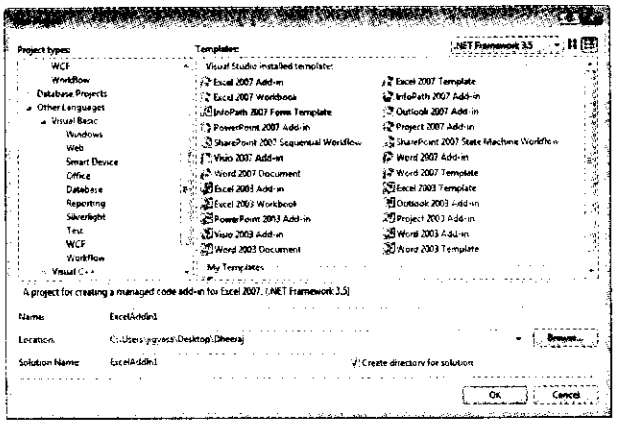


Figure 14.2: Microsoft Office Templates

It supports UI style, including the Ribbon Bar, Ribbon Status Bar, and Mini-toolbar of Microsoft Office applications.

The ASP.NET Life Cycle

The ASP.NET life cycle shows you how ASP.NET processes pages to produce dynamic output. The life cycle also shows you how the application and individual pages are instantiated and processed, and how ASP.NET compiles the pages dynamically. We look at these aspects in detail in the following heads:

- ASP.NET Application Life Cycle on IIS 7.0
- ASP.NET Page Life Cycle

Life Cycle of ASP.NET Applications on IIS 7.0

In this section, we take a look at the complete application life cycle of ASP.NET applications running on integrated mode of IIS 7.0 in addition to using .NET Framework 3.0 or advanced versions. IIS 7.0 also supports Classic mode, which behaves similar to any ASP.NET application running in IIS 6.0.

In an integrated mode of IIS 7.0, application resource request process is similar to IIS 6.0. However, In IIS 7.0 three new events such as `MapRequestHandler`, `LogRequest`, and `PostLogRequest` were raised during request process in addition to events in IIS 6.0. Additionally, IIS 7.0 uses a unified request-processing pipeline for the entire application resource request which was not there in IIS 6.0.

Note: IIS 6.0 uses two separate pipelines for processing an application resource request. One pipeline is for unmanaged code filters and extensions; other is for managed code components.

The following are the benefits of using the integrated pipeline in IIS 7.0:

- Raises all the events of the `HttpApplication` class, to enable ASP.NET modules in IIS 7.0.
- Enables configuration of both managed code and unmanaged code modules. In addition, using the integrated pipeline, we can enable or disable the managed code modules for all requests, even if the user requests for an ASP.NET Web page.

- Enables calling of the managed code modules at any stage in the pipeline, such as prior to the processing of a user request, or after the request is processed, or in the middle of processing a user request.
- Enables in registering and then either enabling or disabling modules by using the `Web.config` file.

Table 14.1 lists the different phases of the life cycle of ASP.NET applications on IIS 7.0 Integrated mode:

Table 14.1: Phases of the Life Cycle of ASP.NET Applications on IIS 7.0 Integrated Mode

Phase	Description
User requests for an application resource	In this phase, application resource is requested by the user on the Web server. This request passes through the unified pipeline that calls the ASP.NET modules. Now, even if a page has `.htm` extension and is not explicitly mapped with ASP.NET, still this request can take benefits of ASP.NET functions such as authentication and authorization process, which are embedded in managed-code. Earlier in IIS 6.0 these processes require manual configuration due to two separate pipelines.
The integrated request-processing pipeline receives the first user request.	When a request for an application resource is received for the first time, an application domain is created by using the object of the `ApplicationManager` class. The application domain is a process that isolates the applications from each other and enables the processing of the user requests. In this phase, the information about an application can also be accessed when the `HostingEnvironment` object is created within the application domain.
For each user request, response objects are created	In this phase, response objects, such as `HttpContext`, `HttpRequest`, and `HttpResponse` are created. The `HttpContext` class has objects, such as the `HttpRequest` and `HttpResponse` objects. These objects are valid only to the current user request. Now, the `HttpRequest` object holds information about the current request. The `HttpResponse` object holds information sent to the user such as cookies and the final output.
An object of the `HttpApplication` class is created and allocated to the request	In this phase, the application is started when an object of the `HttpApplication` class is created. However, if a `Global.asax` file exists, then an object of the `Global.asax` class is created instead of the `HttpApplication` class. The `Global.asax` class derives from the `HttpApplication` class and represents an ASP.NET application. The modules that need to be invoked depend on the parent application's managed-code modules. In addition, it also depends on the modules defined in the `Web.config` file. You can add or remove modules by specifying the module names in the `Web.config` file within the modules element of the `system.webServer` section.
The `HttpApplication` class processes the user request.	The `HttpApplication` class performs the following tasks to process the user request: 1. Validates the user request to confirm that it does not contain any malicious markup. 2. Performs URL mapping to confirm whether any URLs are specified in the `UrlMappingsSection` section of the `Web.config` file. 3. Invokes the following events and methods in the same order as displayed: • `BeginRequest` • `AuthenticateRequest` • `PostAuthenticateRequest` • `AuthorizeRequest` • `PostAuthorizeRequest` • `ResolveRequestCache` • `PostResolveRequestCache`

Table 14.1: Phases of the Life Cycle of ASP.NET Applications on IIS 7.0 Integrated Mode	
Phase	Description
	• MapRequestHandler • PostMapRequestHandler • AcquireRequestState • PostAcquireRequestState • PreRequestHandlerExecute • Calls the ProcessRequest method of the IHttpHandler class for the request • PostRequestHandlerExecute • ReleaseRequestState • PostReleaseRequestState • Performs response filtering only if the Filter property for the HttpResponse class is defined. • UpdateRequestCache • PostUpdateRequestCache • LogRequest • PostLogRequest • EndRequest • PreSendRequestHeaders • PreSendRequestContent

Life Cycle of an ASP.NET Web Page

When a user requests for an ASP.NET Web page, the page passes through a number of stages, which are collectively called the life cycle of an ASP.NET Web page. For example, the life cycle of a page begins when a user requests for a Web page and ends when the page is rendered completely to the browser. When you want a specific action to take place at a particular stage of a page, you should completely understand the page life cycle. Moreover, if you are creating any custom controls for your project, you should have a thorough knowledge of the page life cycle as a control's life cycle depends on the page's life cycle.

Let's now understand about the different stages involved in a page life cycle.

Different Stages of ASP.NET Page

In general, a Web page passes through the stages mentioned in Table 14.2, which comprises of 8 stages. Table 14.2 lists the general page life cycle stages of an ASP.NET Web page:

Table 14.2: Different Stages of an ASP.NET Web Page	
Stage	Description
Page request	Refers that when a user requests for a Web page, only after that the page life cycle begins. The request for the page begins before the start of the page life cycle. When a user requests for a page for the first time, ASP.NET ascertains whether the request is new or an old one. If request is a new one then the page is compiled and executed. However, if the request is an old one then a cached copy of the page is sent without executing the page.
Start	Refers that at this stage, the Request and Response properties of the page are configured. The page also ascertains if the request is an old one or a new request by using the IsPostBack property. In addition, at this stage, the page's UICulture property is also configured.
Page initialization	Refers that at this stage, each control on the page is assigned a unique ID by setting the UniqueID property. At this stage, if there are any themes, they are added to the page. If the request is an old one, then the postback data is not loaded and control property values still need to be restored to the view state values.

Table 14.2: Different Stages of an ASP.NET Web Page

Stage	Description
Load	Refers that at this stage, if the current request is an old one, the properties of the controls on the page are loaded with data from the view state and control state.
Validation	Refers that at this stage, the validation controls used on the page invoke the `Validate` method.
Postback event handling	Refers that if the request made is an old one, then any event handlers are invoked.
Rendering	Refers that prior to rendering the page, the view state for the page and for all the controls on the page are saved. While rendering the page, it invokes the `Render` method for each control on the page and writes the output of the rendering stage to the `OutputStream` object of the page's `Response` property.
Unload	Refers that at this stage the `Response` and `Request` properties of the page are unloaded and any cleanup operation, if required, is performed.

ASP.NET Pages Life Cycle Events

At each stage of the life cycle of a page, the page raises some events that you can handle by running your own code. For handling control events, you bind the event handler to the specified event, using declarative attributes, such as `onclick` or `handle` within code.

ASP.NET Web pages also support automatic event wire-up, which means that ASP.NET searches for methods that have particular names and automatically executes those methods when certain events are raised. If the `AutoEventWireup` attribute of the `@ Page` directive is `True`, page events are automatically bound to the methods used to handle these events. The following are the most common page events:

- Preinit
- Init
- Load

Preinit

Preinit is the first event that occurs during the page life cycle. It is used to check the `IsPostBack` property of the page to determine whether the page is being processed for the first time. It also creates or recreates dynamic controls. It sets the `Theme` property and master page dynamically. In addition, it gets and sets profile property values.

Init

This event is raised after the initialization of all the controls and after any skin settings have been applied. It is used to read or initialize control properties.

Load

The `OnLoad` event method is called to set the properties of controls and establish database connections. The `Page` class calls the `OnLoad` event method on the Web page and then recursively does the same for each child control until the page and all controls are loaded.

NOTE

We have explained some of the main events. However, ASP.NET Life Cycle also supports these events such as InitComplete, PreLoad, LoadComplete, PreRender, SaveStateComplete, Render, and Upload.

After understanding the concepts of ASP.NET 3.5, let's move on to explore the Microsoft Visual Studio 2008 IDE. It provides a single environment for developing all types of .NET applications and provides many options and rich features to create interactive Web applications by using ASP.NET 3.5.

Overview of Visual Studio 2008

Microsoft Visual Studio is an Integrated Development Environment (IDE) provided by Microsoft to create and develop Windows-based, Web-based, console-based, and mobile-based applications in .NET Framework. These applications might be created by using different languages, such as Visual Basic, C#, and Visual C++. You can use Notepad as a text editor to create these applications. For this, you need a compiler that could convert the programming language into machine language. Now, you might be wondering that if a Notepad can create all these applications, why do we need Visual Studio as an editor. Building applications by using Notepad takes more time than taken while building the same applications in Visual Studio. This is because Microsoft Visual Studio provides various development tools to design and create the applications, as compared to a Notepad. For example, you need to memorize all the language syntax if you are using a Notepad. In case of Visual Studio, you do not need to remember the syntax; you need to type the first few letters of a command and Visual Studio would prompt you with the available options for the command using IntelliSense.

The following are some of the benefits of using Visual Studio 2008:

- Helps in minimizing the development time
- Simplifies the process of testing applications
- Provides different toolsets for integrating graphic designers into the overall development process
- Supports multiple versions of the .NET Framework
- Enhances data retrieval and data binding
- Inspects code to find sections of code that need refactoring
- Provides support for Web, mobile, client, Vista, and Office application development
- Integrates with the .NET Framework 3.5

Now, let's explore Visual Studio 2008 and its new features.

New Features in Visual Studio 2008

Visual Studio 2008 is an integrated development environment designed for writing, compiling, and debugging the code in a much easier way than Visual Studio 2005. It contains a complete set of development tools for building ASP.NET Web applications, Web services, desktop applications, and mobile applications.

The followings are some of the new features and enhancements made in Visual Studio 2008:

- Support for multi-targeting
- Support for ASP.NET Asynchronous JavaScript and XML (AJAX) and JavaScript IntelliSense
- Support for Language Integrated Query (LINQ)
- Improved deployment
- Support for client application services
- Support for reporting applications

Support for Multi-Targeting

In the previous versions of Visual Studio, you were required to install the latest .NET Framework version. For example, for using Visual Studio 2005, you also had to install .NET Framework 2.0. Installing a new .NET Framework version may provide additional functionalities to your project but at the same time, it may also add new .NET Framework dependencies. These dependencies will prevent your project from running on systems where they have already been run. This may create a problem in case your project needs backward compatibility.

However, Visual Studio 2008 resolves this issue by letting you target the specific .NET Framework version you want for your project by selecting it in the Framework Version box in the upper-right corner of the New Project dialog box, as shown in Figure 14.3:

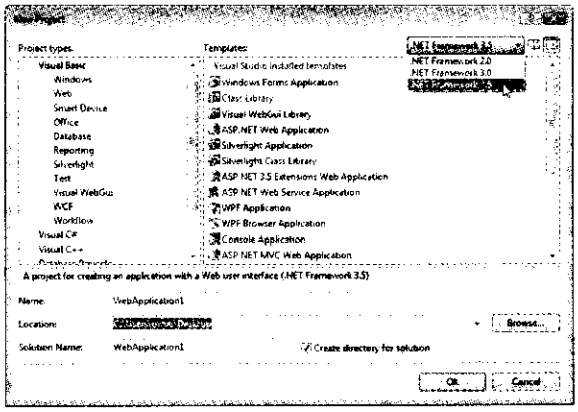


Figure 14.3: New Project Dialog Box

When you upgrade Visual Studio 2005 projects to Visual Studio 2008, these projects still continue to target .NET Framework 2.0. You can change the target from .NET Framework 2.0 to either .NET Framework 3.0 or .NET Framework 3.5 in order to take advantage of new features available in the newer versions of.NET Framework.

Support for ASP.NET AJAX and JavaScript Intellisense

One of the new features that developers find interesting in Visual Studio 2008 is its in-built IntelliSense support for JavaScript and ASP.NET AJAX scripting. This has made building applications using JavaScript and AJAX considerably easier. If you double-click the HTML Button control in the Design view, this will automatically create a `Click` event for the button and a basic skeleton for the JavaScript function. Figure 14.4 displays built-in JavaScript IntelliSense:

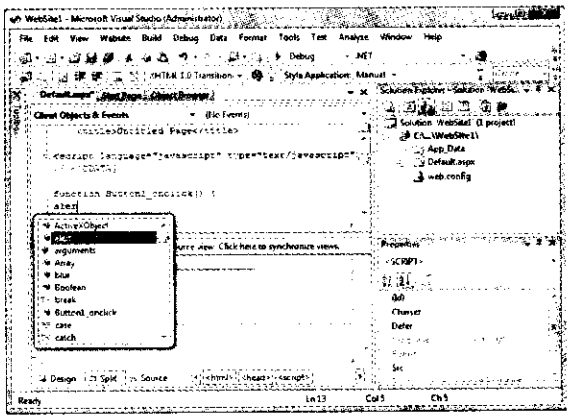


Figure 14.4: Displaying built-in JavaScript IntelliSense

JavaScript IntelliSense is also supported for referring external script files that use eXtensible Markup Language (XML) code comments. These code comments describe the summary, parameter, and return details of the client script. ASP.NET AJAX also uses XML code comments for providing IntelliSense of ASP.NET AJAX types and members.

Support for Language Integrated Query

LINQ, one of the new features of Visual Studio 2008, extends powerful query capabilities into the language syntax. It adds native data querying capabilities to .NET languages using syntax similar to SQL. It can support any kind of data source. It also introduces a set of new assemblies for enabling the use of LINQ with collections, SQL databases, and XML documents.

Improved Deployment

Deployment in Visual Studio 2008 has improved with the enhancement of Windows Installer deployment. It is a deployment technology which enables you to create installer packages to be distributed to users. In this case, for installing the application, the user needs to run the setup file which opens a wizard to install the application. Windows Installer deployment has the following enhancements and has been updated for Windows Vista and for the latest .NET Framework versions:

- Windows Installer has been updated to ensure that it gets installed on Windows Vista smoothly even when it is running under UAC
- .NET Framework Launch Condition helps in supporting targeting of applications for the new .NET Framework 3.0 and 3.5 versions

Support for Client Application Services

Client application services are newly introduced in .NET Framework 3.5. These services enable Windows-based applications which include both Windows Forms and WPF applications to easily access the ASP.NET login, roles, and profile services. These services also allow you to authenticate users and retrieve user roles and application settings from a shared server. For example, by using these services, you can authenticate a user, determine the roles of these authenticated users, and finally store and access application settings located on the server.

You can include client application services by configuring client service providers in the Visual Studio Project Designer or in your application configuration file. These providers connect to the Web extensibility model for accessing the Web services through existing .NET Framework login, roles, and settings APIs. These services also support occasional connectivity by storing and retrieving the user data from the local data cache when the application goes offline.

Support for Reporting Applications

A reporting application is used to prepare reports. Visual Studio 2008 supports many new reporting features and enhancements which are as follows:

- **New Report Projects**—Visual Studio 2008 provides two new project templates for creating reporting applications. The first template is the Reports Application template, which is available on the New Project dialog box and the second template is the ASP.NET Reports Web Site template, which is available on the New Web Site dialog box. When you create a new Reports Application project, Visual Studio provides a report (`.rdlc`) and a form (`.vb/.cs`) with a ReportViewer control bound to the report.
- **Report Wizard**—Visual Studio 2008 provides a Report Wizard, which guides you through a number of steps to create a basic report. After the wizard is finished, you can make enhancements to the report by using Report Designer.
- **Expression Editor Enhancement**—The Expression Editor provides sample expressions that you can directly use in your report or customize it according to your requirement.
- **ReportViewer Printing**—When you configure the ASP.NET ReportViewer control for local processing, the RSClientPrint control becomes available. Using this, you can print reports that have been processed by the ReportViewer control and are not dependent on a report server.
- **PDF Compression**—The ReportViewer controls can now compress reports that have been rendered or exported to the PDF format.

Now, let's discuss Visual Studio 2008 Service Pack 1 (SP1) and .NET Framework 3.5 SP1 as both are required to work with all the new technologies introduced in ASP.NET 3.5. The best way to grasp the Visual Studio 2008 IDE is to start working on a sample Web application and a sample website. Now, let's start with sample Web application first.

Exploring a Sample ASP.NET 3.5 Web Application

In ASP.NET, you can create both Web applications and websites. Now, let's explore a Web application by following these steps:

1. Open Microsoft Visual Studio 2008.
2. Click the File→New→ Project option, as shown in Figure 14.5:

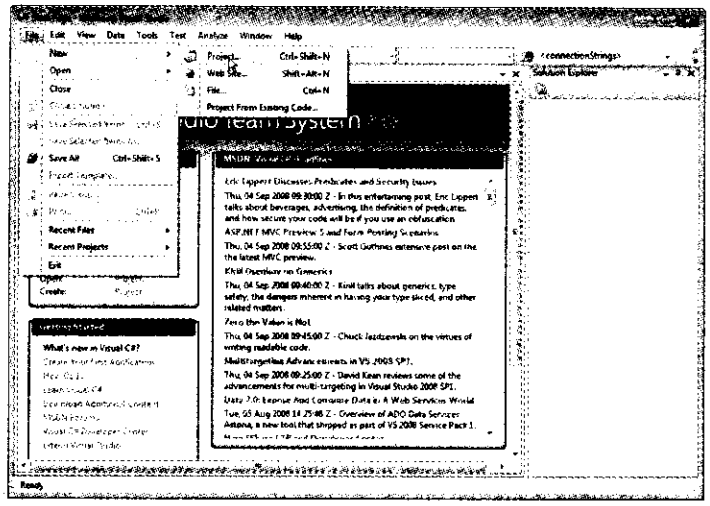


Figure 14.5: Start Page of ASP.NET

The `New Project` dialog box appears (Figure 14.6).

3. Select the `project type` as `Web` under the `Project types` pane and `ASP.NET Web Application` under the `Templates` pane (Figure 14.6).
4. Click the `OK` button, as shown in Figure 14.6:

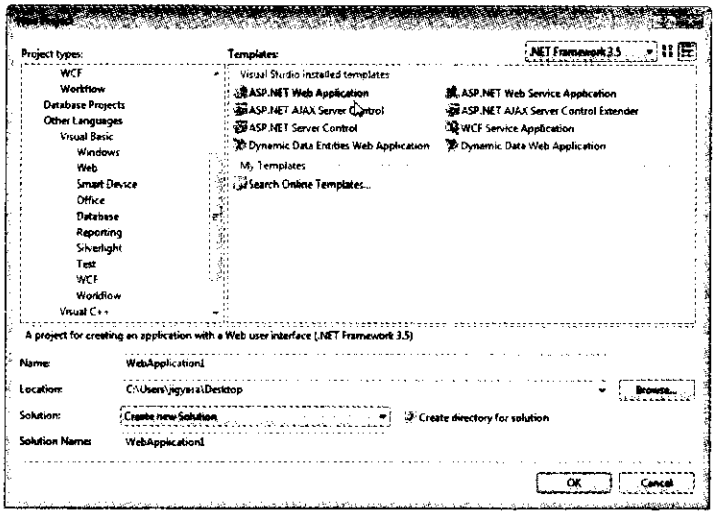


Figure 14.6: New Project Dialog Box

It opens the Visual Studio 2008 IDE with the `Default.aspx` file, also known as the source file, as shown in Figure 14.7:

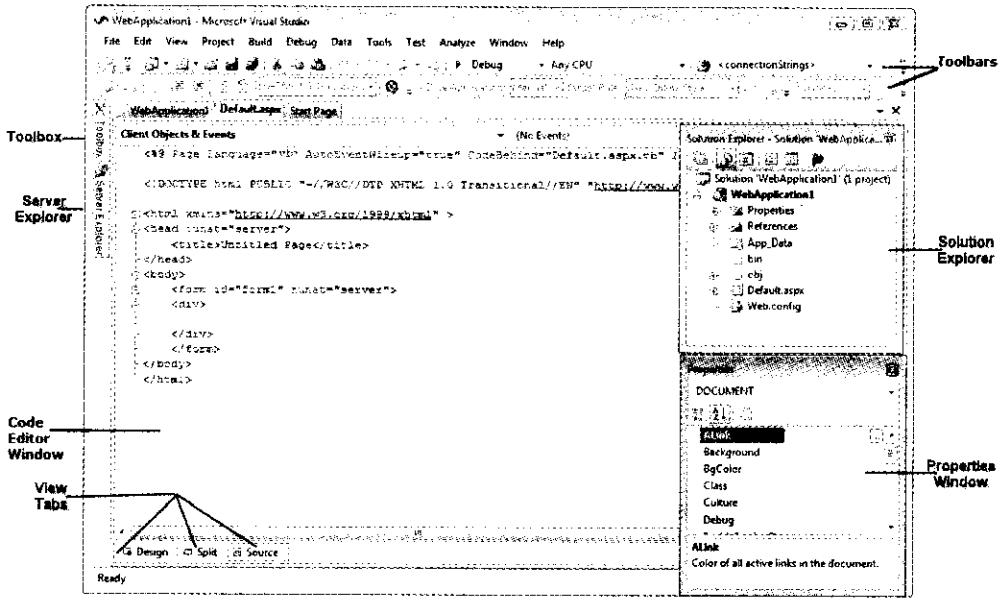


Figure 14.7: Visual Studio 2008 IDE

- **Toolbox**—Contains a number of tabs and each tab contains a list of controls that can be dragged and used on a Web Form. The **Toolbox** is docked on the left side of the Visual Studio 2008 IDE; however, if it is not visible, you can access the **Toolbox** by selecting **View→Toolbox** from the menu bar or by pressing the **CTRL+ALT+X** keys together to open it in the Visual Studio 2008 IDE.
- **Server Explorer**—Displays a list of available servers from which data can be accessed through the network. The **Server Explorer** window is viewed by selecting **View→Server Explorer** from the menu bar or by pressing the **CTRL+ALT+S** key combination from the keyboard.
- **Code Editor Window**—Helps you to recognize keywords and identifiers of the programming language. It also checks syntax errors, incorrect initialization of variables, invalid declarations, and unrecognized calling of procedures by highlighting them. You can access the Code editor window by double-clicking the control or the Web Form or pressing the **F7** key in the design mode. Code editor window also provides **IntelliSense** feature, which provides you a list of options that help to write code easily and automatically completes the code you are typing.
- **Toolbars**—Provides developers with an easy access to the features of the Visual Studio 2008 IDE. The Visual Studio 2008 IDE provides different toolbars for each category of tasks needed in the process of developing an application. The **Standard** toolbar is the default visible toolbar, the other toolbars such as **Formatting** and **Build** can be viewed by using the **Customize** option available on the **Tools menu**. Users can customize this toolbar by adding and removing buttons from it.
- **Solution Explorer**—Provides a conceptual overview of a project, which makes managing project elements such as class and Web Forms, easier. The project appears in a tree view format, showing nodes and child nodes. You can access the Solution Explorer window by selecting **View→Solution Explorer** from the menu bar or by pressing the **CTRL+ALT+L** key combination from the keyboard.
- **Properties Window**—Makes the task of setting properties of the controls and the Web Forms very easy. The Properties window is viewed by selecting **View→Properties Window** from the menu bar or pressing the F4 key from the keyboard.
- **View Tabs**—of Used to see different layout Web page. Design tab shows the design of Web Form, similarly **Source** tab shows the source code of the Web Form, whereas, **Split** tab shows both design and source code partially.

After exploring the Visual Studio 2008 IDE in brief, let's move further and see how to create a Web application.

NOTE

Throughout this book, we are creating ASP.NET websites and referring them as Web applications interchangeably.

Creating a Sample ASP.NET 3.5 Website

Let's create a website named SampleWebsiteVB. You can find the code for the SampleWebsiteVB application in the Code\ASP.NET\Chapter 14\SampleWebsiteVB folder on the CD.

To do so, let's follow these steps:

1. Open Microsoft Visual Studio 2008.
2. Click the File→New→Web Site option, as shown in Figure 14.8:

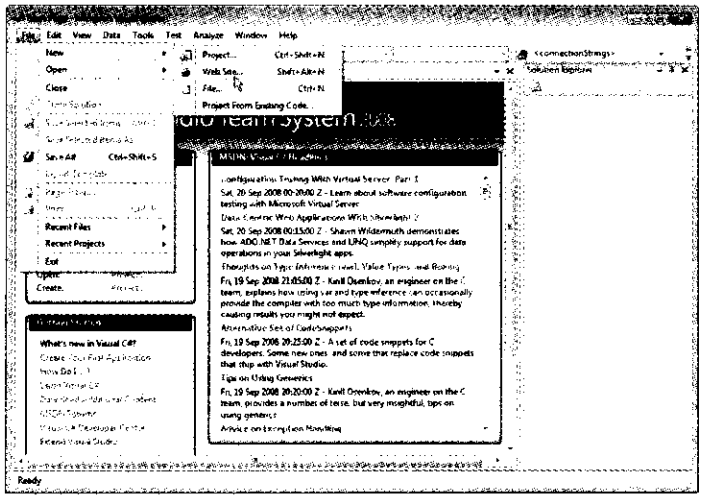


Figure 14.8: First Step to Create a Website

The `New Web Site` dialog box appears (Figure 14.9).

3. Select `ASP.NET Web Site` in the Templates pane, select the language as Visual Basic and set the location to create the website, and click the `OK` button, as shown in Figure 14.9:

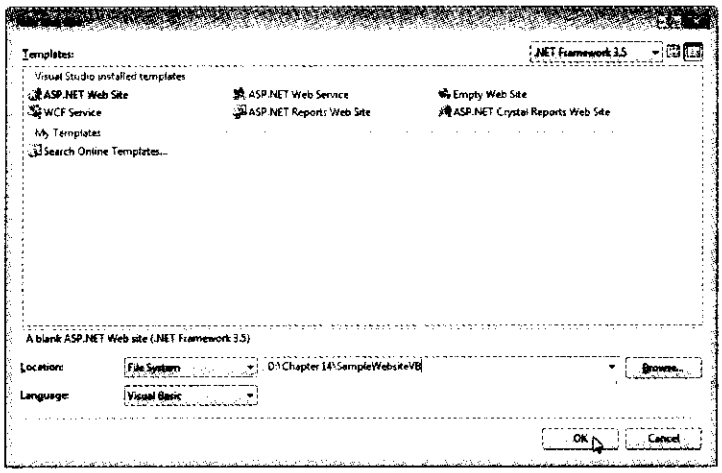


Figure 14.9: New Web Site Dialog Box

It opens the Visual Studio 2008 IDE with the `Default.aspx` file, also known as the source file, as shown in Figure 14.10:

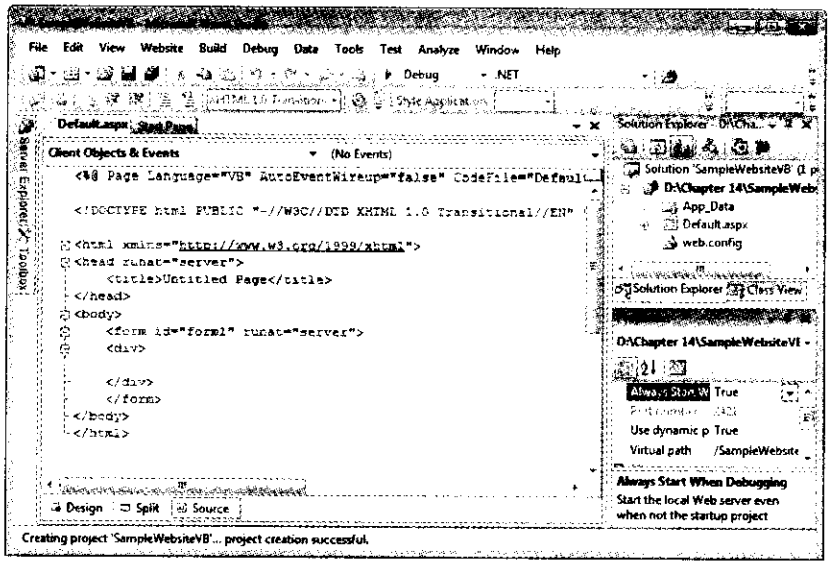


Figure 14.10: Visual Studio 2008 IDE

4. Now, open the code-behind file by pressing the F7 key. The default code-behind file (Default.aspx.vb) window appears, as shown in Figure 14.11:

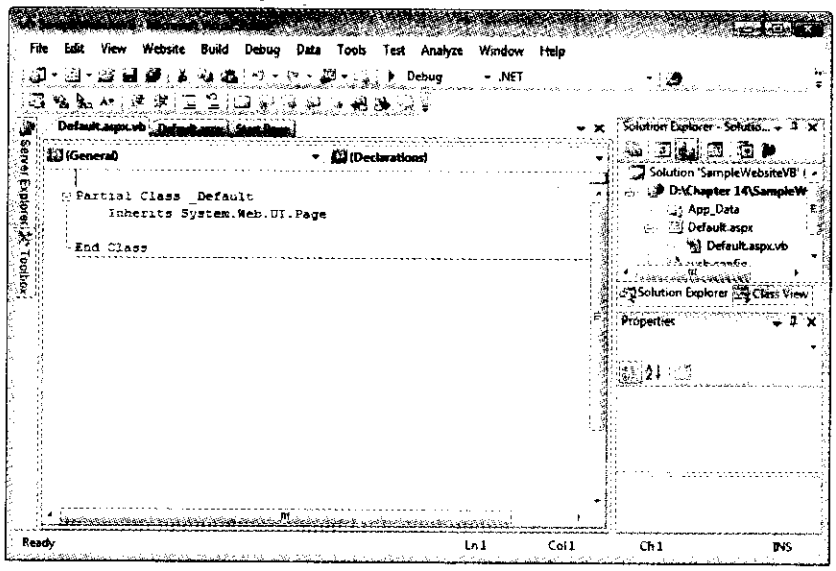


Figure 14.11: Code-Behind File

To load an event, follow these steps:

1. Select the form1 object from the drop-down list on the Default.aspx.vb file, as shown in Figure 14.12:

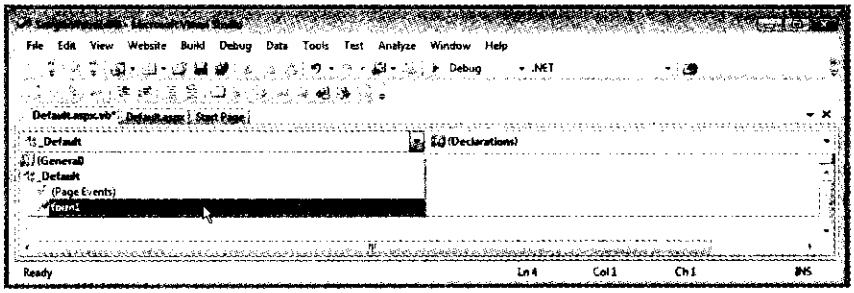


Figure 14.12: Drop-Down List Showing the form1 Object

2. Now, select the Load event corresponding to the form1 object from the drop-down list, as shown in Figure 14.13:

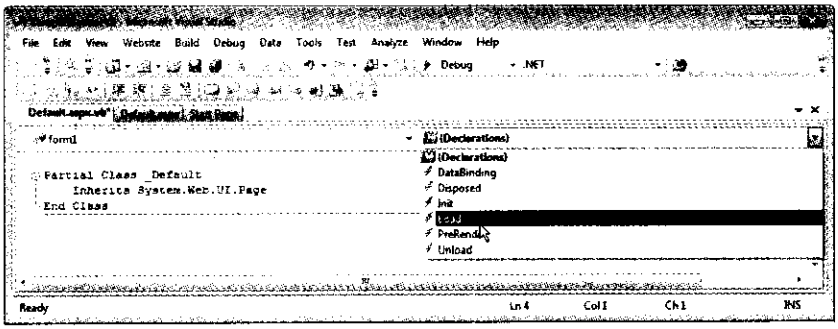


Figure 14.13: Drop-Down List Showing the Events List

After adding Load event, the code-behind file (Default.aspx.vb) looks, as shown in Figure 14.14:

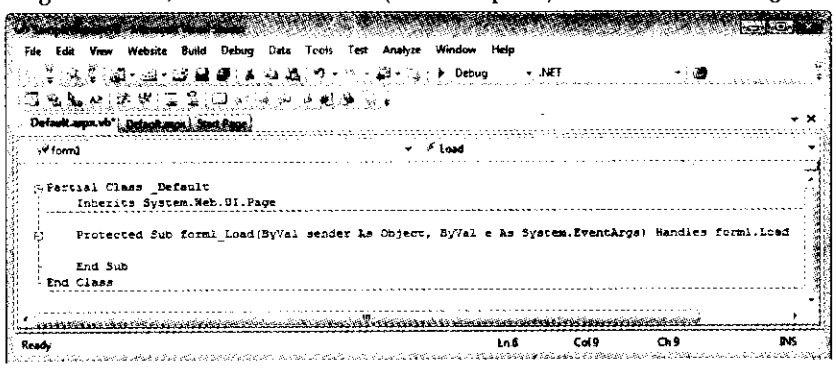


Figure 14.14: Code-Behind File

You can find the code for the `Default.aspx` page in Listing 14.1:

Listing 14.1: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Sample Website</title>
</head>
<body>
  <form id="form1" runat="server">
```

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

3. Now, in order to display the text Hello World, type the highlighted code in the code-behind files. Now, add the code in the code-behind file of the `Default.aspx` page as shown in Listing 14.2:

Listing 14.2: Showing the code for the Code-Behind File of the `Default.aspx` Page

```
Partial Class _Default
  Inherits System.Web.UI.Page
  Protected Sub form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
  form1.Load
      [redacted]
  End Sub
End Class
```

4. Press the `F5` key to get the output, as shown in Figure 14.15:

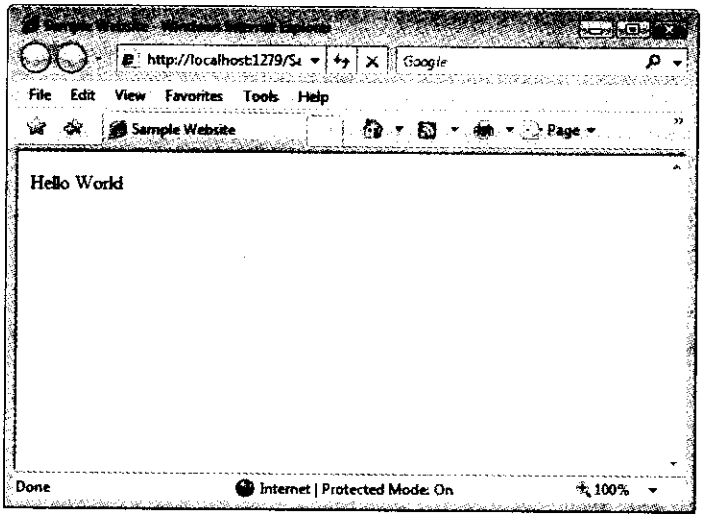


Figure 14.15: Output of the Application

You can see the output as Hello World in the Figure 14.15.

Summary

In this chapter, we have covered new features in ASP.NET 3.5, such as ASP.NET AJAX support, ListView control, and DataPager control, ASP.NET life cycle, and overview of Visual Studio 2008. It also discusses new features of Visual Studio 2008 such as support for multi-targeting, and support for LINQ. WE have also explored and created a sample ASP.NET 3.5 Web application.

In the next chapter, we discuss about how to develop a Web application in ASP.NET 3.5.

Quick Revise

Q1. Explain the different versions of .NET Framework?

Ans: The different versions of .NET Framework are:

1. **.NET Framework 1.0** —.NET Framework 1.0 is the first version of .NET Framework and was released by Microsoft on February 13, 2002. It is a part of Visual Studio .NET 2002, which is the first version of Visual Studio .NET.

2. **.NET Framework 1.1** —The first major upgrade of .NET Framework, .NET Framework 1.1, was released on April 3, 2003. It is a part of Visual Studio .NET 2003, which is the second version of Visual Studio .NET. In contrast to .NET Framework 1.0, NET Framework 1.1 has in-built support for mobile ASP.NET controls, Open Database Connectivity (ODBC), and Oracle databases. It also has support for Internet Protocol version 6 (IPv6).
3. **.NET Framework 2.0**—The second major upgrade of .NET Framework,.NET Framework 2.0, was released on January 22, 2006. It is a part of Visual Studio 2005 and Microsoft SQL Server 2005. .NET Framework 2.0 is the latest version of .NET Framework that has support for Windows 2000. .NET Framework 2.0 has many changes and enhancements as compared to .NET Framework 1.1. It has a number of Application Programming Interface (API) changes. It contains many new ASP.NET Web controls and data controls. It also contains new personalization features for ASP.NET, for example, support for themes, skins, and WebParts.
4. **.NET Framework 3.0**—The third major upgrade of .NET Framework, .NET Framework 3.0, was released on November 21, 2006. It contains a set of managed code APIs that form an integral part of Windows Vista and Windows Server 2008. .NET Framework 3.0 uses the same version of CLR that was incorporated with .NET Framework 2.0. .NET Framework 3.0 includes the following four new components:
   - Windows Presentation Foundation
   - Windows Communication Foundation
   - Windows Workflow Foundation
   - Windows CardSpace (WCS)
5. **.NET Framework 3.5**—The fourth major upgrade of.NET Framework, .NET Framework 3.5, was released on November 19, 2007. Similar to .NET Framework 3.0, .NET Framework 3.5 also uses the CLR version 2.0. .NET Framework 3.5 also installs.NET Framework 2.0 Service Pack (SP) 1, .NET Framework 2.0 SP2 (with 3.5 SP1), and .NET Framework 3.0 SP1, which includes methods and properties that are required for the .NET Framework 3.5 features, such as Language-Integrated Query (LINQ). In addition to LINQ, .NET Framework 3.5 includes many other new features, such as extension methods, lambda expressions, anonymous types, and built-in support for ASP.NET AJAX.

Q2. Define Merge tool?

Ans: Merge tool (`Aspnet_merge.exe`) allows you to combine and manage assemblies created by the ASP.NET precompilation tool (`Aspnet_compiler.exe`).

Q3. What are the different stages of an ASP.NET Web Page?

Ans: The different stages of an ASP.NET Web page are as follows:

1. Page request
2. Start
3. Page initialization
4. Load
5. Validation
6. Postback event handling
7. Rendering
8. Unload

Q4. Which of these is not a stage of an ASP.NET Web Page?

1. **Load**
2. **Postback event handling**
3. **Rendering**
4. **Pre-compilation**

Ans: Pre-compilation

Q5. The Framework Version box appears in..... dialog box.

1. **New Project**
2. **Add New Item**
3. **Add Existing Item**
4. **Properties**

Ans: New Project

Q6. The.....window helps you to recognize keywords and identifiers of the programming language.

1. **Properties**
2. **Code Editor**
3. **Add Existing Item**
4. **New Project**

Ans: Code Editor

Q7. Which shortcut key combination is used to open Solution Explorer?

Ans: CTRL+ALT+L

Q8. What are the different stages of the ASP.NET application life cycle with integrated mode in IIS 7.0?

Ans: Different stages of the ASP.NET application life cycle with integrated mode in IIS 7.0 are as follows:

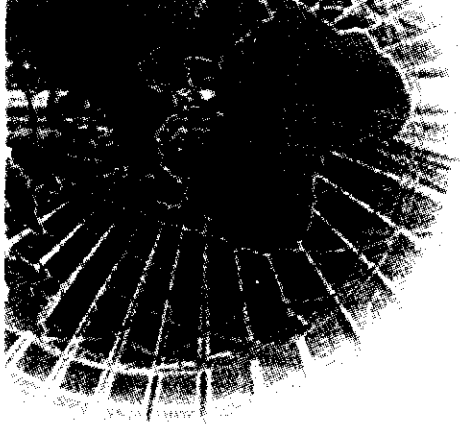
1. A request is made for accessing application resource.
2. The unified pipeline receives the first request for the application.
3. Response objects are created for each request.
4. An HttpApplication object is assigned to the request.
5. The request is processed by the HttpApplication pipeline.

Q9. Which tabs are used to see different layout of a Web page?

Ans: View tabs

Q10. What do you mean by IntelliSense?

Ans: Visual Studio Code Editor supports the feature of IntelliSense, which helps you to write accurate code. As you start typing the code the methods, properties, and events of the .NET Framework Class Library are automatically offered as a sorted list with the help of IntelliSense feature. This list, which automatically opens in an IntelliSense, shows all the members that are valid for using the code. When the Web developer selects an option from the IntelliSense, it is included in the line of code with accurate spelling and case. While writing a code, if there is any error in the code, the IntelliSense displays a squiggle under the code or the keyword that signifies an induced error.



15

Developing a Web Application

Earlier, we used to work only on desktop applications that were accessible to a single computer, where they were installed, and could not be accessed by other users. They could not be hosted on the Internet either. To overcome these shortcomings, Web applications came into existence. A Web application implies an application hosted on the Internet, which can be accessed through a Web browser, such as Internet Explorer or Mozilla Firefox. Now-a-days, business through the Internet has gained immense popularity due to various features and benefits, such as the ease of making transactions from home or office. Therefore, it becomes quite crucial for companies to make their presence felt over the Internet so that the services provided by them can be easily accessible to their respective customers. To do so, companies develop their own Web applications; which not only helps them reaching out to prospective clients, but also helps them in conducting businesses online. A very good example of such a scenario is an online shopping website, where you can perform transactions related to buying or selling a product. Another feature that makes Web applications more popular is the convenience of updating and maintaining them without working individually on potentially thousands of client computers. Since Web applications reside on Web servers, updating and maintaining them at a central location (Web server) can serve the purpose of updating and maintaining them over thousands of client computers.

ASP.NET provides an efficient development environment for developing Web applications. To create a Web application, you need to be aware of the following concepts:

- Specifying a Location for a Web Application
- File Types in ASP.NET 3.5
- ASP.NET 3.5 Pages
- Understanding ASP.NET 3.5 Page Directives
- Working with Server Controls
- Understanding ASP.NET 3.5 Provider Model
- ASP.NET 3.5 Coding Models
- Implementing Code Sharing
- Compilation in ASP.NET 3.5

Specifying a Location for a Web Application

After a Web application is developed, it needs to be stored in a particular location in a system. We can then open the application from this location to work on it. Visual Studio allows you to create a Web application with a virtual directory mapped to the Internet Information Services (IIS) server or a stand-alone application outside the boundaries of the IIS server. Visual Studio 2008 allows you to store Web applications in the following three locations:

- File System
- HyperText Transfer Protocol (HTTP)
- File Transfer Protocol (FTP)

File System

Visual Studio allows you to keep the files of a Web application or website in a folder on the local hard drive or in a shared location on the local area network. These websites, known as File System websites, are not associated with any IIS application, unless you create an IIS virtual directory. You can run these websites by using the built-in Web server, which executes at runtime. This Web server allows you to run Web applications, even if the IIS server is not configured on your machine. You can also verify the existence of the built-in Web Server by selecting the New→Web Site option in the Visual Studio IDE. In Windows Vista, the default location for your Web applications is `C:\Users\Jigyasa\Documents\Visual Studio 2008\WebSites`, as shown in Figure 15.1:

NOTE

Here, Jigyasa is the Windows user name, so it will vary from system to system.

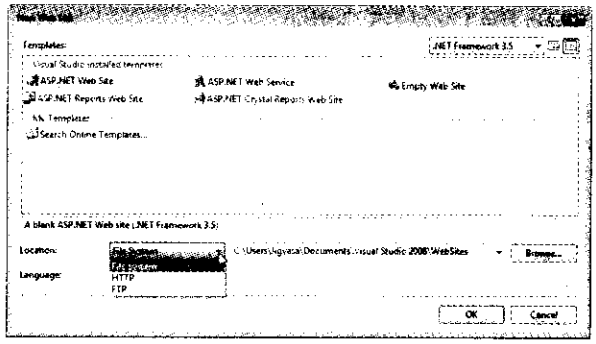


Figure 15.1: Default Path of a Website

However, you can also store your Web application at any other location. When you are using a Web server, the location of your Web application is not constrained to any particular folder.

In the figure, you can also see the remaining location options, HTTP and FTP.

NOTE

You can change the default path of the Web application according to the need of your project by clicking the Browse button.

Now, open the Choose Location dialog box by clicking the Browse button; the File System tab is selected by default, as shown in Figure 15.2:

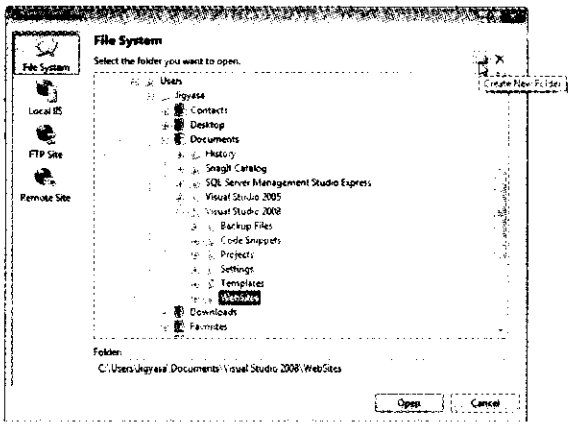


Figure 15.2: Create New Folder Option

NOTE

You can also change the default path of the Web application by creating a new folder in the File System tab (Figure 15.2).

HTTP

Visual Studio allows you to develop and store your Web application on the IIS server. For that, Visual Studio provides an HTTP option for developing and storing Web applications on the IIS server. This option can also be used to develop and store your Web application on the following servers:

- Local IIS Server
- Remote IIS Server

Now, let's discuss these options in detail.

Storing Web Applications Using the Local IIS Option

A local IIS website is an IIS Web application. Visual Studio communicates with the website by using the HTTP protocol. In the Choose Location dialog box, select the Local IIS option to store the website files on the local IIS server. When you select this option, the `Local Web Servers` option is displayed in the Choose Location window, as shown in Figure 15.3:

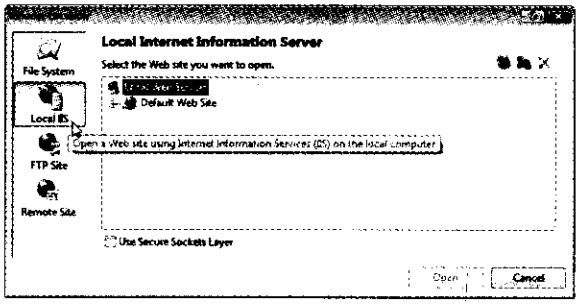


Figure 15.3: Local IIS Option Selected

This `Local Web Servers` option shows a list of all the virtual application roots on your machine.

NOTE

If you want to see the local IIS instance, right-click the Visual Studio 2008 in the start menu and choose the option Run as administrator from the context menu. The local IIS instance will appear, as shown in Figure 15.4.

The following steps are performed to create a new virtual directory for your Web application:

1. Select the Default Web Site option. It will enable two options: Create New Web Application and Create New Virtual Directory. These options are available at the right-hand corner of the dialog box, as shown in Figure 15.4:

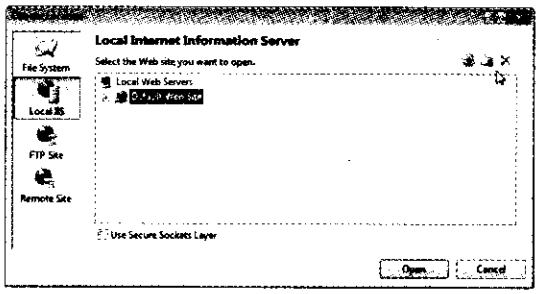


Figure 15.4: Highlighted Default Web Site Option

2. Create a new virtual directory, say, Test, using the Create New Virtual Directory option, as shown in Figure 15.5:

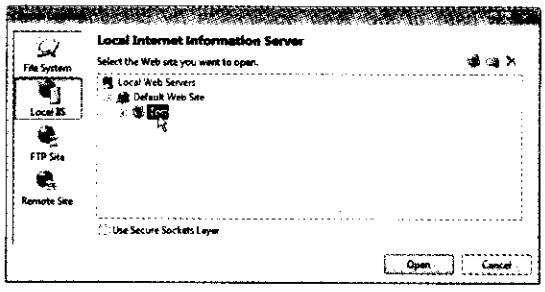


Figure 15.5: New Virtual Directory Named Test

NOTE

To delete an existing virtual directory, select the directory that you want to delete and click the Delete option.

3. Now, click the Open button. You can see this directory (Test) added to the New Web Site dialog box, as shown in Figure 15.6:

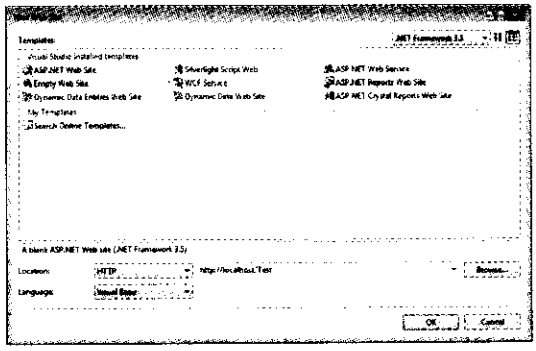


Figure 15.6: Test Directory Added to New Web Site Dialog Box

4. Click the OK button, as shown in Figure 15.6. Visual Studio then creates a new Web application.

However, instead of depending on the built-in Web server, here your application will use the IIS server. When you invoke your Web application, the URL that appears consists of something similar to `http://localhost/Test/Default.aspx`, which shows that it is using the IIS server.

NOTE

To register the IIS server in ASP.NET 3.5, run the Visual Studio 2008 Command Prompt as an administrator and use the aspnet_regiis –i syntax. For more details on the aspnet_regiis.exe file, refer to Chapter 24, Managing Web Applications.

Storing Web Applications Using the Remote Site Option

The Remote Site option stores files on a remote server that is accessible over a local network. A remote website is an IIS Web application linked with a copy of the IIS server that is running on another computer. Visual Studio communicates with the website by using the HTTP protocol to host the site on the remote IIS server. Enter the location of website in the Web site location textbox and click the Open button, as shown in Figure 15.7:

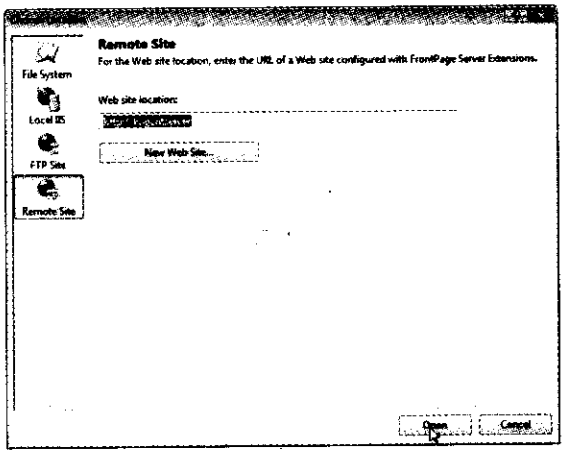


Figure 15.7: Remote Site Option

You can see the Remote Site option using the HTTP location, as shown in Figure 15.8:

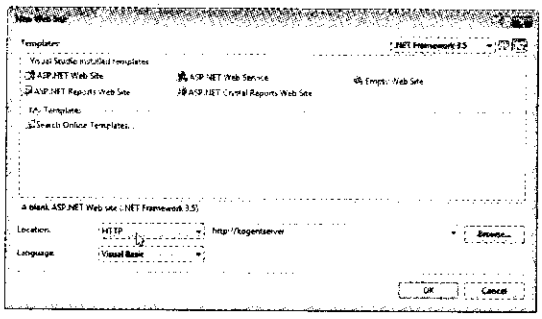


Figure 15.8: Remote Site Using HTTP Location

FTP

FTP enables you to actually store and even code your applications while they reside on a server somewhere else in your enterprise. For that, you must have the credentials to attach to the FTP server and permissions to read and write to the FTP location. You can also use the FTP features, such as exchanging and manipulating files over any Transfer Control Protocol (TCP) based computer network, to work on different locations within the same server. To create your application on the remote server using FTP, you have to provide the Server name, Port to use, Directory, and other credentials, if required, as shown in Figure 15.9:

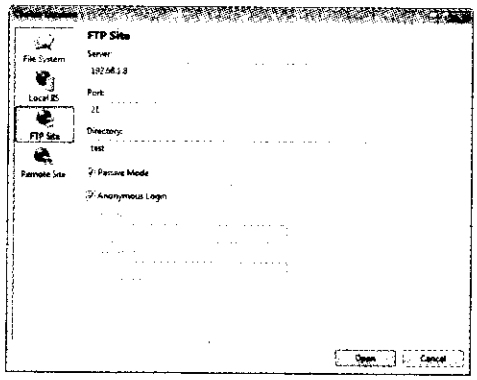


Figure 15.9: FTP Site Option

If you provide correct information, Visual Studio accesses the remote server and creates the appropriate files required for your application, as shown in Figure 15.10:

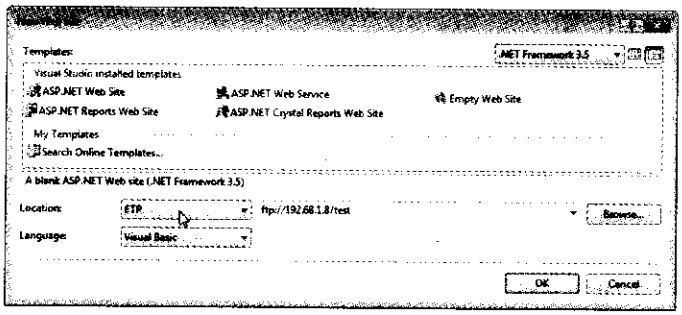


Figure 15.10: FTP Site Option Using FTP Location

After understanding the various options for storing Web applications (websites) at different locations, let's understand the file types supported by ASP.NET 3.5.

NOTE

While creating a ASP.NET website on FTP location, you need ensure that the virtual directory on FTP server is mapped to virtual directory on IIS web server.

File Types in ASP.NET 3.5

Web applications contain a number of file types—some are supported and managed by ASP.NET 3.5, and the rest are supported and managed by the IIS server. Most of the ASP.NET 3.5 file types can be automatically generated using the **Add New Item** dialog box in Visual Studio 2008.

File types are mapped to Web applications using application mappings. For example, if you double-click a `.txt` file in `Windows Explorer`, it will probably open a `Notepad` because in Windows, `.txt` file types are mapped to the `Notepad.exe` file by default. In Web applications, file types are mapped to application extensions in the IIS server. Table 15.1 lists the file types managed by ASP.NET:

Table 15.1: File Types Managed by ASP.NET

File Type	Location	Description
`.asax`	Application root	It is a `Global.asax` file containing code that derives from the `HttpApplication` class. This file represents the Web application and contains optional methods that run at the start or end of the application's lifetime.
`.ascx`	Application root or a subdirectory	It is a Web user control file that defines a custom, reusable control.
`.ashx`	Application root or a subdirectory	It is a generic handler file that contains code to implement the `IHttpHandler` interface.
`.asmx`	Application root or a subdirectory	It is an XML Web services file containing classes and methods that are available to other Web applications through Simple Object Access Protocol (SOAP).
`.aspx`	Application root or a subdirectory	It is an ASP.NET Web forms file (page) containing Web controls, and presentation and business logic.
`.axd`	Application root	It is a handler file used to manage website administration requests, typically `Trace.axd`.
`.browser`	`App_Browsers` subdirectory	It is a browser definition file used to identify the features of client browsers.
`.cd`	Application root or a subdirectory	It is a class diagram file.
`.compile`	Bin subdirectory	It is a precompiled stub file that points to an assembly representing a compiled website file. Executable file types (`.aspx, ascx, .master`, theme files) are precompiled and put in the Bin subdirectory.
`.cs, .jsl, .vb`	`App_Code` subdirectory, or in the case of a code-behind file for an ASP.NET page, in the same directory as the Web page	It is a class source-code file that is compiled at runtime. A class can be an HTTP Module, an HTTP Handler, a code-behind file for an ASP.NET page, or a stand-alone class file containing application logic.
`.csproj, .vbproj, .vjsproj`	Visual Studio project directory	It is a project file for a Visual Studio client-application project.
`.disco, .vsdisco`	`App_WebReferences` subdirectory	It is an XML Web services discovery file, which is used to help locate available Web services.
`.dsdgm, .dsprototype`	Application root or a subdirectory	It is a distributed service diagram (DSD) file that can be added to any Visual Studio solution that provides or

Table 15.1: File Types Managed by ASP.NET

File Type	Location	Description
		consumes Web services to reverse-engineer an architectural view of the Web service interactions.
.dll	Bin subdirectory	It is a compiled class library file (assembly). Note that instead of placing compiled assemblies in the Bin subdirectory, you can put source code for classes in the App_Code subdirectory.
.licx, .webinfo	Application root or a subdirectory	It is a license file. Licensing helps authors protect intellectual property by checking whether the user is authorized to use the control.
.master	Application root or subdirectory	It is a master page that defines the layout for other Web pages in a Web application.
.mdb, .ldb	App_Data subdirectory	It is an Access database file.
.mdf	App_Data subdirectory	It is a SQL database file to use with SQL Server Express.
.msgx, .svc	Application root or a subdirectory	It is an Indigo Messaging Framework (MFx) service file.
.rem	Application root or a subdirectory	It is a remoting handler file.
.resources, .resx	App_GlobalResources or App_LocalResources subdirectory	It is a resource file containing resource strings that refer to images, localizable text, or other data.
.sdm, .sdmDocument	Application root or a subdirectory	It is a system definition model (SDM) file.
.sitemap	Application root	It is a site-map file containing the structure of the website. ASP.NET comes with a default site-map provider that uses site-map files to easily display a navigational control in a Web page.
.skin	App_Themes subdirectory	It is a skin file containing property settings to apply to Web controls for consistent formatting.
.sln	Visual Web Developer project directory	It is a solution file for a Visual Web Developer project.
.soap	Application root or a subdirectory	It is a SOAP extension file.

NOTE

File types that are managed by ASP.NET are mapped to the aspnet_isapi.dll handler in the IIS server.

Table 15.2 lists the file types managed by the IIS server:

Table 15.2: File Types Managed by the IIS server

File Type	Location	Description
.asa	Application root	It is a Global.asa file containing optional methods that run at the start or end of the ASP session or application lifetime
.asp	Application root or a subdirectory	It is an ASP Web page containing @ directives and script code, which use the ASP built-in objects
.cdx	App_Data subdirectory	It is a compound index file structure file for Visual FoxPro
.cer	Application root or a	It is a certificate file used to authenticate a website

Table 15.2: File Types Managed by the IIS server

File Type	Location	Description
	subdirectory	
`.idc`	Application root or a subdirectory	It is an Internet Database Connector file mapped to `httpodbc.dll`
`.shtm, .shtml, .stm`	Application root or a subdirectory	It is mapped to `ssinc.dll`

NOTE

File types that are managed by ASP.NET are usually mapped to the asp.dll handler in the IIS server.

After getting familiar with the file types used to create ASP.NET website applications, let's move on to explore ASP.NET 3.5 Web pages, which are required to create such applications.

Exploring ASP.NET 3.5 Web Pages

ASP.NET 3.5 Web pages are files with the `.aspx` extension, and contain the code to implement a Web application. Now, the question is: where are these pages stored? Well, these Web pages are stored on the IIS virtual directory. When a client request is received by the IIS server, the requested page is searched for and information is sent back in a `Response` object to render the HTML markup on to the client browser. The rendered HTML markup defines everything about a Web page, such as its content, layout, and appearance.

Generally, a Web application developed in any programming language is first compiled and then converted into its intermediate code, also known as Intermediate Language (IL). This is true for an ASP.NET Web application also. This ASP.NET intermediate code is platform-independent and is obtained by compiling the source code written in any .NET compatible language. Some of the .NET compatible languages for ASP.NET applications are Visual Basic, and C#.

When the request for a Web page is received for the first time, its corresponding `.aspx` file is compiled to generate the class file. The class file created is then used to process the request. However, when a request for the same Web page is received subsequently, its class file is referenced directly to process the request. This enhances the performance and reduces the response time of processing a request.

The code used to develop an ASP.NET Web page is similar to the code used in the HTML file. For instance, the following code shows a simple ASP.NET Web page displaying a welcome page on the browser. You can find this code in the Code\ASP.NET\Chapter 15\WelcomePage folder on the CD. Listing 15.1 shows the code for the Default.aspx page:

Listing 15.1: Showing the Code for the Default.aspx Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
      <title>Untitled Page</title>
  </head>
  <body>
      <form id="form1" runat="server">
          <div>
          <h1> Welcome to the World of ASP.NET 3.5</h1></div>
      </form>
  </body>
</html>
```

In the preceding listing, you can see that the first line of the ASP.NET page is enclosed between the <% and %> tags. These blocks, also known as code render blocks, contain information about the code, such as the language used in developing Web applications.

Now, let's build the application by pressing the F5 key. It simply displays a message, Welcome to the World of ASP.NET 3.5 on the browser, as shown in Figure 15.11:

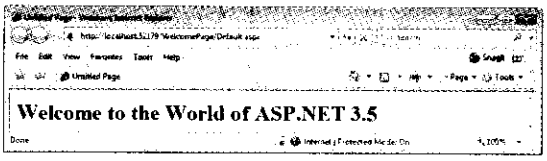


Figure 15.11: The Welcome Page

Now, let's learn about these blocks in detail.

Code Render Blocks

Code render blocks define the inline code or inline expressions that are executed when a Web page is rendered. This inline code is then executed by the server and the output is displayed on the client browser that requested for the page. The code render blocks are capable of displaying information on the client browser without customarily calling the `Write` method. The code render blocks are represented on a page by the `<% %>` symbols. The code present inside these symbols is executed in the top-down manner. Now, let's put the concept of code render blocks into use by creating an application, `CodeRenderBlock`. This application uses a code render block to implement a loop used for changing the text size. You can find the code for the `CodeRenderBlock` application in the Code\ASP.NET\Chapter 15\CodeRenderBlock folder on the CD. You can find the code for the Default.aspx page for code render blocks in Listing 15.2:

Listing 15.2: Showing the Code for the Default.aspx Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
      <div>
            <% Dim I As Int64
            For I = 0 To 6 Step 1%>

            <font size=" <%=I%>" > Welcome to ASP.NET 3.5</font> <br>
            <% Next %>
      </div>
  </form>
</body>
</html>
```

Now, let's build the application by pressing the F5 key. The output is as shown in Figure 15.12:

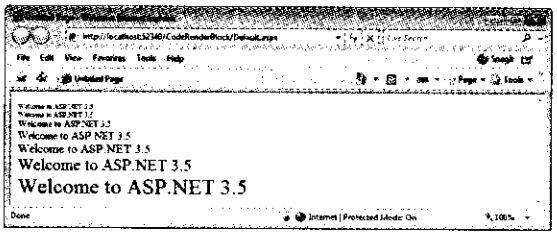


Figure 15.12: Output of CodeRenderBlock Application

Understanding ASP.NET 3.5 Page Directives

ASP.NET page directives are instructions to specify optional settings, such as registering a custom control and page language. These settings are used to describe how ASP.NET Web form pages (.aspx) or User control (.ascx) pages are processed by.NET Framework.

Page directives can be placed anywhere in an `.aspx` or `.ascx` file; although the standard practice is to include them at the top of the file.

The syntax to use a page directive is as follows:

```
<%@ directive [attribute=value]* %>
```

ASP.NET treats any directive block (`<%@ %>`) that does not contain an explicit directive name as a `@ Page` directive (for a page) or as a `@ Control` directive (for a user-control). ASP.NET 3.5 pages consist of the following directives:

- @ Page
- @ Control
- @ Import
- @ Implements
- @ Register
- @ Assembly
- @ Master
- @ WebHandler
- @ PreviousPageType
- @ MasterType
- @ OutputCache
- @ Reference

The @ Page Directive

The @ Page directive defines page-specific attributes used by the ASP.NET page parser and compiler. It can be included only in `.aspx` files.

The syntax for the `@ Page` directive is as follows:

```
<%@ Page attribute="value" [attribute="value"...] %>
```

The @ Control Directive

The @ Control directive defines control-specific attributes used by the ASP.NET page parser and compiler. It can only be used in `.ascx` files (user controls).

The syntax for the `@ Control` directive is as follows:

```
<%@ Control attribute="value" [attribute="value"...] %>
```

The @ Import Directive

The @ Import directive explicitly imports a namespace into ASP.NET application files, such as Web page, a user control, a master page,. or a `Global.asax` file. It also makes all classes and interfaces of the imported namespace available to the application files.

The syntax for the `@ Import` directive is as follows:

```
<%@ Import namespace="value" %>
```

The @ Implements Directive

The @ Implements directive indicates that the Web page, user control, or master page must implement the specified .NET Framework interface.

The syntax for the `@ Implements` directive is as follows:

```
<%@ Implements interface="ValidInterfaceName" %>
```

The @ Register Directive

The @ Register directive creates an association between tag prefix and custom controls, which helps you to refer to custom controls in ASP.NET Web pages and user controls.

The syntax for the `@ Register` directive is as follows:

```
<%@ Register tagprefix="tagprefix"
  namespace="namespace"
  assembly="assembly" %>
<%@ Register tagprefix="tagprefix"
  namespace="namespace" %>
<%@ Register tagprefix="tagprefix"
  tagname="tagname"
  src="pathname" %>
```

The @ Assembly Directive

The @ Assembly directive links an assembly to the ASP.NET application files, such as a Web page, a user control, a master page, or a `Global.asax` file during compilation and makes all the classes and interfaces of an assembly available for use to the application.

The syntax for the `@ Assembly` directive is as follows:

```
<%@ Assembly Name="assemblyname" %>
<%@ Assembly Src="pathname" %>
```

The @ Master Directive

The @ Master directive defines attributes of a master page (`.master` file) that are used by the ASP.NET page parser and compiler.

The syntax for the `@ Master` directive is as follows:

```
<%@ Master attribute="value" [attribute="value"...] %>
```

The @ WebHandler Directive

The @ WebHandler directive identifies an ASP.NET `IHttpHandler` page.

The syntax for the `@WebHandler` directive is as follows:

```
<%@ WebHandler attribute="value" [attribute="value"...] %>
```

The @ PreviousPageType Directive

The @ PreviousPageType directive provides the means to get strong typing against the previous page as accessed through the `PreviousPage` property.

The syntax for the `@ PreviousPage` directive is as follows:

```
<%@ PreviousPageType attribute="value" [attribute="value"...] %>
```

The @ MasterType Directive

The @ MasterType directive provides a way to create a strongly typed reference to the ASP.NET master page when the master page is accessed from the `Master` property.

The syntax for the `@ MasterType` directive is as follows:

```
<%@ MasterType attribute="value" [attribute="value"...] %>
```

The @ OutputCache Directive

The @ OutputCache directive declaratively controls the output caching policies of a Web page or a user control.

The syntax for the `@ OutputCache` directive is as follows:

```
<%@ OutputCache Duration="#ofseconds"
  Location="Any | Client | Downstream | Server | None |
  ServerAndClient "
  Shared="True | False"
  VaryByControl="controlname"
  VaryByCustom="browser | customstring"
  VaryByHeader="headers"
  VaryByParam="parametername"
  CacheProfile="cache profile name | ""
  NoStore="true | false"
  SqlDependency="database/table name pair | CommandNotification"
%>
```

The @ Reference Directive

The @ Reference directive dynamically compiles and links the user control, page source file, or arbitrary file located at some virtual path against the current Web page or user control in which this directive is declared.

The syntax for the `@ Reference` directive is as follows:

```
<%@ Reference Page="path to .aspx page"
  Control="path to .ascx file"
  virtualPath="path to file" %>
```

Working with Server Controls

Along with the HTML and programming code, an ASP.NET Web page also contains server controls, such as the `TextBox` and `Image` controls. Server controls are programmable objects that act as User Interface (UI) elements on a Web page. For instance, a typical example of server control can be a `TextBox` control that takes data from the user as input and displays the output at the click of a `Button` control. The code for a server control is executed and compiled at the server, so these controls are also known as server-side objects. Working with server controls is a great experience as you can define their complete behavior by setting the properties. You can set the properties either by explicitly writing them in the server control tag or by defining them in the `Properties` window.

You can use server control tags or intrinsic HTML tags containing the `runat="server"` attribute to declare server controls in a Web application. The server controls declared within the intrinsic HTML tags are executed with the help of the `System.Web.UI.HtmlControls` namespace.

Now, let's develop an application, named `ServerControl` that uses various server controls, such as Label, TextBox, DropDownList, and Button. You can find the code for the `ServerControl` application in the Code\ASP.NET\Chapter 15\ServerControl folder on the CD. Listing 15.3 shows the code of the `Default.aspx` page for the server controls:

Listing 15.3: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
      <div>
      <asp:Label ID="Label1" runat="server" Text="Select
        Title"></asp:Label> <asp:DropDownList ID="DropDownList1"
        runat="server">
      <asp:listitem >Mr</asp:listitem>
```

```
            <asp:listitem >Mrs</asp:listitem>
            <asp:listitem >Miss</asp:listitem>
            </asp:DropDownList>
               <asp:Label ID="Label2" runat="server" Text="Enter
              Name"></asp:Label>
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
            <br />
               <br />
               
            <asp:Button ID="Button1" runat="server" Text="Submit" Width="60px" /></div>
    </form>
</body>
</html>
```

Now, add the code in the code-behind file of the `Default.aspx` page for the server controls, as shown in Listing 15.4:

Listing 15.4: Showing the code for the Code-Behind File of the `Default.aspx` Page

```
Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
      System.EventArgs) Handles Button1.Click
    Dim str As String = "Welcome  " & DropDownList1.SelectedValue & " " &
      TextBox1.Text
    Button1.Visible = False
    DropDownList1.Visible = False
    TextBox1.Visible = False
    Label1.Visible = False
    Label2.Visible = False
    Response.Write(str)
    End Sub
End Class
```

Now, press the F5 key to execute the Web application, as shown in Figure 15.13:

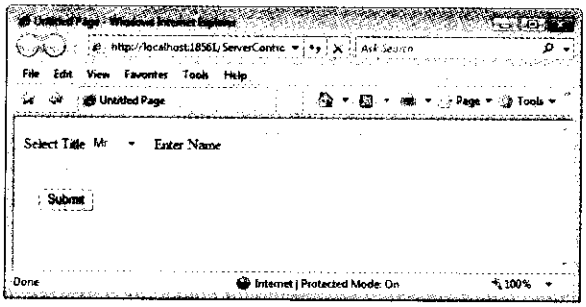


Figure 15.13: Output of ServerControl Application

As shown in Figure 15.13, the Web page contains a `DropDownList`, `TextBox`, two Labels, and a `Button` control that allows you to enter your name. Click the Submit button after selecting the title from the Select Title field and entering text in the Enter Name field. Figure 15.14 shows the output:

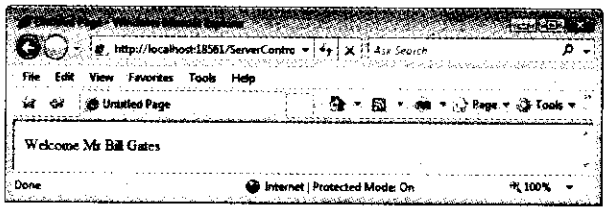


Figure 15.14: Output Generated after Clicking the Submit Button

After learning how to create a Web application in ASP.NET, let's now discuss the ASP.NET 3.5 provider model.

Understanding ASP.NET 3.5 Provider Model

The provider model illustrates the position of providers. Providers work as a bridge between `Data Stores` such as `SQL Server` and `ASP.NET Services` such as Membership in ASP.NET applications. You can best visualize the ASP.NET provider model by looking at Figure 15.15:

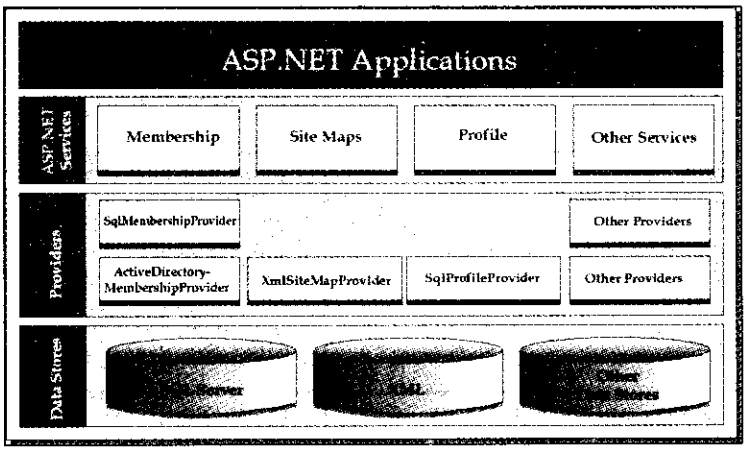


Figure 15.15: ASP.NET 3.5 Provider Model

As shown in Figure 15.15, providers act as communication channels that communicate with data sources. To elaborate more on the concept of providers, let's take the example of an online banking website, which allows its customers to access and modify their account information. In such a situation, the bank needs to maintain the records of each customer in a storage media, such as `Oracle` or `DB2` databases. Whenever a user tries to access the information through a website, an appropriate provider is selected, which retrieves or updates the information in the data source. The type of the provider selected depends on the type of data source used. The functionality of providers is very much the same as that of device drivers that extract data from the physical hardware devices, such as floppy disc and CD.

Providers can be of many types, based on the type of services they provide. The Membership provider stores the user names and passwords, and the Roles provider stores the user roles. By default, ASP.NET 3.5 uses the `AspNetSqlMembershipProvider` provider. You can change the default provider by using the `Web Site Administration Tool` or the `Microsoft Management Console` (MMC), which provides various management tools called snap-ins.

Table 15.3 lists the different types of service providers:

Table 15.3: Different Types of Service Providers

Provider Type	Default Provider(s)	Built-in Provider(s)
Membership	`System.Web.Security.SqlMembershipProvider`	`System.Web.Security.ActiveDirectoryMembershipProvider` `System.Web.Security.SqlMembershipProvider`

Role management	`System.Web.Security.SqlRoleProvider`	`System.Web.Security.AuthorizationStoreRoleProvider`
		`System.Web.Security.SqlRoleProvider`
		`System.Web.Security.WindowsTokenRoleProvider`

Site map	`System.Web.XmlSiteMapProvider`	`System.Web.XmlSiteMapProvider`

Table 15.3: Different Types of Service Providers

Provider Type	Default Provider(s)	Web.config Provider(s)
Profile	`System.Web.Profile.SqlProfilePr ovider`	`System.Web.Profile.SqlProfileProvider`

Session state	`System.Web.SessionState.InProcS essionStateStore`	`System.Web.SessionState.InProcSessionSta teStore`
		`System.Web.SessionState.OutOfProcSession StateStore`
		`System.Web.SessionState.SqlSessionStateS tore`

Web events	N/A	`System.Web.Management.EventLogWebEventPr ovider`
		`System.Web.Management.SimpleMailWebEvent Provider`
		`System.Web.Management.TemplatedMailWebEv entProvider`
		`System.Web.Management.SqlWebEventProvider`
		`System.Web.Management.TraceWebEventProvi der`

Web Parts personalization	`System.Web.UI.WebControls.WebPa rts.SqlPersonalizationProvider`	`System.Web.UI.WebControls.WebParts.SqlPe rsonalizationProvider`
Protected configuration	N/A	`System.Configuration.DPAPIProtectedConfi gurationProvider` `System.Configuration.RSAProtectedConfigu rationProvider`

All the providers are derived from a common base class known as `ProviderBase`, which is defined in the `System.Configuration.Provider` namespace.

ASP.NET 3.5 Coding Models

Generally, a Web form consists of controls, such as Label, Button, Hyperlinks, and business logic. You use ASP.NET coding techniques to manage these controls and business logic. ASP.NET 3.5 coding techniques refer to the methodologies used for writing code in a Web application. ASP.NET 3.5 provides two types of coding techniques, `Single-File page model` and `Code-Behind page model`. In the single-file coding approach, developers write code directly in the `.aspx` page of the application. A major drawback of the single-file code model is that writing code in a single file results in difficult-to-read Web pages that mix presentation (HTML) with functionality (code). This drawback is specifically evident while developing complex applications. To deal with this drawback of the single-file page model, ASP.NET has introduced the code-behind page model. In this model, you need to maintain separate code files for each Web page. One file stores the code to implement the functionalities of a Web page written in some programming language, such as VB .NET and C#; and the other file stores the HTML markup of the Web application. The following list shows the coding models in ASP.NET 3.5:

- Single-File Page Model
- Code-Behind Page Model

Now, let's discuss each of these models one by one.

Single-File Page Model

In the single-file page model, the functionalities of a Web application and the HTML markup are implemented in the same file. The programming code is in a **script** block that contains the attribute **runat="server"** to mark it as code that ASP.NET should execute. Visual Studio also supports the Intellisense feature. The IntelliSense feature

provides the facility of quickly accessing the members of a particular type, from which we can select the one we require, that is, when you start writing code inside the `<% %>` brackets, Visual Studio provides full IntelliSense support for code completion, help, and code insertions. Now, let's understand the single-file page model through an application named as `SingleFileModelExampleVB`. You can find the code for the `SingleFileModelExampleVB` application in the Code\ASP.NET\Chapter 15\SingleFileModelExampleVB folder on the CD. You can find the code for the `Default.aspx` page for the single-file page model in Listing 15.5:

Listing 15.5: Showing the Code for the Default.aspx Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
  Protected Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
        Label1.Text = "Clicked at " & DateTime.Now.ToString()
  End Sub
</script>
<html>
<head id="Head1" runat="server">
  <title>Single-File Page Model</title>
</head>
<body>
  <form id="form1" runat="server">
        <div>
              <asp:Label ID="Label1"
              runat="server"></asp:Label>
              <asp:Button ID="Button1"
                    runat="server" OnClick="Button1_Click" Text="Click Me!">
              </asp:Button>
        </div>
  </form>
</body>
</html>
```

In the preceding listing, we have placed the button's click-event code inside the `<script>` tags at the beginning. It should be kept in mind that the `runat="server"` attribute is mandatory within the `<script>` tags, because it will post the page to the Web server.

Now, let's build the application by pressing the F5 key. The output of the application is as shown in Figure 15.16:

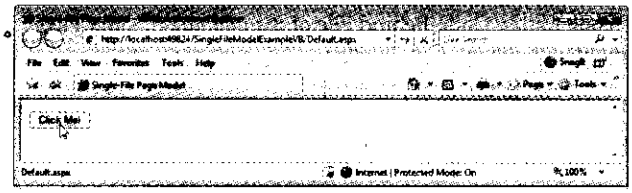


Figure 15.16: Showing the Output of the Application

Now, click the Click Me! button (Figure 15.16). The result of the `click` event is as shown in Figure 15.17:

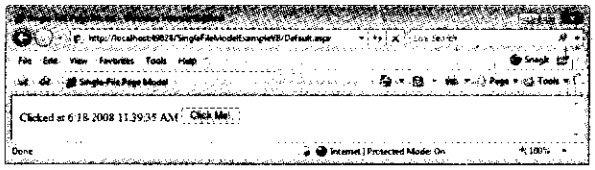


Figure 15.17: Result of Clicking the Click Me! Button

Code-Behind Page Model

In the code-behind model, there are two separate files, `Default.aspx` and `Default.aspx.vb`, for ASP.NET presentation tags and programming logic to implement the functionalities of a Web application. These two files are then linked together to run the Web application. By default, all the different versions of ASP.NET--ASP.NET 1.x, ASP.NET 2.0, ASP.NET 3.x support this model. The code-behind model in ASP.NET 3.5 is similar to the code-behind model supported by ASP.NET 2.0.

The code-behind file uses the partial class rather than the main class and it does not inherit the `.aspx` file, whereas both the files are combined together during compilation to implement the complete application.

Now, let's understand the code-behind model through an application named as `CodeBehindExampleVB`. You can find the code for the `CodeBehindExampleVB` application in the Code\ASP.NET\Chapter 15\ CodeBehindExampleVB folder on the CD. You can find the code for the `Default.aspx` page for the code-behind model in Listing 15.6:

Listing 15.6: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head id="Head1" runat="server" >
  <title>Code-Behind Page Model</title>
</head>
<body>
  <form id="form1" runat="server">
      <div>
          <asp:Label id="Label1"
          runat="server" ></asp:Label>
          <br />
          <asp:Button id="Button1"
                runat="server"
                Text="Click Me!" >
          </asp:Button>
      </div>
  </form>
</body>
</html>
```

Now, add the code in the code-behind file of the `Default.aspx` page for the code-behind page model, as shown in Listing 15.7:

Listing 15.7: Showing the code for the Code-Behind File of the `Default.aspx` Page

```
Partial Class _Default
  Inherits System.Web.UI.Page
  Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles Button1.Click
  Label1.Text = "Clicked at " & DateTime.Now.ToString()
  End Sub
End Class
```

The output of this listing is the same as shown in Figure 15.16 and Figure 15.17.

Implementing Code Sharing

Code sharing is a mechanism through which a common code is shared among different pages of a Web application. Code sharing is particularly beneficial when you want to make the code, with specific functionality, accessible to all the pages in a Web application. To implement code sharing, the following three methods are used:

- Using the `Code` Directory
- Using the `bin` Directory
- Using the Global Assembly Cache

Using the Code Directory

ASP.NET 3.5 provides a unique directory known as `App_Code` to store code files that are accessible to all the Web pages of a Web application. When you add something inside the `App_Code` folder, such as class or `.wsdl` files, it is automatically detected and compiled. Once compiled, any page in your Web application can access those files or classes. To create the `App_Code` folder, right-click the project name in the Solution Explorer and select Add ASP.NET Folder→App_Code. It adds the `App_Code` folder in the Solution Explorer. Now, right-click the `App_Code` folder and select the Add New Item option. The Add New Item dialog box appears, as shown in Figure 15.18:

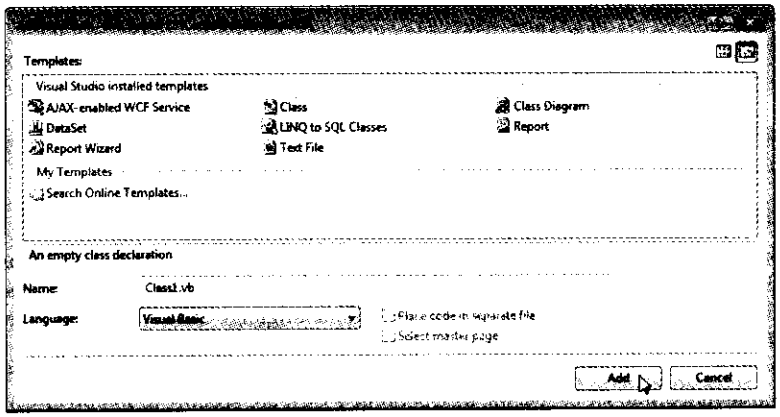


Figure 15.18: Add New Item Dialog Box

As shown in Figure 15.18, you can add as many different classes and files to the `App_Code` folder so that they are accessible across all the pages in the Web application. To add an item, just select its icon and click the Add button. Figure 15.19 shows the Solution Explorer, with a class file added to the `App_Code` folder:

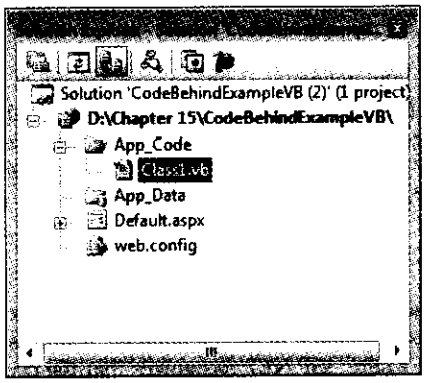


Figure 15.19: The Solution Explorer

Using the bin Directory

The code for a specific functionality in a Web application can also be made accessible to all the Web pages of a Web application by storing the sharable files (the files you want to share) in the `bin` directory. Since ASP.NET 1.x, this method is used for sharing the code. The `bin` directory is similar to the `App_Code` directory except that it stores the precompiled assemblies of the code files that are required by all the Web pages of a Web application.

You can use this method when you need to reuse the code written in other Web applications. In this case, you need to include the executable or the `DLL` file of that code in the `bin` directory.

Using the Global Assembly Cache

Another method of making the code for a specific functionality available to all the Web pages of a Web application is by using the global assembly cache. .NET Framework provides various assemblies that are stored in the global assembly cache. These assemblies can be accessed by all the Web pages of the Web application. To register or add the required assemblies in the global assembly cache, write the following code in the `web.config` file of a Web application:

```
<configuration>
  <system.web>
    <compilation>
      <assemblies>
        <add assembly="System.Data, Version=1.0.2411.0,
        Culture=neutral,
        PublicKeyToken=b77a5c561934e089"/>
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```

Now, let's learn about compilation in ASP.NET 3.5.

Compilation in ASP.NET 3.5

ASP.NET first compiles code into one or more assemblies. Assemblies are files with the extension `.dll`. You can choose multiple languages, such as VB, C#, and others to write ASP.NET code. When the code is compiled, it is converted into a language-independent and CPU-independent language called Microsoft Intermediate Language (MSIL). ASP.NET 3.5 introduced a new tool for compilation known as Merge tool, which we will discuss later in this chapter. Compilation enhances performance, security, stability, and interoperability of a Web application. At runtime, MSIL is translated into CPU-specific instructions, which are specific to the processors of the computers running the applications. The following topics help you to familiarize with compilation in ASP.NET 3.5:

- Compiling on First Request
- Recompiling on Change
- Compilation Dependencies
- Using the Merge Tool

Compiling on First Request

In ASP.NET, Web pages and code files are compiled dynamically by default when users first request a resource. A resource can be an ASP.NET page (`.aspx` file), which the user requests from a website. ASP.NET supports the dynamic compilation of ASP.NET pages (`.aspx` files), ASP.NET Web services (`.asmx` files), ASP.NET `HTTP handlers` (`.ashx` files), and ASP.NET application files (`Global.asax`), as well as other files, such as source code and class files.

Recompiling on Change

If you make any changes to a dynamically compiled ASP.NET file, it automatically invalidates the file's cached compiled assembly and triggers recompilation of all the affected resources, such as `.dll` files. Now, when the request to the application is made again, ASP.NET recognizes that the code has been changed and recompiles the affected resources of the Web application. This process enables you to quickly develop applications with a minimum of compilation overhead. Whether a single page or the entire website is recompiled depends on the percentage of changes made to the resources.

Compilation Dependencies

ASP.NET compiles files in a particular order when a request is made to an application. The first items to be compiled are referred to as the top-level items. After the first request, the top-level items are recompiled only if

dependency changes. Top-level items include the `App_GlobalResources` folder, `App_WebResources` folder, profile properties, `App_Code` folder, and `Global.asax` file. After the top-level items are compiled, ASP.NET compiles additional items. These items include the `App_LocalResources` folder, individual ASP.NET pages (`.aspx` files), ASP.NET user controls (`.ascx` files), ASP.NET HTTP Handlers (`.ashx` files), and ASP.NET HTTP modules (`.asmx` files) as well as themes, master pages, and other source files.

Using the Merge Tool

The ASP.NET Merge tool (`Aspnet_merge.exe`) enables you to combine and manage assemblies that are created by the ASP.NET Compilation tool (`Aspnet_compiler.exe`). The ASP.NET Compilation tool enables you to compile an ASP.NET Web application for deployment to a target location, such as a production server. The ASP.NET Merge tool works on assemblies that have been created by using ASP.NET version 2.0 or advanced versions. It creates one assembly for each content folder in the target website, or it creates one assembly for each content file. It also gives you additional flexibility for deployment. The Merge tool enables you to do the following tasks:

- Creating one assembly for the whole website
- Creating an assembly for each website folder, and adds a prefix to the assembly name
- Creating a single assembly for just the website UI elements, such as pages and controls

The syntax for using the Merge tool is as follows:

```
aspnet_merge    [-?]
                applicationPath
                [-keyfile filename [-delaysign]]
                [-o assemblyname | -w assemblyname | -prefix prefix]
                [-copyattrs [assemblyfile]]
                [-debug]
                [-nologo]
                [-errorstack]
                [-r]
                [-xmldocs]
                [-a]
                [-logfile logfile]
                [-allowattrs textfile]
```

Summary

In this chapter, we have covered some important aspects of developing a Web application, such as how to specify the location of a Web application by using the File System, HTTP, and FTP options. This chapter also focuses on server controls, page directives, provider models, and coding models. It also describes the concepts of code sharing and compilation, required for developing Web applications. We also discussed a new tool introduced in ASP.NET 3.5 for compilation known as the Merge tool.

In the next chapter, we discuss about the structure and state management concepts related to an ASP.NET Web application.

Quick Revise

Q1. Which protocol is used to host the site on the remote IIS server in Visual Studio?

1. **HTTP**
2. **FTP**
3. **SOAP**
4. **TCP/IP**

Ans: HTTP

Q2. Which of the file type is not managed by ASP.NET?

1. **.asax**
2. **.cd**

3. **.compile**
4. **.cdx**

Ans: .cdx

Q3. What do you understand by ASP.NET page directives?

Ans: ASP.NET page directives are instructions to specify optional settings, such as registering a custom control and page language. These settings are used to describe how ASP.NET Web pages (.aspx) or User control (.ascx) pages are processed by.NET Framework.

Q4. Is it necessary to place page directives at the top of the file?

Ans: No

Q5. Mention page directives of ASP.NET?

Ans: ASP.NET 3.5 pages consist of the following directives:

1. @ Page
2. @ Control
3. @ Import
4. @ Implements
5. @ Register
6. @ Assembly
7. @ Master
8. @ WebHandler
9. @ PreviousPageType
10. @ MasterType
11. @ OutputCache
12. @ Reference

Q6. What is the functioning of providers in ASP.NET?

Ans: Providers work as a bridge between Data Stores and ASP.NET Services.

Q7. What is the drawback of single-file code model?

Ans: Drawback of the single-file code model is that writing code in a single file results in difficult-to-read Web pages that mix presentation (HTML) with functionality (code). This drawback is specifically evident while developing complex applications.

Q8. What is the function of membership provider and which membership provider is used ASP.NET by default?

Ans: The Membership provider stores the user names and passwords. The default membership provider is AspNetSqlMembershipProvider.

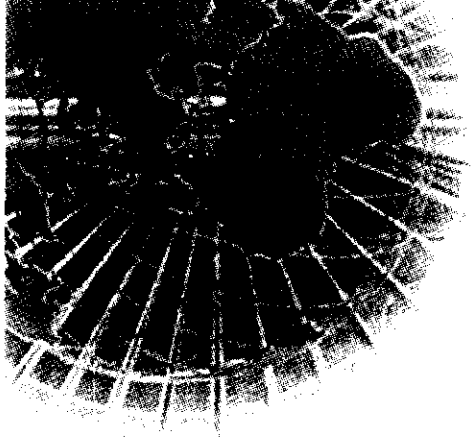
Q9. The @ Control directive is used in .....files?

1. **.aspx**
2. **.ascx**
3. **Global.asax**
4. **.master**

Ans: .ascx

Q.10. How can you register IIS server in ASP.NET 3.5?

Ans. To register IIS server in ASP.NET 3.5, run the Visual Studio 2008 command prompt as an administrator and use the aspnet_regiis -i syntax.



16

Application Structure and State

The structure of a Web application includes three main concepts: application domain, application lifetime, and application directory structure. Apart from these concepts, a Web application also contains a component called the `Global.asax` file. The `Global.asax` file contains the code for the events that are raised when the application is executed. Basically, an ASP.NET Web application comprises files, pages, handlers, and executable code, which are invoked from the virtual directories of a Web server.

State of a Web concept refers to the status of a Web application at any instant of time. State of the Web application is associated with its runtime. Preserving application state ensures that the changes made by a user (such as the products selected and saved in the shopping cart in a case of any online shopping website) are persisted whenever a page is reloaded. States are further categorized into several types depending on the nature of information they preserve. These types include application state, session state, and view state.

In this chapter, we describe the structure of a Web application and explain the concept of states. We also get to know about the `Global.asax` file, which is used to implement application-level and session-level events. In addition, we learn how to add objects to it. The chapter concludes with a detailed discussion about HTTP Handlers used to handle user requests for Web application resources, and postbacks used for sending data back to the server for processing.

Structure of an Application

As already stated, the structure of an ASP.NET Web application includes the concepts of application domain, application lifetime, and application directory structure. Application domain is a virtual boundary inside which an application runs, while application lifetime is the span of time for which an application domain exists. Application directory structure, on the other hand, specifies the directory structure that organizes the various entities associated with an application, such as references, resources, and code files, in separate directories. Each directory possesses a well-qualified name that reflects what is stored inside it. Now, let's explore all these concepts one by one.

The Application Domain

To understand the concept of domain, we first need to understand how a Web application works after it is deployed. We deploy a Web application created in ASP.NET on a Web server, such as the Internet Information Services (IIS), and Windows Activation Services (WAS). The client browser sends a request for the Web application to the Web server, which further passes the request to the ASP.NET worker process. The worker process then separates the code execution into different application domains with the Web services and Web pages hosted on the same virtual directory. These Web services and Web pages executes in the same application domain, whereas, the Web services and Web pages hosted on different virtual directories executes in different application domains. An application domain is actually implemented by Common Language Runtime (CLR), and the prime objective of the application domain is to prevent concurrently running applications from entering into each other's domain.

NOTE

ASP.NET runs within a process known as ASP.NET worker process. The ASP.NET worker process allows ASP.NET 3.5 Web applications to run on the Web server. It enables different assemblies of a program to work in conjunction with each other.

What do we gain from the concept of the application domain? The most vital benefit of application domains is the independent execution of Web applications. None of the applications can access the in-memory resources of other simultaneously running applications. The main component that helps in the independent execution of applications is a virtual directory. In case of an IIS Web server, the virtual directory with all its subdirectories is created in the IIS console. Some of the primary application entities that are stored in the virtual directory are:

- **Web pages**—Files that are used as Web Form pages in the Microsoft .NET environment. The extension of these files is `.aspx`.
- **Web services**—Files that enable you to share information with applications on other computers and platforms using XML Web services. These files have the `.asmx` extension.

- **Code-behind files**—Source code files that contain code for an ASP.NET page in a separate class file. The source code files have the .vb extension if the files are coded in VB.
- **Configuration files**—Configuration files, such as `web.config` which stores the configuration settings of a Web application and `machine.config` which store the configuration settings of the machine on which the Web application is running.
- `Global.asax` **files**—Files that contain event handlers to respond to events, such as starting of an ASP.NET Web application for the first time.

In addition, there are other resources in the virtual directory, such as images and XML files. The virtual directory encapsulates all these resources of an ASP.NET Web application and forms an application domain whenever the application is called for the first time.

The Application Lifetime

Application lifetime refers to the time span for which an application domain persists. Now, this does not mean that application lifetime is equivalent to the time span for which an application runs. In fact, an application domain can shut down in several circumstances, including specific error conditions, configuration changes, or Web server shut-down. Therefore, an application's runtime might encounter repeated restarting of the application domain. The application domain might require a restart under the following circumstances:

- Excess requests in the queue
- Excess memory usage by an application
- Lengthy lifetime of the application
- Modifications in the `web.config` file
- Replacement of existing Dynamic Link Libraries (.dll files) or Web files

It is not that the shut-down of an application domain is bad on any count. As a matter of fact, developers deliberately include specific settings in the machine.config file to ensure that the application domain is recycled at specific time intervals. This helps in monitoring the performance of the application.

The Application Directory Structure

An efficiently designed directory structure plays a key role in Web application development. It segregates all the code and resources used by an application among different directories, thereby enhancing the productivity of developers. Let's take an example to illustrate this. Suppose you are developing a Web application, the code for which has been divided under different modules, with each module handled by different team members. In such a scenario, it will be easier for you to just put the code, developed by your team members, in different directories and run the application. In this way, the debugging process can also be performed in a more structured manner. Another advantage of using a directory structure is that it enhances the reusability of an application. It becomes easy for you to upgrade the application in the long run because of its structured code files and application resources.

In the 1.x version of ASP.NET and earlier, there was only one default directory and bin for an application and you had to create your own custom directories, if any. However, in ASP.NET 2.0, there is support for a default directory structure, which provides several built-in directories for Web applications. These directories are as follows:

- `Bin`—Contains compiled .NET assemblies, containing precompiled Web pages and Web service classes. You can directly use these assemblies in your application, thereby reducing the amount of code that needs to be written for an ASP.NET Web application.
- `App_Code`—Contains source code files, compiled dynamically, to be used in the ASP.NET Web application.
- `App_GlobalResources`—Contains the resources that can be used in all Web pages as well as the controls of the ASP.NET Web application.
- `App_LocalResources`—Contains the resources that are accessible only to a specific Web page of the ASP.NET Web application.

- App_WebReferences—Contains the references to the Web services used by the ASP.NET Web application.
- App_Data—Contains the database and XML files of the ASP.NET Web application.
- App_Browsers—Contains browser definition files that identify browsers for different ASP.NET Web applications.
- App_Themes—Contains the themes used by the ASP.NET Web application. The themes can be applied to a control or Web page, or to the entire ASP.NET Web application.

Next, let's discuss the Global.asax application file, which is also known as the ASP.NET application file.

The **Global.asax** Application File

The Global.asax file resides in the root directory of an ASP.NET Web application and is also called the ASP.NET application file. This file contains the code that is executed when certain events, such as start of an application or error in an application, are raised by an ASP.NET Web application.

Events and states, such as session state and application state that are specified in the Global.asax file, are applied to all the resources of the Web application. For example, if an application state variable is defined in the Global.asax file, then all the .aspx files within the root directory can access the variable.

The code in the Global.asax file is written in the same way as in Web forms; the only difference is that the code written in the Global.asax file does not contain HTML or ASP.NET tags. Instead, the code in this file contains methods with predefined names.

The Global.asax file is created either in Notepad or as a compiled class, which is deployed as an assembly in the \bin directory of the ASP.NET Web application. Now, let's create an application named GlobalVB, which uses the Global.asax file to display the date and time. You can find the code for the GlobalVB application in the Code\ASP.NET\Chapter 16\GlobalVB folder on the CD. Listing 16.1 shows the code of the Global.asax file:

Listing 16.1: Showing the Code for the Global.asax File

```
<%@ Application Language="VB" %>
<script runat="server">
    Sub Application_OnEndRequest(ByVal sender As Object, ByVal e As EventArgs)
        Response.Write("This page was executed on:" & DateTime.Now.ToString)
    End Sub
</script>
```

To add a Global.asax file to an application, first right-click the name of the website in Solution Explorer and select the Add New Item option, and then after selecting the Global Application Class template on the Add New Item dialog box and click the Add button.

Running the application with the preceding listing opens a Web page that displays the date and time, using the Global.asax file, as shown in Figure 16.1:

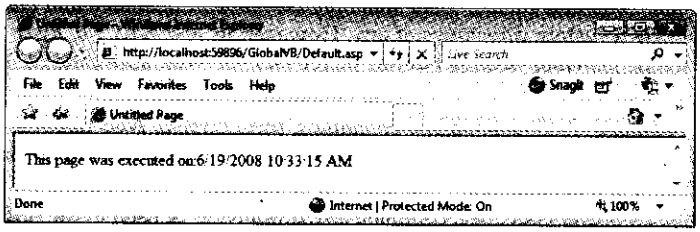


Figure 16.1: Output of the Global.asax File

The new Global.asax file added to the ASP.NET Web application contains empty event handlers for the commonly used application events, such as the start and end of an application. The code for the event handlers is inserted in the appropriate method of the Global.asax file. The name of the method for which an event handler is created should be similar to the name of the event occurring in the Web application. For example, the

`Application_OnStart()` method is called when the `Application_OnStart` event occurs in the ASP.NET Web application.

The various methods corresponding to the events that occur in the `Global.asax` file are invoked in the following order when an ASP.NET Web application starts:

- `Application_BeginRequest()` – Invoked when a request for the Web application is received
- `Application_AuthenticateRequest()` – Invoked just before the authentication of user credentials is performed. You can use this method to specify your own authentication logic before the authentication process begins
- `Application_AuthorizeRequest()` – Invoked when the current user's credentials are successfully validated. You can use this method to assign special permissions to the user
- `Application_ResolveRequestCache()` – Invoked when the ASP.NET Page Framework completes the execution of authorization request. This method is mainly used to handle the output caching of the application that renders the cached HTML without executing the code again
- `Application_AcquireRequestState()` – Invoked just before the session information is retrieved for the current client request
- `Application_PreRequestHandlerExecute()` – Invoked before the ASP.NET page framework begins to run the HTTP Handler to handle a request
- `Application_PostRequestHandlerExecute()` – Invoked after the HTTP Handler has finished executing the request
- `Application_ReleaseRequestState()` – Invoked just before the session information is serialized from the Session collection for the next request
- `Application_UpdateRequestCache()` – Invoked just before the information for handling the request is added to the output cache of the Web page
- `Application_EndRequest()` – Invoked at the end of the request

The methods corresponding to the events that are not invoked with each user request are:

- `Application_Start()` – Invoked when an ASP.NET Web application starts and an application domain is created. An ASP.NET Web application starts when any resource in the application is requested.
- `Session_Start()` – Invoked when a session is started. A new session starts each time a new user requests a page.
- `Application_Error()` – Invoked when an error occurs. You can use this method to handle errors on the ASP.NET Web application instead of using the `web.config` method.
- `Session_End()` – Invoked when a user's session expires.
- `Application_End()` – Invoked when an application ends or terminates. The application terminates when there are no more active sessions.
- `Application_Disposed()` – Invoked when an application is destroyed.

The `Global.asax` file is also used to declare the data that is available across different browser sessions. This process is known as application state and session state management.

There are some similarities between an ASP.NET Web page and the `Global.asax` file. Similar to an ASP.NET page, the `Global.asax` file is compiled when a request for any resource in an ASP.NET Web application is made for the first time. Another similarity is that when changes are made to the `Global.asax` file, ASP.NET Framework detects the changes and recompiles the file. New requests are then directed to the recompiled `Global.asax` file.

Now that you are familiar with the `Global.asax` application file, let's discuss about the states of an application.

Using States

State is quite an innovative concept in Web development because it eliminates the drawback of losing state data due to reloading of a Web page. By using states in a Web application, you can preserve the state of the application either at the server or client end. The state of a Web application helps you to store the runtime changes that have been made to the Web application. For example, as already described earlier, a change in the data source of the Web application might be initiated by a user when he/she selects and saves some products in the shopping cart. If you are not using states, these changes are discarded and are not saved. You may think that the whole concept of storing states is optional. However, under certain circumstances, using states with applications is imperative. For example, it is necessary to store states for Web applications, such as an e-commerce shopping site or an Intranet site of a company, to keep track of the requests of users for the items they have selected on the shopping site or the days requested for vacation on the Intranet site.

There are various methods for storing state information. These include the following:

- **Hidden Fields**—This control is not visible when a Web application is viewed in the browser. The content of the control is sent in the HTTP Form collection control along with the values of other controls to the server on page reloads. This control acts as a storage area for any page-specific storing information.
- **Cookies**—These are text files that store data, such as user ID and preferences at the client end. When a browser requests a Web page again, the cookie is sent along with the request. The Web server then retrieves the information from the cookie.
- **Query strings**—These are the information strings added at the end of a URL to maintain the state of a Web application. However, using query strings is not secure because their values are exposed to the Internet through the URL.

In all these methods, state information is stored at the client end. However, the state of an ASP.NET Web application can also be on a Web server. Saving the state of the application on the Web server is safer than saving it on a client machine. The following are some of the methods used to save state information on the Web server:

- **Application state**—Stores application data not frequently modified by users. An object of the `HttpApplicationState` class is used to store the state of an ASP.NET Web application.
- **Session state**—Stores information specific to a user session. User session refers to the duration for which a user uses a website. An object of the `HttpSessionState` class is used to store the session state for each ASP.NET Web application that is executed.
- **Profile Properties**—Stores the application state when the information to be stored is too large. The database can be used along with cookies to store the application state.

Now, let's discuss the various state types: application state, session state, and view state.

Application State

Application state is used to store data corresponding to all the variables of an ASP.NET Web application. The data in application state is stored once and read several times. Application state uses the `HttpApplicationstate` class to store and share the data throughout the application. You can access the information stored in an application state by using the `HttpApplication` class property. Data stored in application state is accessible to all the pages of the application and is the same for all users accessing the application. The `HttpApplicationstate` class provides a lock, method, which you can use to ensure that only one user is able to access and modify the data of an application at any instant of time.

Session State

Each client accessing a Web application maintains a distinct session with the Web server, and there is also specific information associated with each of these sessions. Session state is defined in the `<sessionState>` section of the `web.config` file. It also stores the data specific to a user session in session variables. Different session variables are created for each user session. In addition, session variables can be accessed from any page of the application. When a user accesses a page, a session ID for the user is created. The session ID is transferred between the server and the client over the HTTP protocol using cookies.

View State

View state stores page-specific information, when a page is posted back to the server. When a page is processed, the current state of the page and its controls is hashed (transforming a sequence of characters into a fixed-length value) into a string and saved as a hidden field on the page. Such a state of the page is called view state.

The `ViewState` property is used to save the view state of each control used in a page. If the `ViewState` property is not used, then the values written in the controls are not retained when the page is reloaded.

The hidden field of a view state is placed at the top of each page. The source code to maintain a ViewState contains an input tag with three attributes: type, name, and value. The value of the type attribute is set to hidden. The name attribute contains the name of the controls for which the `ViewState` property is used, and the value attribute specifies a string value, which is not in readable format. The string value is a combination of control values that are used in a Web page. When the Web page is reloaded, the server reads the values in the value attribute and restores them when the page is sent back to the client.

View state is maintained in a Web page by default. You can include the following directive at the top of the Web page to disable its view state:

```
<%@ Page EnableViewState = "false"%>
```

To disable the view state of a Web control, add the following attribute to the control:

```
EnableViewState ="false"
```

Using Application State, Session State, and View state

Let's create a sample Web application, `StateImplementationVB`, which uses all the three state types--application, session, and view. You can find the code for the `StateImplementationVB` application in the Code\ASP.NET\Chapter 16\StateImplementationVB folder on the CD. After creating the Web application, follow these steps:

1. Add the code in the `Default.aspx` page, as shown in Listing 16.2:

Listing 16.2: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" Debug="true" %>
<script runat="server">
  Sub Page_Load(ByVal Sender As Object, ByVal E As EventArgs)
        Application("visits") = Int(Application("visits") + 1)
        Label2.Text = Application("visits")
        Label3.Text = Session("mytext").ToString
  End Sub
  Protected Sub btnSubmit_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim txtvalue As String = txtName.Text
        ViewState.Add("item", txtvalue)
        Dim item As String = ViewState("item").ToString
        Label1.Text = item
  End Sub
</script>
<html>
<head>
  <title></title>
</head>
<body bgcolor="#FFFFFF">
  <form id="frmViewState" action="Default.aspx" method="post" runat="server">
  <strong><span style="text-decoration: underline">View State</span></strong><br />
  <br />
  Enter Your Name:
  <asp:TextBox ID="txtName" runat="server" />
  <br />
                         
            
  <br />
                         
             
  <asp:Button ID="btnSubmit" Text="Submit" OnClick="btnSubmit_Click" runat="server" />
```

```
    <br />
    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label><br />
     
    <br />
    <br />
    <br />
    <span style="text-decoration: underline"><strong>Application State<br />
    <br />
    </strong></span>Number of last visits:
    <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label><br />
    <br />
    <br />
    <strong><span style="text-decoration: underline">Session State<br />
    </span></strong>
    <br />
    <asp:Label ID="Label3" runat="server" Text="Label"></asp:Label><br />
    </form>
</body>
</html>
```

2. Next, add the `Global.asax` file in the `StateImplementationVB` Web application and add the code in the `Global.asax` file as shown in Listing 16.3:

Listing 16.3: Showing the Code for the `Global.asax` File

```
<%@ Application Language="VB" %>
<script runat="server">
  Sub Session_Start(ByVal Sender As Object, ByVal E As EventArgs)
        Session("mytext") = "I am the user!!"
  End Sub
  Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Code that runs on application startup
        Application("visits") = 0
  End Sub
</script>
```

3. Now, run the application. Figure 16.2 shows the output:

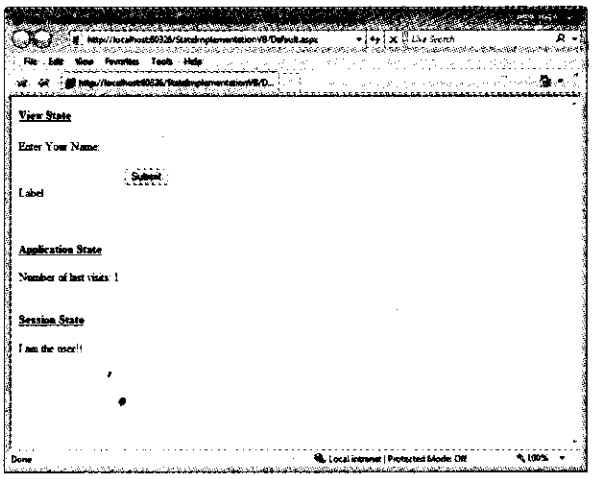


Figure 16.2: Output of the StateImplementation Web Application

4. Now, enter a name in the Enter Your Name textbox and click the Submit button. This displays the name as a Label control.
5. Next, refresh the page to reload it. Figure 16.3 shows the output after reloading the page:

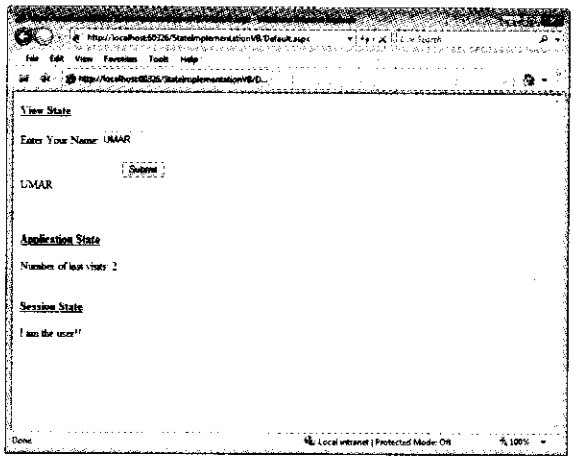


Figure 16.3: Output after Refreshing the Page

As shown in Figure 16.3, the name that you entered in the text box appears as a label, indicating that view state is maintained. The Number of last visits value also increases by 1, indicating that application state is maintained. However, the string I am the user!! remains unchanged. This denotes the property of session state, whose value does not change throughout a session.

Next, we discuss the concept of HTTP Handlers in Web applications.

HTTP Handlers

HTTP Handlers, as the name suggests, handle user requests for Web application resources. They are the backbone of the request–response model of Web applications. For each user request type, there is a specific event handler to handle the request and send back the corresponding response object.

Each user request to the IIS Web server flows through the HTTP pipeline, which refers to a series of components (HTTP Modules and HTTP Handlers) to process the request. HTTP Modules act as filters to process the request as it passes through the HTTP pipeline. The request, after passing through the HTTP Modules, is assigned to an HTTP Handler that determines the response of the server to the user request. The response then passes through the HTTP Modules once again and sends back to the user.

You can define HTTP Handlers in the `<httpHandlers>` section of a configuration file. The `<add>` element tag is used to add new handlers and the `<remove>` elements tag is used to remove existing handlers. To create an HTTP Handler, you need to define a class that implements an `IHttpHandler` interface. The two methods of the `IHttpHandler` interface are:

- `ProcessRequest()` – Invoked when a user request is received. It processes the request using the `HttpContext` object, which is passed as a parameter.
- `IsReusable()` – Determines whether the HTTP Handler object, accessed using the `ProcessRequest()` method, can be reused. The HTTP Handler object can only be reused if the `IsReusable()` method returns a true value. The object is discarded when the HTTP Handler object returns a false value.

Creating an HTTP Handler Application

To understand the concept of HTTP Handlers better, let's create a sample HTTP Handler Web application, called `HttpHandlerVB`, which sends a test message in response to an HTTP request. You can find the code for the `HttpHandlerVB` application in the Code\ASP.NET\Chapter 16\HttpHandlerVB folder on the CD. For this, you first need to create a class file `CustomHandler`, and add the code shown in Listing 16.4:

Listing 16.4: Showing the Code for the `CustomHandler` Class File

```
Imports Microsoft.VisualBasic
Namespace CustomHTTP
```

```
    Public Class CustomHandler
        Implements IHttpHandler
    Public ReadOnly Property IsReusable() As Boolean Implements
     System.Web.IHttpHandler.IsReusable
        Get
            Return True
        End Get
        End Property
    Public Sub ProcessRequest(ByVal context As System.Web.HttpContext) Implements
     System.Web.IHttpHandler.ProcessRequest
        Dim Response As HttpResponse = context.Response
            Response.Write("<html><body><H2>Sample HTTP Handler</H2></body></html>")
        End Sub
    End Class
End Namespace
```

Next, add the code in the `<system.web>` section of the `web.config` file, as shown in Listing 16.5:

Listing 16.5: Showing the Code for the `web.config` File

```
<compilation debug="true"/>
  <authentication mode="Windows"/>
    <httpHandlers>
        <add verb="*" path="CustomHttp.test"
        type="CustomHTTP.CustomHandler"/>
    </httpHandlers>
```

In the preceding listing, we have specified the path of the file that launches the HTTP Handler, as `CustomHttp.test`. Now, let's run the application. When you run the application, the `Default` page opens in the browser window. You need to type the `CustomHttp.test` filename at the end of the URL and press the ENTER key to initiate the custom HTTP Handler, as shown in Figure 16.4:

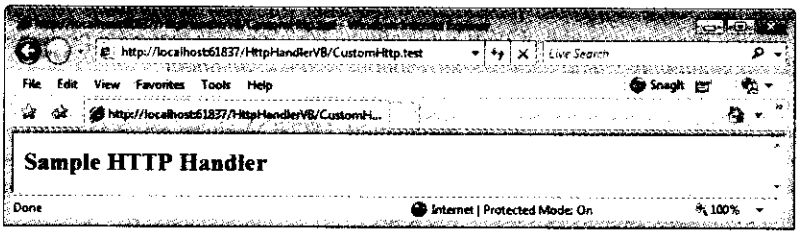


Figure 16.4: Output of the HttpHandler Application

Generic Handlers

HTTP Handlers were somewhat tough to understand and create in the previous versions of Visual Studio such as 2002 and 2003, because these versions did not provide any user-friendly methods for creating the handlers. With Visual Studio 2005, however, things changed and it became easy to create and add HTTP Handlers to an application. All you need to do is simply add the Generic Handler file type in your application. Creating a Generic Handler is the easiest way to create a new HTTP Handler. We learn how to do this in the next section.

Creating a Generic Handler Application

Let's now create an application, named `GenericHandlerVB`, to understand the use of Generic Handlers. You can find the code for the `GenericHandlerVB` application in the Code\ASP.NET\Chapter 16\GenericHandlerVB folder on the CD.

After creating the application, open the Add New Item dialog box by right-clicking the application name in the Solution Explorer window. In the Add New Item dialog box, select the Generic Handler template from the Templates pane, enter the name `GenericHandler.ashx` in the name field, and finally click the Add button, as shown in Figure 16.5:

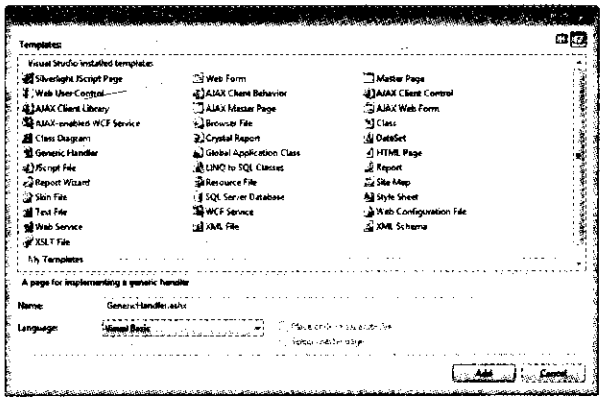


Figure 16.5: The Add New Item Dialog Box

A file with the `.ashx` extension is added to the Solution Explorer window. Now, add the code in the `GenericHandler.ashx` file, as shown in Listing 16.6:

Listing 16.6: Showing the Code for the `GenericHandler.ashx` File

```
<%@ WebHandler Language="VB" Class="GenericHandler" %>
Imports System
Imports System.Web
Public Class GenericHandler : Implements IHttpHandler
  Public Sub ProcessRequest(ByVal context As HttpContext) Implements
   IHttpHandler.ProcessRequest
  ManageForm(context)
   End Sub
   Public Sub ManageForm(ByVal context As HttpContext)
          context.Response.Write("<html><body><form>")
          If context.Request.Params("Feature") Is Nothing Then
          context.Response.Write("<h2>Hello there. Select your favorite feature of ASP .NET
           3.5 </h2>")
          context.Response.Write("<select name='Feature'>")
          context.Response.Write("<option> ASP.NET AJAX</option>")
          context.Response.Write("<option> LINQ </option>")
          context.Response.Write("<option> Silverlight </option>")
          context.Response.Write("<option> WCF </option>")
          context.Response.Write("</select>")
          context.Response.Write("<input type=submit name='Lookup'
          value='Lookup'></input>")
          context.Response.Write("</br>")
          End If
          If context.Request.Params("Feature") IsNot Nothing Then
                context.Response.Write("Hi, you picked: ")
                context.Response.Write(context.Request.Params("Feature"))
                context.Response.Write(" as your favorite feature.</br>")
          End If
          context.Response.Write("</form></body></html>")
   End Sub
   Public ReadOnly Property IsReusable() As Boolean Implements IHttpHandler.IsReusable
          Get
                Return False
          End Get
   End Property
End Class
```

Run the application after adding the code shown in Listing 16.6 in the `GenericHandler.ashx` file. Figure 16.6 shows the output:

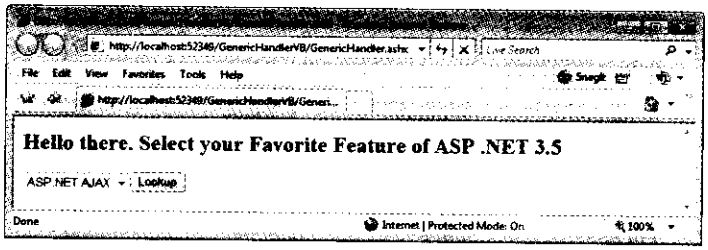


Figure 16.6: Output of the GenericHandler Application

In the Web browser shown in Figure 16.6, when you click the drop-down list, you see a list of ASP.NET 3.5 features. Select a feature of ASP.NET 3.5 from the drop-down list and click the Lookup button. In our case, we have selected the WCF feature of the ASP.NET 3.5, as shown in Figure 16.7:

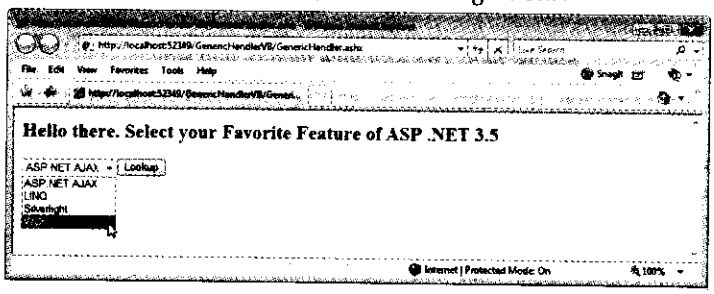


Figure 16.7: Selecting a Feature of ASP.NET 3.5 from Drop-Down List

This displays the selected feature, as shown in Figure 16.8:

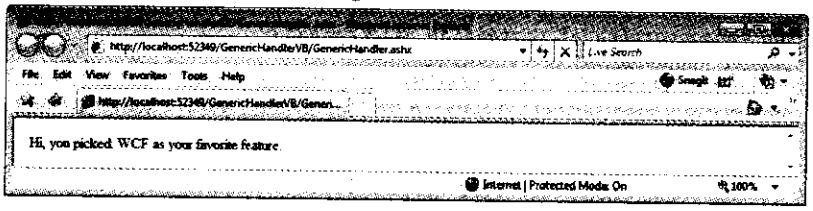


Figure 16.8: Output After Selecting the Feature

After understanding the concept of HTTP Handlers and learning how to create them in an application, we now move on to know postback.

Postback

Postback is the process of sending the data back to the server for processing. This is done to authenticate the login and password of a client or other such tasks that a client cannot perform on its own. ASP.NET provides a rich framework for handling postbacks from ASP.NET Web pages. Postback includes cross-page posting, which is the name given to the process of receiving the response of a request by the server on another page. Now, let's look into the concept of cross-page posting in detail.

Cross-Page Posting

Normally, when we post back data to the server, the server sends the response back to the same page. If we want to receive the response on another page, we use the concept of cross-page posting.

Creating a Cross-Page Posting Application

Let's create an application named `CrossPagePostbacksVB` to understand the working of cross-page posting. You can find the code for the `CrossPagePostbacksVB` application in the Code\ASP.NET\Chapter 16\CrossPagePostbacksVB folder on the CD. Now, follow these steps:

1. Add the code in the `Default.aspx` page of the `CrossPagePostbacksVB` Web application, as shown in Listing 16.7:

Listing 16.7: Showing the Code for the `Default.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Cross-Page Posting</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
  Enter your name:
  <asp:TextBox ID="TextBox1" runat="server" AutoPostBack="True"
   Width="150px"></asp:TextBox>
   <br />
  <br />
  <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
  <br />
  </div>
  <asp:Button ID="Button1" runat="server" Text="Same Page Post Back"
   OnClick="Button1_Click" />
   <asp:Button ID="Button2" runat="server" Text="Cross Page Post Back"
   PostBackUrl="~/Default2.aspx" />
  <p>
  <asp:Label ID="Label1" runat="server"></asp:Label>
  </p>
  </form>
</body>
</html>
```

In the preceding listing, we have added a TextBox control, a Calendar control, two Button controls, and a Label control. The `Butttonl` control is used to postback the data on the same page and the `Button2` control is used to postback the data on another page (cross page). After adding the code in the `Default.aspx` page, the design view of the page appears, as shown in Figure 16.9:

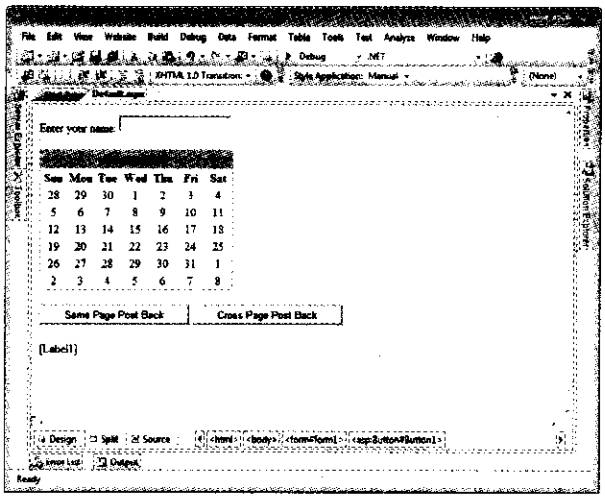


Figure 16.9: The Design View of the Default.aspx Page

Instead of adding the code, you can also add the controls with the help of the Toolbox.

2. Now, add another page named `Default2.aspx` in the application, and add the code in it, as shown in Listing 16.8:

Listing 16.8: Showing the Code for the `Default2.aspx` Page

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default2.aspx.vb"
  Inherits="Default2" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
      <asp:Label ID="Label1" runat="server"></asp:Label></div>
  </form>
</body>
</html>
```

3. Now, double-click the Same Page Postbacks button (Figure 16.9) and add the code shown in Listing 16.9, for the button click event, in the code-behind file of the `Default.aspx` page:

Listing 16.9: Showing the Code for the `Default.aspx.vb` File

```
Partial Class _Default
  Inherits System.Web.UI.Page
  Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   Handles Button1.Click
      Label1.Text = "Hi " + TextBox1.Text + ", here is the output of the Same Page Post
       Back Button: " + Calendar1.SelectedDate.ToString()
  End Sub
End Class
```

4. Now, add the code shown in Listing 16.10, for the page load event in the code-behind file of the `Default2.aspx` page:

Listing 16.10: Showing the Code for the `Default2.aspx.vb` File

```
Partial Class Default2
  Inherits System.Web.UI.Page
  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
   Me.Load
      Dim Calendar1 As Calendar
      Dim TextBox1 As TextBox
      Calendar1 = CType(PreviousPage.FindControl("Calendar1"), Calendar)
      TextBox1 = CType(PreviousPage.FindControl("TextBox1"), TextBox)
      Label1.Text = "Hi " + TextBox1.Text + ", here is the output of the Cross Page
       Post Back Button: " + Calendar1.SelectedDate.ToString()
  End Sub
End Class
```

5. Now, run the application and see the output. However, before you do that, set the `Default.aspx` page as your start up page; otherwise, some errors may occur in the application at runtime. When you run the application, the first page appears in your browser as shown in Figure 16.10:

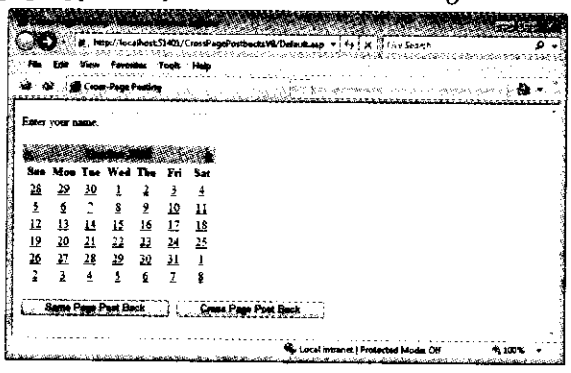


Figure 16.10: First Page of the Application

6. Enter your name in the TextBox control and select the date in the calendar that you want to display. When you click the Same Page Post Back button, the name and the selected date displays on the same page, as shown in Figure 16.11:

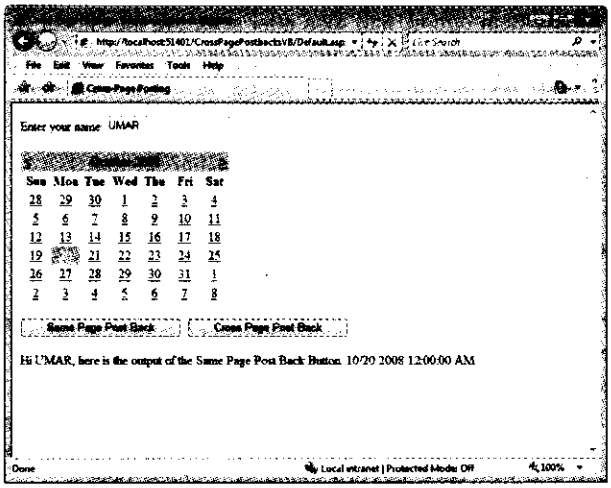


Figure 16.11: Showing Postback on the Same Page

7. To demonstrate cross-page postback, enter your name, select a date, and click the Cross Page Post Back button. The name and selected date displays on another page, as shown in Figure 16.12:

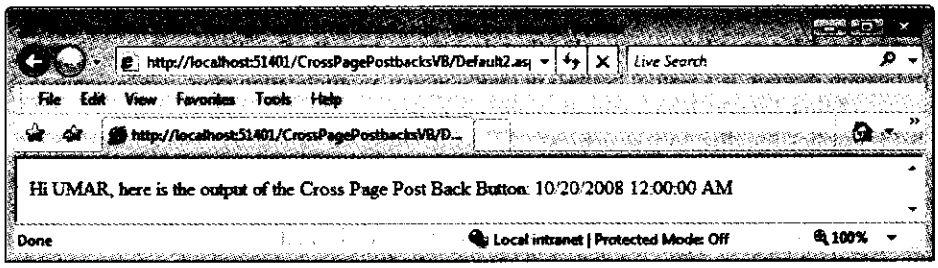


Figure 16.12: Showing Cross-Page Postback

Summary

This chapter discusses some important aspects of Web development in ASP.NET, such as the directory structure of an application and the three states in ASP.NET: application, session, and view. In addition, this chapter also describes the concepts of `Global.asax` file, HTTP Handlers, and postback data.

The next chapter will familiarize you with some common controls, such as Buttons, Textboxes, Labels, and Literals, used for Web programming in ASP.NET 3.5.

Quick Revise

Q1. Which of the following files is knows as ASP.NET application file?

1. **machine.config**
2. **web.config**
3. **Global.asax**
4. **Default.aspx**

Ans: Global.asax

Q2. What is the procedure to add Global.asax file in a Web application?

Ans: To add a Global.asax file in an application, you have to perform following steps:

1. First right-click the name of the website in Solution Explorer.
2. Select the Add New Item option, and then after selecting the Global Application Class template on the Add New Item dialog box and click the Add button.

Q3. Application state uses the .....class to store and share the data throughout the application.

1. **HttpApplicationstate**
2. **HttpApplication**
3. **ViewState**
4. **Application_OnStart**

Ans: HttpApplicationstate

Q4. In which file, the session state is defined? Please also mention the section name.

Ans: Session state is defined in the <sessionState> section of the web.config file.

Q5. What are HTTP Handlers?

Ans: HTTP Handlers handle user requests for Web application resources. They are the backbone of the request-response model of Web applications. For each user request type, there is a specific event handler to handle the request and send back the corresponding response object.

Q6. What is postback?

Ans: Postback is the process of sending the data back to the server for processing. This process is used to authenticate the login and password of a client or other such tasks that a client cannot perform on its own.

Q7. What is the difference between application domain and application lifetime?

Ans: Application domain is a virtual boundary inside which an application runs, while application lifetime is the span of time for which an application domain exists.

Q8. What is application directory structure?

Ans: Application directory structure specifies the directory structure that organizes the various entities associated with an application, such as references, resources, and code files, in separate directories.

Q9. What is the benefit of using the concept of the application domain?

Ans: The benefit of application domains is the independent execution of Web applications. None of the applications can access the in-memory resources of other simultaneously running applications.

Q10. In what situations, the application domain might require a restart?

Ans: The application domain might require a restart under the following situation:

- Excess requests in the queue
- Excess memory usage by an application
- Lengthy lifetime of the application
- Modifications in the web.config file
- Replacement of existing Dynamic Link Libraries (.dll files) or Web files