

Web Development in ASP.NET Learner's Guide

Are you registered with
Onlinevarsity.com?

Yes



No



Did you download this book
from **Onlinevarsity.com**?

Yes



No



Scores

For each **YES** you score **50**

For each **NO** you score **0**

If you score less than 100 this book is illegal.

Register on **www.onlinevarsity.com**

Web Development in ASP.NET

Learner's Guide

© 2013 Aptech Limited

All rights reserved

No part of this book may be reproduced or copied in any form or by any means – graphic, electronic or mechanical, including photocopying, recording, taping, or storing in information retrieval system or sent or transferred without the prior written permission of copyright owner Aptech Limited.

All trademarks acknowledged.

APTECH LIMITED

Contact E-mail: ov-support@onlinevarsity.com

Edition 1 - 2013



Dear Learner,

We congratulate you on your decision to pursue an Aptech Worldwide course.

Aptech Ltd. designs its courses using a sound instructional design model – from conceptualization to execution, incorporating the following key aspects:

- Scanning the user system and needs assessment

Needs assessment is carried out to find the educational and training needs of the learner

Technology trends are regularly scanned and tracked by core teams at Aptech Ltd. TAG* analyzes these on a monthly basis to understand the emerging technology training needs for the Industry.

An annual Industry Recruitment Profile Survey is conducted during August - October to understand the technologies that Industries would be adapting in the next 2 to 3 years. An analysis of these trends & recruitment needs is then carried out to understand the skill requirements for different roles & career opportunities.

The skill requirements are then mapped with the learner profile (user system) to derive the Learning objectives for the different roles.

- Needs analysis and design of curriculum

The Learning objectives are then analyzed and translated into learning tasks. Each learning task or activity is analyzed in terms of knowledge, skills and attitudes that are required to perform that task. Teachers and domain experts do this jointly. These are then grouped in clusters to form the subjects to be covered by the curriculum.

In addition, the society, the teachers, and the industry expect certain knowledge and skills that are related to abilities such as *learning-to-learn, thinking, adaptability, problem solving, positive attitude etc.* These competencies would cover both cognitive and affective domains.

A precedence diagram for the subjects is drawn where the prerequisites for each subject are graphically illustrated. The number of levels in this diagram is determined by the duration of the course in terms of number of semesters etc. Using the precedence diagram and the time duration for each subject, the curriculum is organized.

- Design & development of instructional materials

The content outlines are developed by including additional topics that are required for the completion of the domain and for the logical development of the competencies identified. Evaluation strategy and scheme is developed for the subject. The topics are arranged/organized in a meaningful sequence.

The detailed instructional material – Training aids, Learner material, reference material, project guidelines, etc.- are then developed. Rigorous quality checks are conducted at every stage.

➤ Strategies for delivery of instruction

Careful consideration is given for the integral development of abilities like thinking, problem solving, learning-to-learn etc. by selecting appropriate instructional strategies (training methodology), instructional activities and instructional materials.

The area of IT is fast changing and nebulous. Hence considerable flexibility is provided in the instructional process by specially including creative activities with group interaction between the students and the trainer. The positive aspects of Web based learning –acquiring information, organizing information and acting on the basis of insufficient information are some of the aspects, which are incorporated, in the instructional process.

➤ Assessment of learning

The learning is assessed through different modes – tests, assignments & projects. The assessment system is designed to evaluate the level of knowledge & skills as defined by the learning objectives.

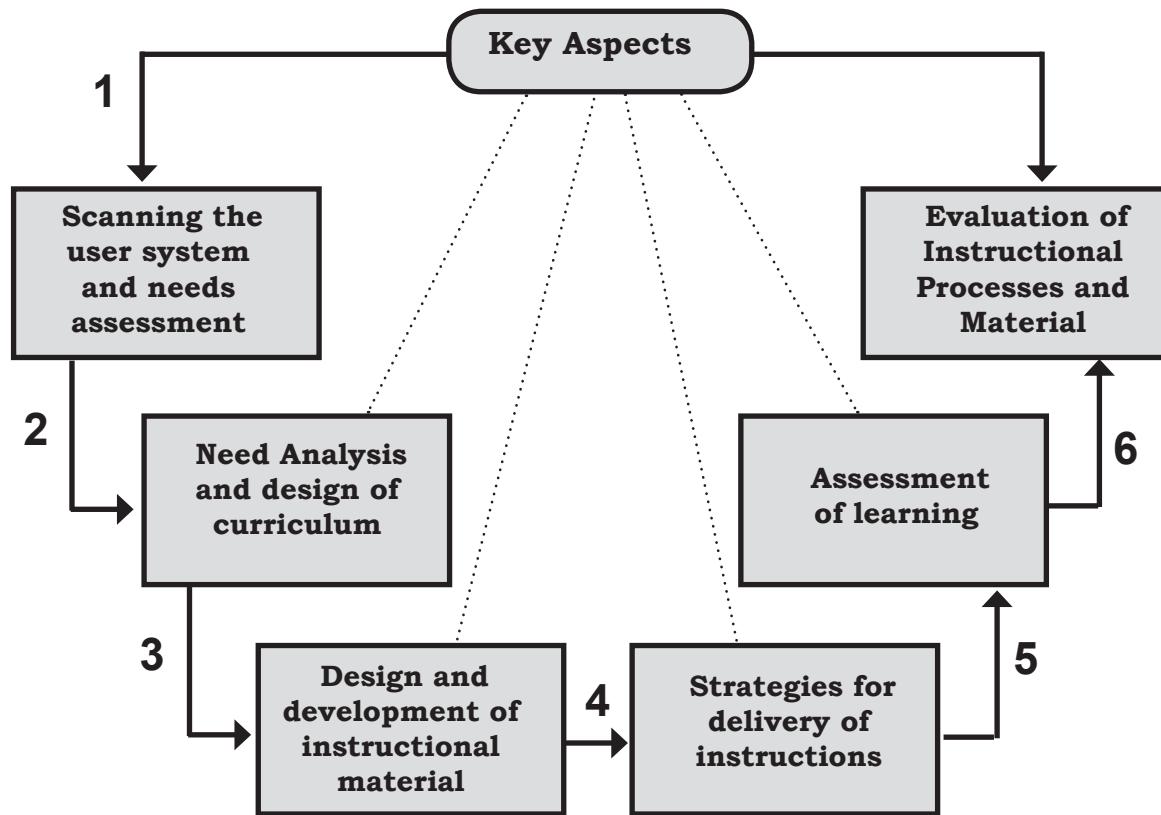
➤ Evaluation of instructional process and instructional materials

The instructional process is backed by an elaborate monitoring system to evaluate - on-time delivery, understanding of a subject module, ability of the instructor to impart learning. As an integral part of this process, we request you to kindly send us your feedback in the reply pre-paid form appended at the end of each module.

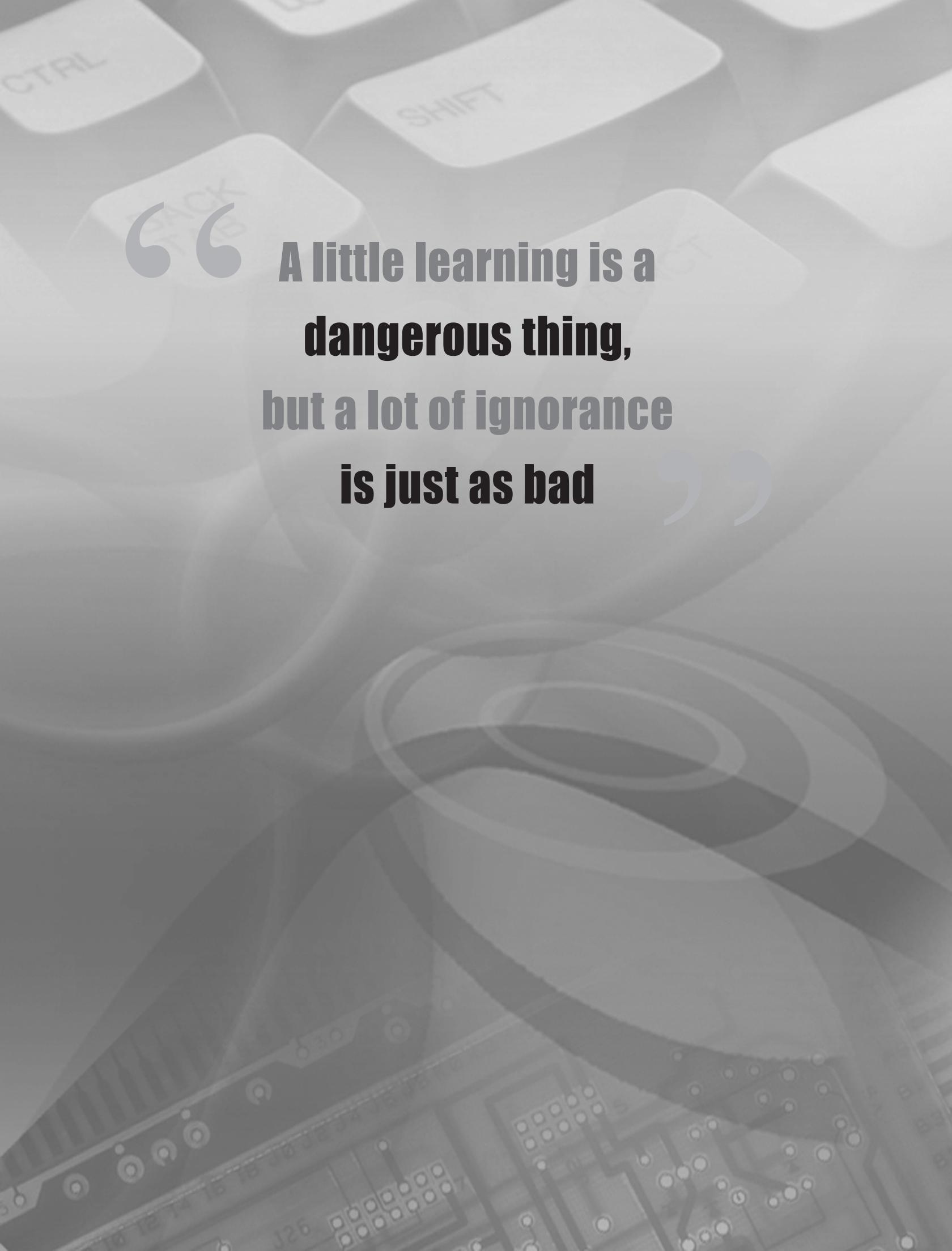
*TAG – Technology & Academics Group comprises of members from Aptech Ltd., professors from reputed Academic Institutions, Senior Managers from Industry, Technical gurus from Software Majors & representatives from regulatory organizations/forums.

Technology heads of Aptech Ltd. meet on a monthly basis to share and evaluate the technology trends. The group interfaces with the representatives of the TAG thrice a year to review and validate the technology and academic directions and endeavors of Aptech Ltd.

Aptech New Products Design Model



“ A little learning is a
dangerous thing,
but a lot of ignorance
is just as bad ”



Preface

Over the last few decades, the Internet and the world of Web applications have seen a lot of changes. Earlier, Web applications could offer only static content with hardly any dynamic elements. Visitors used to just read the information. User interaction was very minimal. Today however, Web applications are extremely dynamic reflecting the changing times and technologies.

Microsoft has yet again conquered developer's hearts by offering ASP.NET for the Web application world. ASP.NET allows Web programmers to create efficient, dynamic Web applications which can seamlessly interoperate with database and Web servers. As ASP.NET has been built for the .NET platform, it can take full advantage of the features offered by the .NET framework. Another advantage of ASP.NET is that it supports .NET languages such as C# and VB.NET. This book uses C# for coding the ASP Web pages.

This book helps beginners in ASP.NET to learn to develop a Web application using ASP.NET and at the same time covers various powerful features of ASP.NET enabling them to build complex Web applications. This book covers an introduction to Web Forms, new features of Web development environment, ASP.NET page model, basic event handling, intrinsic objects in Web applications such as Response, Request, Application and Server. Various categories of Web server controls such as basic Web server controls, selection and validation Web server controls are discussed in depth. The book also covers mobile application development with ASP.NET and explores how to trace and debug Web applications. In ASP.NET 3.5, Microsoft has added several new features and enhanced existing features enabling developers to build powerful, robust, rich, and secure Web applications. The book includes an overview of the new features introduced in ASP.NET 3.5 and Visual Studio 2008 Integrated Development Environment (IDE), and explores various other features such as nested master pages and so forth. The book describes the use of stored procedures and Language Integrated Query (LINQ) with ASP.NET 3.5 to achieve efficient and secure data manipulation. The book concludes with a description of eXtensible Markup Language (XML) Web services and ASP.NET Asynchronous JavaScript and XML (AJAX).

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

We will be glad to receive your suggestions.

Design Team

**“Knowing is not enough
we must apply;
Willing is not enough,
we must do”**

Table of Contents

Modules

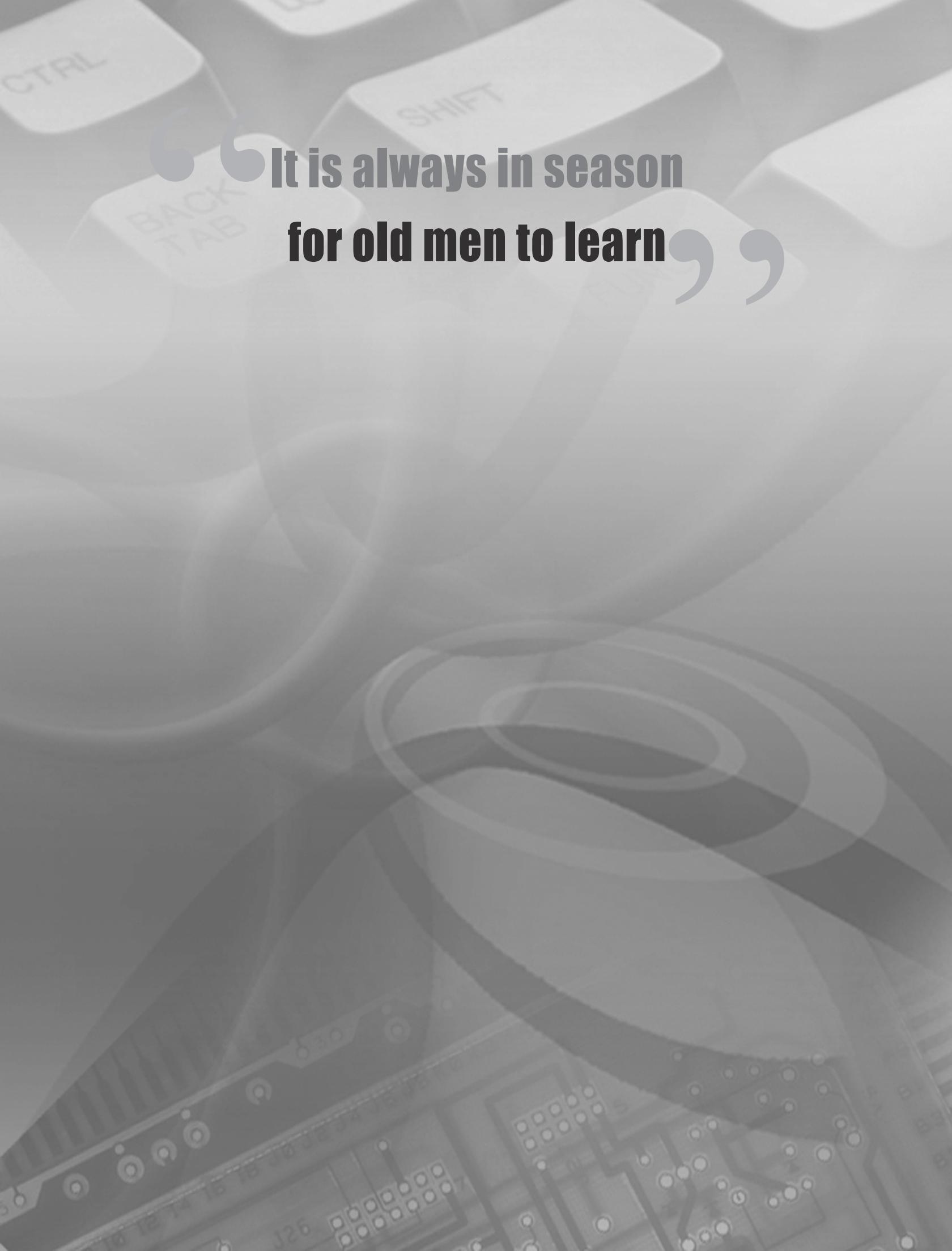
1.	Introduction to ASP.NET	1
2.	Working with the IDE	16
3.	Basics of ASP.NET	29
4.	Request, Response, and Server Objects	48
5.	Basic Web Server Controls	76
6.	More Web Server Controls	96
7.	Advanced Web Controls	127
8.	HTML Server Controls	166
9.	Validation Controls	193
10.	Application, Session, and Cookies	221
11.	An Introduction to Mobile Applications	245
12.	Tracing, Debugging, and Diagnostics	265
13.	Introduction to ASP.NET 3.5	298
14.	Introduction to ASP.NET 3.5 (Lab)	321
15.	Working with ASP.NET 3.5	344
16.	Working with ASP.NET 3.5 (Lab)	362
17.	Authentication and Authorization	384
18.	Authentication and Authorization (Lab)	403
19.	Master Pages and Web Parts	423
20.	Master Pages and Web Parts (Lab)	451
21.	Working with Stored Procedures and LINQ	470

Table of Contents

Modules

- | | | |
|-----|---|-----|
| 22. | Working with Stored Procedures and LINQ (Lab) | 492 |
| 23. | ASP.NET AJAX and XML Web Services | 519 |
| 24. | ASP.NET AJAX and XML Web Services (Lab) | 554 |

**“It is always in season
for old men to learn”**



Module - 1

Introduction to ASP.NET

Welcome to the module **Introduction to ASP.NET**. The module introduces ASP.NET, which is a programming technology based on .NET Framework and is used for creating dynamic Web applications. Developers can create ASP.NET Web applications using any .NET compatible language. ASP.NET simplifies the coding process as it makes use of in-built classes and interfaces of the .NET Framework.

In this module, you will learn about:

- ➔ ASP and ASP.NET
- ➔ Web Forms
- ➔ ASP.NET Application Development

Web Development

http://www



1.1 ASP and ASP.NET

In this first lesson, **ASP and ASP.NET**, you will learn to:

- Outline the features of Active Server Pages.
- List and describe briefly the drawbacks of ASP.
- Describe ASP.NET and its role in Web application development.
- State and describe the advantages and features of ASP.NET 2.0.

1.1.1 Static and Dynamic Web Pages

A Web page is organized as presentation layout and programming code. The layout is created using Hypertext Markup Language (HTML) tags whereas the programming codes are created using scripting languages. Scripting can be either at the client side or at the server side.

→ Client-Side Scripting

HTML tags, by themselves, can create only static Web pages, which provide no interactivity. These tags can, however, be combined with JavaScript to create some extent of interactivity in Web pages. The codes created using JavaScript run only on the client machine and hence, are referred to as client-side scripting. Such code is generally used for validating data input by the user.

Web pages that have only client-side scripts do not provide database connectivity. Hence, there is only limited interactivity possible. JavaScript and VBScript are two of the more widely used client-side scripting languages.

→ Server-Side Scripting

Server-side scripting uses code that runs on the server and can provide database connectivity to Web pages. It also helps in creating dynamic pages having greater interactivity.

Server side scripts can be written separately and attached to the HTML pages by providing the path of the script file in the HTML page. These scripts get compiled only on the server machine.

Active Server Pages (ASP), ColdFusion (CF), Hypertext Preprocessor (PHP), Server Side includes (SSI), and Java Server Pages (JSP) are some of the widely used server-side scripting languages. Of these, ASP is one of the simplest and easiest.

If ASP is the language used to create a server-side script, the file will have an extension of `.asp`.

1.1.2 Features of Active Server Pages

Microsoft developed ASP as a server-side scripting language for creating dynamic Web applications. When a user sends a request to such an application, the ASP code in the application is compiled on the remote server. The server then sends a response to the browser in the form of a dynamically generated HTML page. Figure 1.1 displays the HTTP Request and HTTP Response.

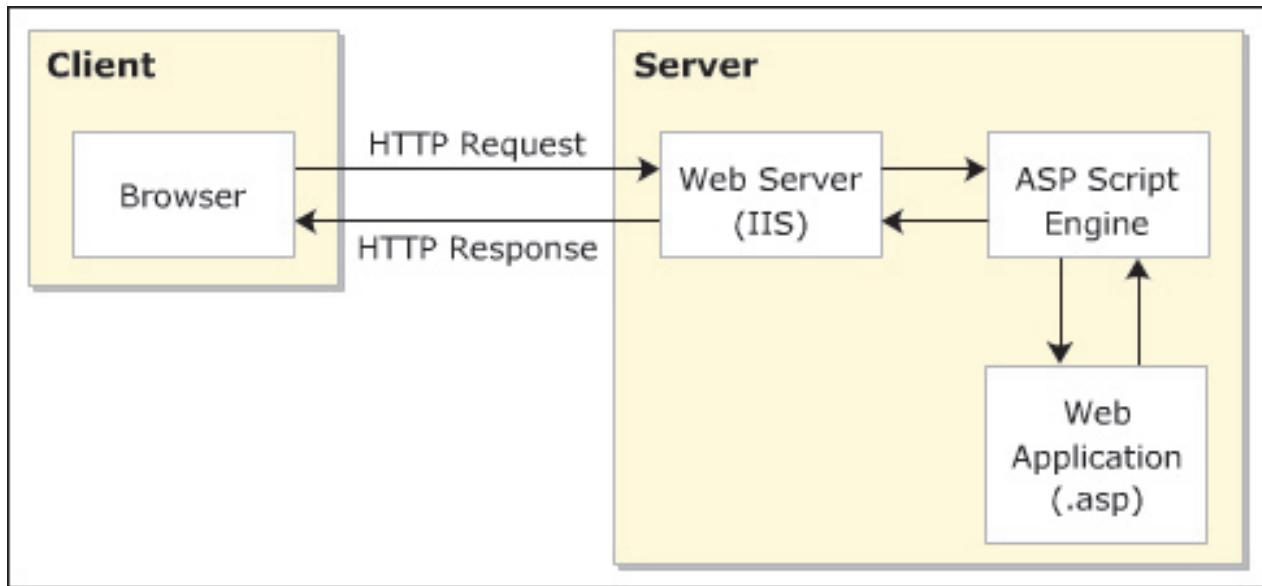


Figure 1.1: HTTP Request and HTTP Response

ASP provides various features useful in creating powerful Web applications. Some of these features are as follows:

→ **Web Page Interactivity**

ASP.NET allows developers to create interactive Web pages wherein users are allowed to submit data to the server and receive response from the server. This feature is useful in creating online surveys, receiving user feedbacks, responding to user queries, and so on.

→ **Database Connectivity**

ASP allows client machines to access and/or update databases on the remote server. This feature is useful in creating Web pages which require data processing such as an online registration application creating a new e-mail ID or checking for availability of tickets and then, booking tickets online.

→ **Source Code Access Restriction**

ASP pages are compiled on the server and only the HTML page is sent to the client. This ensures that the user accessing the Web site is not allowed any access to the ASP source code, thus preventing malicious misuse.

→ Additional Display Support

ASP allows you to display current time, date, month, and year on your Web page.

1.1.3 Drawbacks of ASP

ASP is useful in creating dynamic Web applications, but it also has certain drawbacks. Some of these drawbacks are listed as follows:

- ASP only supports scripts written in JScript and VBScript for client-side scripting. Strongly typed languages like C# and Visual Basic are not supported even in later versions of ASP.
- ASP involves a lot of written code. For example, to display values or to maintain the state of form fields, programmers have to write the entire code; there is no automatic generation of code.
- ASP is based on Component Object Model (COM) and Win32 Application Program Interface (API) technologies. These technologies do not provide support for modern distributed applications.

New technologies were taking a hold in information-sharing and distributed applications using XML. To compete in the market, Microsoft developed the .NET Framework and ASP.NET was created to work on this framework for developing Web applications.

1.1.4 Introduction to .NET Framework

Microsoft.NET is a software platform for developing highly portable Windows and Web applications. The .NET platform provides an application development environment that supports multiple programming languages. It helps in building applications that are easy to develop and that run securely across different operating systems.

ASP.NET is a key part of the .NET Framework and is used to create server-side scripts for Web applications. ASP.NET is not a later version of ASP 3.0 but a completely new technology. Hence, it does not provide any backward compatibility with ASP.

1.1.5 ASP.NET

ASP.NET is a programming technology for creating server-side scripts. It uses Common Language Runtime (CLR) of .NET Framework to make powerful Web applications and XML Web services (which are programs providing application logic to other applications through the Web).

ASP.NET plays a vital role in developing Web applications with .NET; without using ASP.NET, this would not be possible. ASP.NET supports server-side code written in any .NET compatible language such as C#, Visual Basic, and J#. ASP.NET makes use of in-built classes provided specifically for it along with the various in-built classes of the .NET Framework. This reduces the amount of coding required to build large event-driven applications. Hence, it is easier to write ASP.NET code and maintain ASP.NET pages on the server.

Figure 1.2 displays the architecture of .NET Framework.

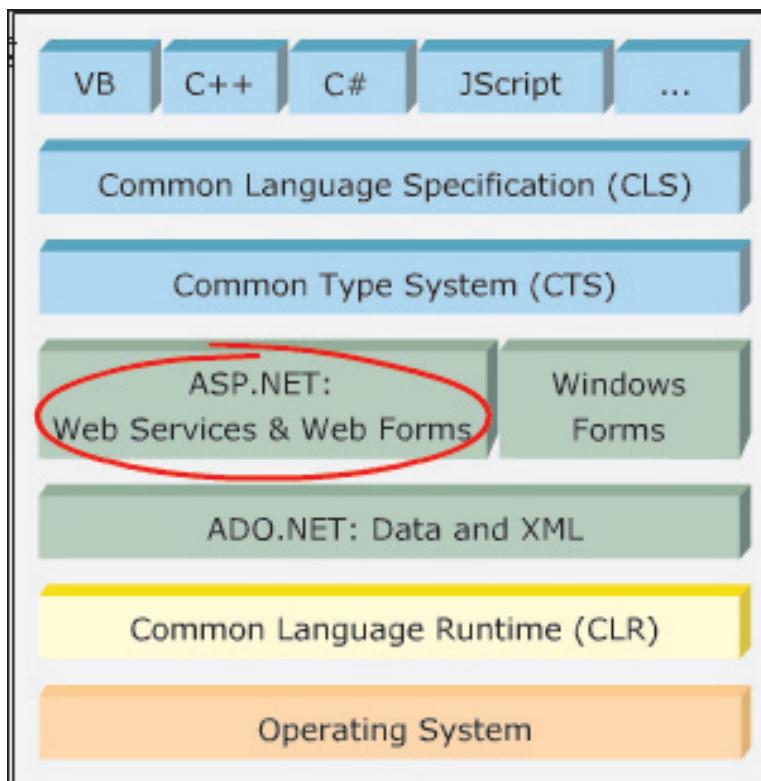


Figure 1.2: Architecture of .NET Framework

1.1.6 Features of ASP.NET 2.0

ASP.NET is a combination of improved performance, reliability, and ease of deployment. It is much easier for new developers to create Web applications due to the following features of ASP.NET:

→ Simple Coding Model

ASP.NET has access to the in-built classes of .NET Framework library. Also, the WYSIWYG (What You See Is What You Get) editor provided by .NET Framework auto-generates code when components are dragged and dropped on the design view of the ASP.NET page. All this makes coding easier for creating Web applications.

→ Multiple Language Options

ASP.NET allows developers to select a language that is suitable for their applications. ASP.NET supports more than 25 languages that can be used for building a Web application.

→ Dynamic Compilation

ASP.NET compiles dynamically whenever any change is detected in the application code. This feature increases the compilation speed and updates the application on a timely basis.

→ Caching Functionality

The practice of storing a copy of an accessed image or a page in the memory is referred to as caching. ASP.NET output caching gives better performance and allows ASP.NET applications to be more scalable. When an ASP.NET page is requested for the first time, the page is stored in the server's cache memory. If another user requests the same ASP.NET page, the page does not get re-compiled and is directly displayed from the cache.

→ Web-Farm Session State

Multiple servers hosting the same site are referred to as a Web farm. The more frequently accessed Web sites use Web farms to improve availability. ASP.NET session state enables users to share state values across machines in the Web farm. This enables users to make requests to multiple servers in the Web farm but still have full access to the previous sessions.

→ Reliability Features

ASP.NET has a number of reliability features such as protection against memory leaks, deadlocks, and crashes. ASP.NET can automatically detect memory leaks and deadlocks in an application. A memory leak is a bug in a program that prevents it from releasing unwanted memory. If an application has a memory leak, then ASP.NET replaces the old copy of the ASP.NET worker process with a new one once all pending requests have been processed. This releases the unwanted memory.

1.1.7 Advantages of ASP.NET 2.0

ASP.NET has various advantages over other server-side scripting technologies, thus making it very popular with Web developers. Some of these advantages are listed as follows:

→ Improved Performance

ASP.NET is compiled on the CLR and does not require an interpreter. This, along with the caching and dynamic compiling features of ASP.NET, greatly improves the performance of an ASP.NET application.

→ Power and Flexibility

The .NET Framework class library and data access solutions are available to ASP.NET as it is based on the CLR. Due to the interoperability of the CLR, any COM-based development is preserved while moving to ASP.NET. Furthermore, ASP.NET is language-independent; hence, Web applications can be created using any .NET-supported language.

→ Manageability

ASP.NET makes it easier to perform tasks like authenticating users, collecting data from the user, and site configuration as it separates the application logic from the HTML code. ASP.NET files are easier to manage on the server as changes can be made in the files without restarting the server.

→ **Scalability**

ASP.NET is designed with scalability in order to improve the performance in a multiprocessor environment. ASP.NET runtime keeps a check on all the processes that are running on the server as it immediately creates a new process when the existing process faces deadlock state.

→ **Customizability and Extensibility**

ASP.NET gives developers the support to modify code at any level of application development. It is also possible to add a component at run-time or replace any component with a custom-made component.

→ **Security**

ASP.NET provides in-built security for the code. It helps in protecting code against unauthorized modification.

Knowledge Check 1

1. Which of the following statements regarding ASP and ASP.NET are true?

(A)	ASP supports languages like C# and J#.		
(B)	The .NET Framework supports cross-language compatibility.		
(C)	An ASP page contains server-side code and can help in displaying the current year.		
(D)	An ASP page can work with COM and ActiveX components.		
(E)	ASP.NET is a programming language that is primarily used for creating XML applications.		

(A)	A, B, D	(C)	B, C, D
(B)	A, C	(D)	A, B, C

2. Can you match the advantages of ASP.NET with their corresponding descriptions?

Descriptions			Advantages
(A)	Improves performance as the code is compiled only once on the server.	(1)	Security
(B)	Protects the code against unauthorized modification.	(2)	Customizability
(C)	Helps in performing tasks like authenticating users.	(3)	Power and Flexibility
(D)	Enables modifying code at any level of application development.	(4)	Enhanced Performance
(E)	Allows selection of the most appropriate language for the application.	(5)	Simplicity and Manageability

(A)	(A)-(4), (B)-(1), (C)-(5), (D)-(2), (E)-(3)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(1), (C)-(5), (D)-(2), (E)-(4)	(D)	(A)-(1), (B)-(2), (C)-(5), (D)-(4), (E)-(3)

1.2 Web Forms

In this second lesson, **Web Forms**, you will learn to:

- Outline the features of Active Server Pages.
- List and describe briefly the drawbacks of ASP.

Imagine a Web application that is created for automating the tasks of a supermarket. Consider that the user interface of the application is like a form with some fields or controls on the screen. These fields and controls are used to accept the items the user has bought and will have code logic to process the user input. In ASP.NET, such a user interface is called a Web Forms page.

1.2.1 Description of Web Forms

Web Forms are part of ASP.NET and provide a page programming model or framework. Web Forms are similar to Windows Forms but are accessible only through a Web browser. Web Forms can contain HTML code, client-side scripting in languages like JavaScript or VBScript, as well as the server-side application logic. A Web Form page is derived from the `System.Web.UI.Page` class. Figure 1.3 displays the hierarchy of System namespaces.

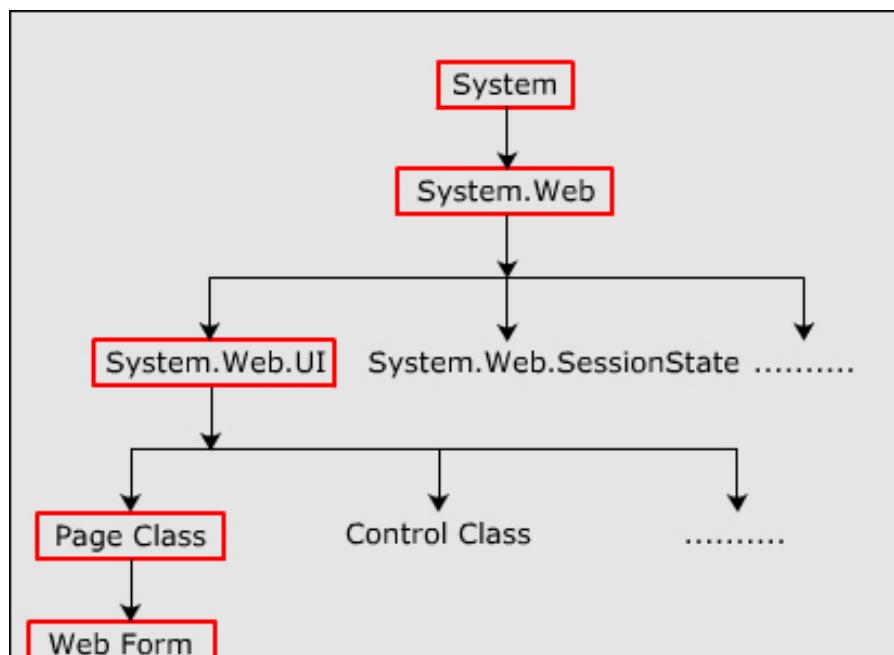


Figure 1.3: Hierarchy of Web

→ Uses of Web Forms

The following are the important uses of Web Forms:

- It is easier to develop Web applications using Web Forms as the code for the HTML page is auto-generated when components are dragged and dropped on the form.

- Web Forms allow the separation of the HTML and client-side codes from the application logic. This makes the compiling of the Web application faster as only the ASP.NET page is sent to the server for compiling when requested by the user.
- Web Forms provide user interactivity by using components like text box and buttons on the form. Also, multiple Web Forms can be attached to a single Web Form, making it possible to call one form from another. Figure 1.4 displays a Web form.

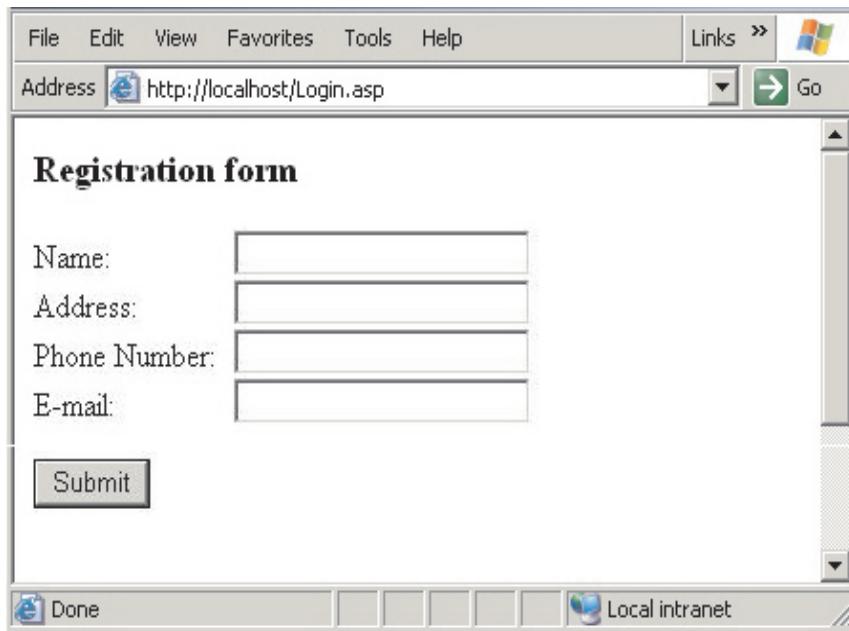


Figure 1.4: Web Form

1.2.2 Components of Web Forms

Various components can be incorporated into a Web Form to provide different functionalities. These components can be broadly classified into the following four categories:

→ Server Controls

These controls run event procedures on the server in response to user events. Server controls have in-built features to save data entered by the user. These controls are used in the creation of the user interface. `TextBox`, `Label`, `Button`, `ListBox`, `DropDownList`, `DataGrid`, `Hyperlink`, `AdRotator`, and `Calendar` are some of the commonly used server controls.

→ HTML Controls

These controls are the standard visual components provided by HTML. `TextArea`, `Table`, `Image`, `Submit`, `Reset`, `Checkbox`, and `Radio` are some of the commonly used HTML controls.

→ **Data Controls**

These controls are used for connecting to the database. They help in retrieving or updating data and performing commands on SQL or OLE databases or XML data files. `GridView`, `DataList`, `FormView`, `Repeater`, and `SqlDataSource` are some of the commonly used data controls.

Figure 1.5 displays the types of controls.

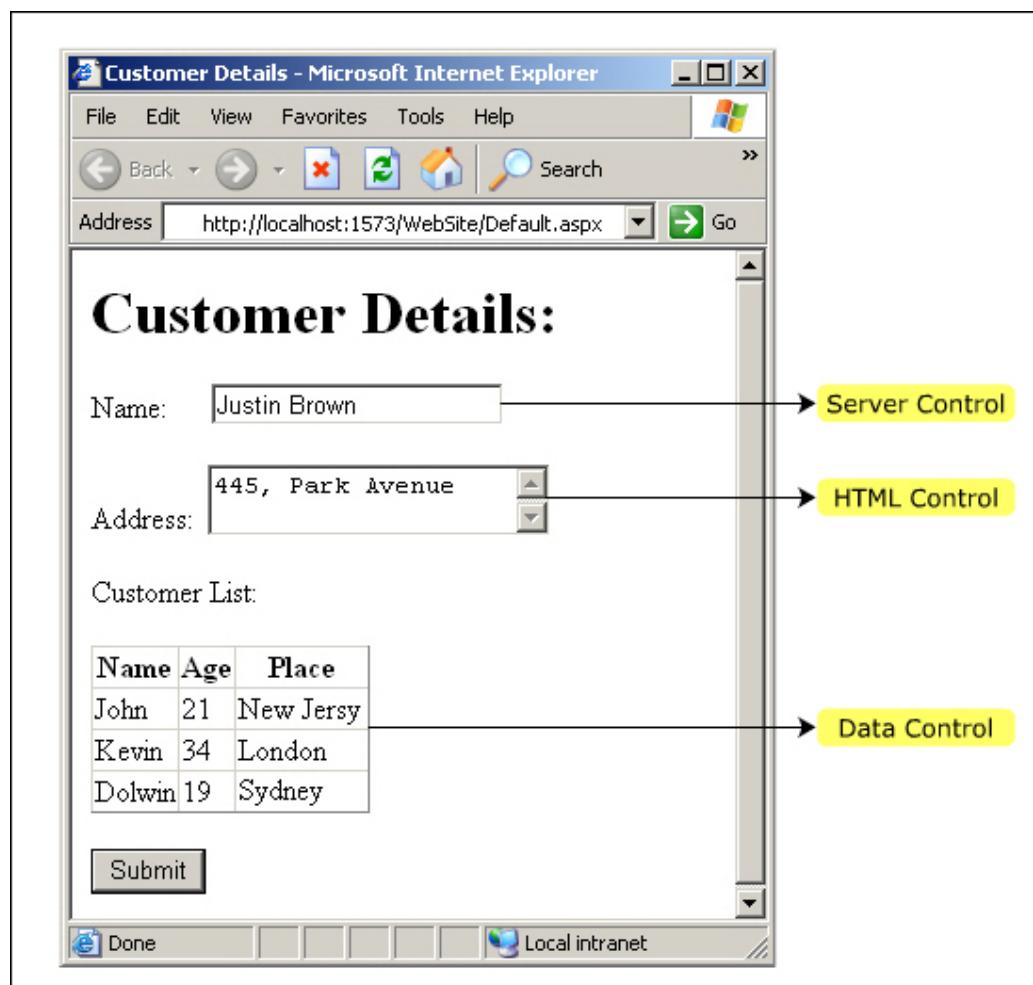


Figure 1.5: Types of Controls

1.2.3 Features of Web Forms

ASP.NET Web Forms provide various features to assist developers in the coding process. Following are some of the important features of Web Forms:

→ **Rendering**

Upon compilation of the applications, Web Forms can be rendered (generated) automatically in any browser. Also, they can be adjusted to take maximum advantages of the features of a particular browser.

→ Programming

The code for Web Forms can be written in any .NET compatible language. Also, the ASP.NET code can directly make Win32 API calls.

→ WYSIWYG

Web Forms are created using the WYSIWYG (what you see is what you get) editor, provided by the .NET Framework. This editor allows developers to drag and drop controls onto the Web Form.

→ Code Separation

Web Forms help the developer in separating the server side logic from the HTML code.

→ State Management

The state management feature of Web Forms allows you to maintain the view-state of various controls between calls to the Web page.

→ Extensibility

Web Forms can incorporate extra components like user controls and mobile controls, thus providing extensibility to the Web Forms.

Knowledge Check 2

- Which of the following statements about Web Forms are true?

(A)	Web Forms can be used for making Web applications that have several controls.
(B)	Web Forms allow you to separate client-side and server-side scripts.
(C)	Web Forms are created by deriving from the <code>Form</code> class that belongs to the <code>System.Web.UI</code> namespace.
(D)	System components in ASP.NET are used for connecting to the database.
(E)	Server controls are used for creating a user interface on the Web Form.

(A)	A, B, D	(C)	B, C, D
(B)	A, C	(D)	A, B, E

2. Can you match the features of Web Forms with their respective descriptions?

Descriptions			Feature
(A)	Helps in coding in any .NET Framework language.	(1)	Rendering
(B)	Helps in maintaining the state between the Web Forms.	(2)	Code Separation
(C)	Helps in adding extra components to the Web Form.	(3)	State Management
(D)	Helps in rendering the Web Form on any browser.	(4)	Extensibility
(E)	Helps in separating the application logic from the HTML code.	(5)	Programming

(A)	(A)-(5), (B)-(3), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(1), (C)-(5), (D)-(2), (E)-(4)	(D)	(A)-(1), (B)-(2), (C)-(5), (D)-(4), (E)-(3)

1.3 ASP.NET Application Development

In this last lesson, **ASP.NET Application Development**, you will learn to:

- Outline the step-by-step procedure to create a simple ASP.NET application.
- List and explain the project files created as part of a Web Forms application.

1.3.1 Steps to Create an ASP.NET Application

ASP.NET is used to create interactive Web applications. You can create a simple Web application by performing a number of steps in sequence.

→ Create a new Web Application

In order to begin creating a Web application, you will launch **Microsoft Visual Studio 2005** and create a new ASP.NET application. Instead of choosing the new project option, you will choose the **Web Site...** option after selecting **New** option from the **File** menu. This will display the **New Web Site** dialog box.

→ Select a Location for the Web Site

Before you actually create an ASP.NET application, you need to decide where you want to place it. There are three choices available for this: on the hard disk (this is done by selecting the **File System** option in the **Choose Location** dialog box), on a network Web server (by using the **FTP option**) or on the Web server (by using the **HTTP** option). After selecting the appropriate location, enter a name for your new Web application in the provided text box.

→ Select the Coding Language

You will now choose the programming language for creating your Web application using the Language drop-down menu. After selecting the appropriate **language**, you will click **OK**. Your new Web application will now be created.

1.3.2 Default Project Files

Whenever an ASP.NET application is created, there are some project files that are created by default. Following is the list of these default project files:

→ AssemblyInfo.cs

The assembly information file is the building block of an ASP.NET application. This file contains the information about the assembly, its version, and the company name. If the ASP.NET code is written in VB instead of C#, the assembly file created is `AssemblyInfo.cs`.

→ Projectname.vsdico

The `Projectname.vsdico` is an XML file that contains the links (URLs) to provide information about the XML Web service. This file cannot be seen in the Solution Explorer window.

→ Default.aspx

The `Default.aspx` file contains the design interface and the client-side code.

→ Default.aspx.cs

The `Default.aspx.cs` file contains the server-side code. This file, by default, cannot be seen in the Solution Explorer window. However, it can be seen if you click '+' symbol in front of the `.aspx` file name.

1.3.3 Optional Project Files

Additional project files, other than the ones created by default, may be optionally generated based on the requirements of the application. These files are referred to as the optional project files. Some of the optional project files are listed as follows:

→ Global.asax

The `Global.asax` file is created in the root directory of the project. This file contains the information about the events that are raised by the Web application when the application is loaded or unloaded. If this file is not created, then it is assumed that there are no application or session handlers in the application.

→ **Web.Config**

The `Web.config` file is an XML file that contains the information needed by the Web server to process the ASP.NET file. Different settings regarding authentication, security, tracing, and error reporting are mentioned in this file.

→ **StyleSheet.css**

The `StyleSheet.css` file is a Cascading Style Sheet (CSS) file that provides the information about the styles for the Web pages. This style sheet is used for customizing the appearance of Web pages to be customized.

Knowledge Check 3

1. Can you match the project files created as part of a Web Forms application with their corresponding descriptions?

Descriptions		Project Files	
(A)	Contains the events of the Web application when it is loaded or unloaded.	(1)	<code>Web.Config</code>
(B)	Provides freedom to select styles and implement them in Web pages.	(2)	<code>WebForm1.aspx.cs</code>
(C)	Contains settings regarding authentication, security, and so on.	(3)	<code>AssemblyInfo.cs</code>
(D)	Contains information about the assembly version and company name.	(4)	<code>Global.asax</code>
(E)	Contains the client-side code.	(5)	<code>Style.css</code>

(A)	(A)-(5), (B)-(3), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(1), (C)-(5), (D)-(2), (E)-(4)	(D)	(A)-(4), (B)-(5), (C)-(1), (D)-(3), (E)-(2)

2. Which of the following statements for creating an ASP.NET application are true and which statements are false?

(A)	You will select the 'New' option from the 'File' menu and choose 'Project....'
(B)	You will choose 'Visual C#' from the 'Language' drop-down menu and click 'OK'.
(C)	You will launch 'Microsoft Visual Studio 2005'.
(D)	You will always select the 'FTP' option from the 'Location' drop-down menu.
(E)	You will select the 'New' option from the 'File' menu and choose 'File...'.

(A)	A, B, D	(C)	B, C
(B)	A, C	(D)	A, B, E

Module Summary

In this module, **Introduction to ASP.NET**, you learnt about:

→ **ASP and ASP.NET**

ASP is a server-side scripting language that helps to create interactive Web pages. ASP.NET is a Web application development technology built on .NET Framework. Developers can create powerful Web applications using ASP.NET.

→ **Web Forms**

Web Forms help in separating the application logic from the HTML code. They use the WYSIWYG editor provided by .NET Framework to auto-generate code by dragging and dropping components on the design view of the form. Web Forms provide features like rendering, code separation, and extensibility to assist in coding and compiling the application.

→ **ASP.NET Application Development**

Default project files are automatically generated whenever a new Web application is created. These default files are `AssemblyInfo.cs`, `Projectname.vsdisco`, `Default.aspx`, and `Default.aspx.cs`. Other than these default files, certain optional files are created that are application specific. These are `Global.asax`, `Web.Config`, and `StyleSheet.css`.

Module - 2

Working with the IDE

Welcome to the module, **Working with the IDE**. Visual Studio 2005 Integrated Development Environment (IDE) is a platform for developing .NET based software applications. The Web development environment of the IDE helps in developing Web applications and provides various tools to assist in the development process. The IDE also consists of an in-built Web browser that proves useful while working with Web applications.

In this module, you will learn about:

- ➔ Working with the Visual Studio 2005 IDE
- ➔ Configuring ASP.NET Applications with IIS
- ➔ Features of the New Web Development Environment

Web Development

http://www



2.1 Working with the Visual Studio 2005 IDE

In this first lesson, **Working with the Visual Studio 2005 IDE**, you will learn to:

- Outline the features of Visual Studio 2005 IDE specific to ASP.NET.
- Identify the use of the Web Browser in the Visual Studio 2005 IDE.

2.1.1 IDE Features for Web Development

ASP.NET Web applications can be created using Visual Studio 2005 Integrated Development Environment (IDE). The IDE is a set of tools for developing, debugging, and deploying software applications created using the .NET Framework. The IDE simplifies the application development process by providing various features such as simplified Web page designing, and localization.

→ Simplified Web Page Designing

Visual Studio 2005 IDE provides a design editor that gives an effect almost like What You See Is What You Get (WYSIWYG) to simplify the creation of a Web page layout. It provides the facility to drag and drop components on the Web Form. The HTML code for this is automatically generated. After the components or controls have been added to the screen, they can be aligned or placed in appropriate positions as desired. The IDE provides support for this too.

→ Localization

With the advent of the Internet, the world is getting smaller. People from all over the world access numerous Web sites. Localization means presenting your Web site content in a way that it is accessible to people from different cultures and languages. It mainly involves translation of the user interface and changing the local attributes such as date and time format.

Visual Studio 2005 IDE supports localization by providing tools to generate the resource files used for localizing the Web application. The content and data based on the preferred language settings are saved in the resource files with extension `.resx`. During runtime, the resource files of the default language are replaced with the resource files for the preferred language. Using in-built classes of the .NET Framework, the Web application translates the Web site content to the preferred language at runtime.

→ Visual Web Developer

The Visual Web developer is a set of software tools that assists you in the development of ASP.NET Web sites. The software tools include code editor, controls for data binding, data source controls, and many more. The Visual Web Developer is integrated with the Visual Studio 2005 IDE. It provides support for creating IIS applications on local as well as remote computers, for using FTP for opening Web sites, and using standalone files from outside the project.

2.1.2 Advantages

Visual Studio .NET 2005 IDE provides more advanced features and capabilities as compared to the previous IDEs for Visual Studio that were released by Microsoft. Some of the advantages of Visual Studio 2005 IDE over Visual Studio 2003 IDE with respect to Web development are as follows:

→ **Easy Configuration**

You do not have to manually configure the IIS virtual directory in Visual Studio 2005 IDE because it has an in-built Web server and the ASP.NET 2.0 is integrated with the IIS snap-in (which is a window to configure Web site settings in IIS).

→ **Automatic File Compilation**

When you compile an assembly created for a Web application, Visual Studio 2005 IDE automatically compiles all files present in the assembly. Visual Studio 2003 IDE compiles only certain file types, which include ASP.NET Web pages, HTTP handlers, resource files, and Global.asax files. You had to manually compile the remaining files.

→ **Flexible Iterative Web development**

In the new Web editing model of Visual Studio 2005 IDE, you can save and refresh the Web application to view the changes made in it. In Visual Studio 2003 IDE, the Web project had to be rebuilt and recompiled to view the changes.

→ **New Objects and Methods**

Visual Studio 2005 IDE includes many new objects and methods as compared to Visual Studio 2003 IDE that simplify the programming process.

2.1.3 Web Browser

Visual Studio 2005 integrates an in-built Web browser with the IDE to allow browsing from within the IDE. The Web browser is basically a version of Internet Explorer hosted within the IDE itself.

The in-built Web browser assists developers in testing and debugging ASP.NET Web applications. After compiling and executing the Web applications, developers can view the output in the browser and then, debug the code accordingly.

It also allows developers to browse through the Internet, intranet, FTP sites, and Microsoft Developer Network (MSDN) Library Help, as well as maintain a Favorites list.

Figure 2.1 displays the Web Browser.

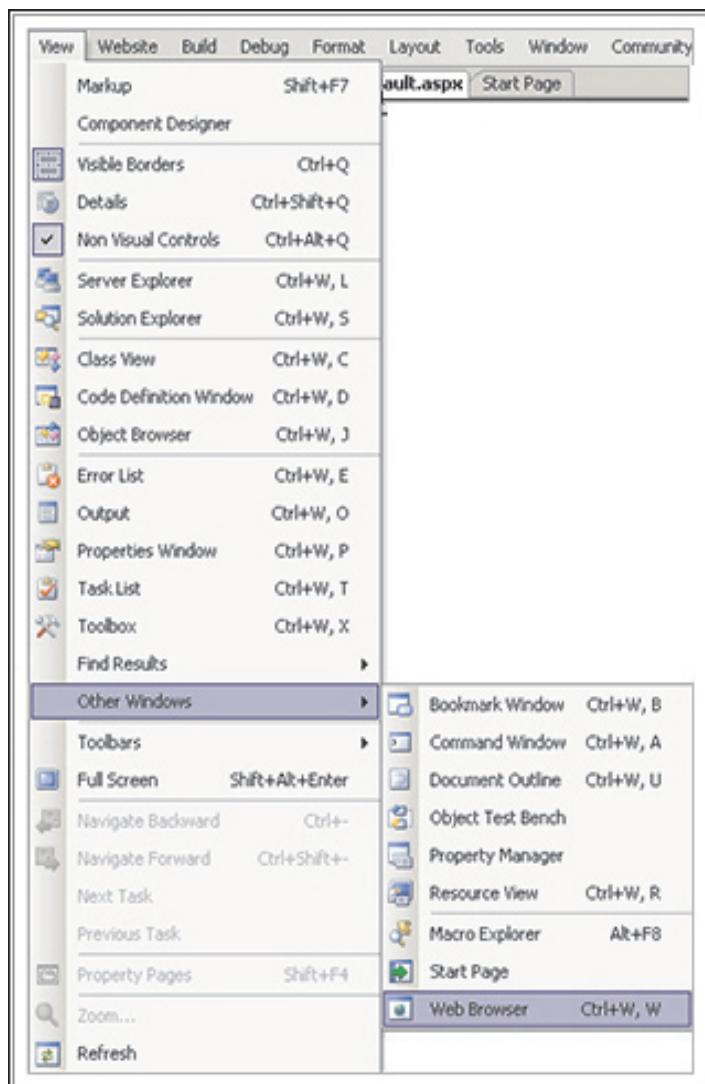


Figure 2.1: Web Browser

In order to set the in-built browser as the default, you will need to use the **File** menu and **Browse With** option and change the default settings. Once the in-built browser is set as the default browser, whenever you execute an ASP.NET Web application and select the **View in Browser** option, it will use the in-built browser.

Knowledge Check 1

1. Which of the following statements about Visual Studio 2005 IDE and Web browser in the IDE are false?

(A)	Visual Studio 2005 IDE does not require external resources such as image files to be included with the project.		
(B)	Visual Studio 2003 IDE automatically compiles all the files that are present within the assembly when the Web application is compiled.		
(C)	The Visual Web Developer provides option to use standalone files from outside the project.		
(D)	The WYSIWYG design editor allows dragging and dropping of components onto the Web form.		
(E)	The in-built Web browser of the IDE does not allow viewing the output of the application code.		

(A)	A, B, D	(C)	B, E
(B)	A, C	(D)	A, B, E

2. Match the components of Visual Studio 2005 IDE and Web browser in the IDE against their corresponding descriptions.

Descriptions		Component	
(A)	Contains data based on preferred language settings.	(1)	Visual Studio 2003 IDE
(B)	Provides facility to drag and drop components onto the Web form.	(2)	Integrated Web browser
(C)	Provides supporting tools for creating Web sites.	(3)	WYSIWYG Editor
(D)	Requires rebuilding and recompiling Web projects to view changes.	(4)	Resource files
(E)	Allows access to MSDN Library Help.	(5)	Visual Web Developer

(A)	(A)-(5), (B)-(3), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(1), (C)-(5), (D)-(2), (E)-(4)	(D)	(A)-(4), (B)-(3), (C)-(5), (D)-(1), (E)-(2)

2.2 Configuring ASP.NET Applications with IIS

In this second lesson, **Configuring ASP.NET Applications with IIS**, you will learn to:

- List the features of IIS and describe its role in Web application development.
- Explain the steps to configure IIS for ASP.NET applications.

2.2.1 Internet Information Server (IIS)

IIS is a Web server developed by Microsoft to host one or more Web sites on a single server. The latest version of IIS is 6.0 and it supports technologies like ASP, ASP.NET, XML, and Simple Object Access Protocol (SOAP).

IIS also allows sending e-mails using the Simple Mail Transfer Protocol (SMTP).

In ASP.NET 2.0, the IIS snap-in (console) is integrated with the ASP.NET Microsoft Management Console (MMC) snap-in. The MMC snap-in for ASP.NET allows you to manipulate the configuration settings such as selecting the ASP.NET version for applications.

Most organizations use IIS to publish and manage their Web sites on the Internet or intranet. Figure 2.2 displays the IIS model.

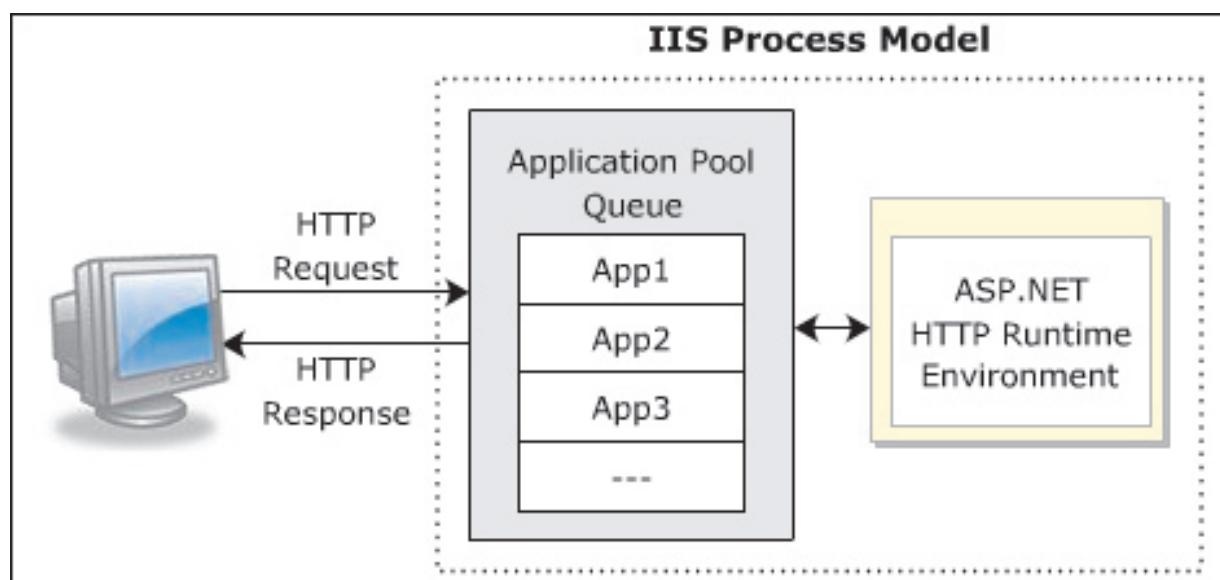


Figure 2.2: IIS Model

2.2.2 Features of IIS

IIS provides a flexible and strong communication platform for running applications over multiple networks.

IIS has the many features which make an ASP.NET Web application:

→ **Reliable**

IIS allows a Web application to become reliable by providing an application isolation environment. The application isolation environment allows each Web application to function individually. Application isolation environment is a reliable environment for running Web applications as it avoids one Web application from interfering with another.

→ **Scalable**

Scalability is the ability to add more systems to the network without loss of performance. The performance is in terms of its speed, simultaneous processing of the requests, and so on.

→ **Secure**

IIS provides advanced authentication and authorization features to protect the system from unauthorized access. It also supports other security features such as encryption and certificates.

→ **Manageable**

IIS provides a set of administrative and manageability tools such as administration scripts and IIS Manager. These tools allow the administrators to configure the IIS according to the organization's needs.

2.2.3 Working of IIS with ASP.NET Applications

ASP.NET applications are stored in the virtual directories of the IIS server. IIS contains the ASP.NET script engine that executes the server scripts within the application. Figure 2.3 displays the working of IIS.

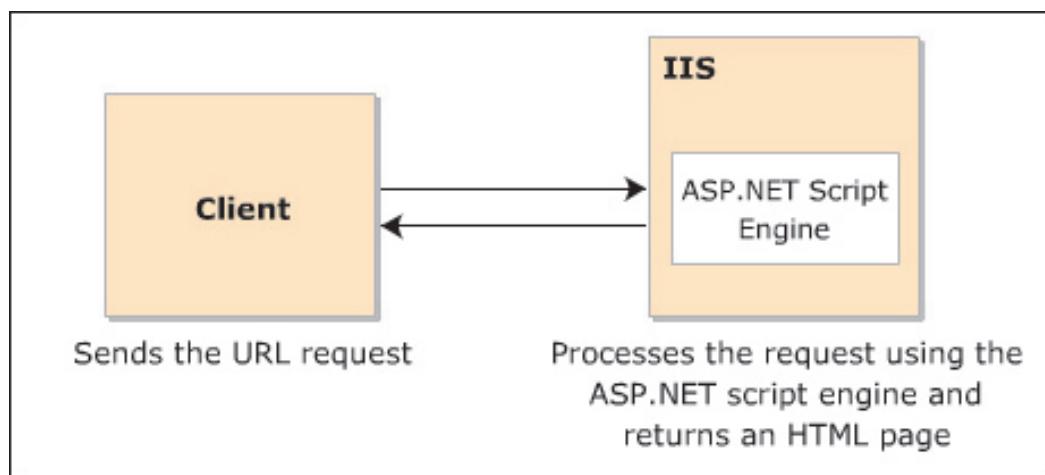


Figure 2.3: Working of IIS

The steps explaining the communication between ASP.NET application and the IIS are as follows:

1. The browser sends a request to IIS.

2. IIS receives the request, searches for the appropriate file, and then sends the file to the ASP.NET Script Engine.
3. The script engine executes the server-side scripts, dynamically generates a HTML page, and sends it to IIS.
4. IIS receives the HTML page and sends it back to the browser.

2.2.4 Configuring IIS

To publish or deploy ASP.NET applications on an IIS server, you have to first configure IIS for ASP.NET. To configure the IIS for ASP.NET 2.0 applications, you will first open the IIS Manager. You will then right-click the **Default Web Site** option and select **Virtual Directory**. This will bring you to the **Virtual Directory Creation** wizard. You will then follow the instructions in the wizard and click **Finish**. The Virtual directory will be created and it will be displayed in the IIS Manager. You will now right-click the newly created virtual directory and select **Properties**. In the **ASP.NET** tab, you will now select the **2.0.50727** version of ASP.NET from the drop-down box.

You can also configure IIS with ASP.NET 2.0 applications by running the `aspnet_regiis` tool. Since the `aspnet_regiis` tool is different for different versions of ASP.NET, the `aspnet_regiis` tool for the appropriate version of ASP.NET should be run to configure the IIS.

2.2.5 Specifying an IIS Web Project

The IIS Web project needs to be specified for creating IIS based Web applications. In order to do this, you use the **File** menu, **New** option in Visual Studio 2005, and choose the **New Web Site** option and then, select the location as **HTTP (>>http://localhost/WebSite)**. This will create the project as an IIS based Web application.

Knowledge Check 2

1. Which of the following statements for configuring IIS for ASP.NET applications are false?

(A)	You will right-click Default Web Site from the IIS Manager and select the Refresh option.
(B)	You will follow the instructions in the Virtual Directory Creation wizard and click Finish .
(C)	You will select the 2.0.50727 version of ASP.NET from the drop-down box in the Directory Security tab.
(D)	You will right-click the newly created virtual directory and select Properties .
(E)	You will select Options from the File menu in the IIS Manager.

(A)	A, B, D	(C)	B, C
(B)	A, C	(D)	A, C, E

2.3 Features of the New Web Development Environment

In this last lesson, **Features of the New Web Development Environment**, you will learn to:

- Identify and describe the new features of Web development environment.
- Describe coding aids for ASP.NET.
- Explain Compile-on-Demand feature in ASP.NET 2.0.

2.3.1 Web Development Environment

The Visual Studio 2005 IDE provides two types of development environments, namely, the application development environment and the Web development environment.

The application development environment is used for developing standalone applications whereas the Web development environment is used for developing Web applications and Web services. Web applications are intended to run over the Internet, intranet, and FTP sites. Web services are reusable components used for interaction between different applications running over the Web.

The Web development environment of the Visual Studio 2005 IDE allows you to easily develop, edit, test, and debug Web applications and Web services.

2.3.2 Features

Web development environment of the IDE provides a platform for developing Web sites. Following are some of the important new features of the Web development environment:

→ Dynamic Application Compilation

In the compilation model used in earlier versions, the executable code for the entire application was compiled into a single assembly. If changes were made to any page, the entire application needed to be recompiled. In the dynamic compilation model, if a page is changed, only that page is recompiled when it is next requested.

→ Secure Web Site Publishing

The Visual Web developer allows pre-compilation of Web sites before building them. The pre-compilation and deployment is done using the **Build Web Site** option in the **Build** menu of Web development environment. Pre-compiling the Web site only deploys the executable code, thus securing the source code. It also generates any errors during the compile time.

→ Easy Programming and Debugging

The code editor equally supports the code-behind and single-file page models with features like IntelliSense and syntax coloration.

Also, there are complete debugging facilities for local Web sites. Debugging in the current Web development environment is much simpler and faster than in previous versions.

→ Improved Code Behind Model

In previous versions, the markup and code were stored in separate files. These were organized using the instance variables and event delegates making it difficult to work with the files separately. However, in the new version, the code is truly separated, allowing the developers to work separately on the two codes.

2.3.3 Coding Aids

Consider a professional runner who uses special running shoes to improve his lap times. These shoes provide better shock absorption at the heels to reduce the stress on his ankles and knees. They also provide better lift-off at the toe-end to reduce the stress on his muscles. All this translates into faster speed and better lap times. Although the athlete can run without using the special shoes, they help to improve his speed and performance.

Similar to the running shoes that assist the runner to improve his speed, coding aids in Visual Studio 2005 assist developers in the coding process.

The Web development environment of the IDE provides developers with various tools that help in making the coding process easier and faster. These tools are referred to as coding aids.

2.3.4 Different Coding Aids

Coding aids save the developer's time as well as effort in the process of developing the Web application. Some of the important coding aids for ASP.NET are as follows:

→ Code Editor

Visual Studio 2005 IDE provides a code editor that supports the IntelliSense feature. When you start typing code, Intellisense displays a list of possible classes, methods, and controls that you can include in your code. You simply select the appropriate item from the list to automatically include it in your code without actually typing it.

→ Toolbox

The toolbox contains various controls that can be added to the form by simply dragging and dropping the controls onto the Design view. This reduces the time and effort required in writing the code to create the control. Text box, DataGridView, DataSet, DataList, and CrystalReportViewer are some of the controls in the toolbox.

→ Class View

Class View displays all the classes and methods used in the project. The upper pane of this view displays classes and namespaces.

The lower pane consists of members such as properties and methods. The search feature is incorporated in Class View, which lets you search for a specific class.

→ **Object Browser**

Object Browser consists of three panes. One pane displays all objects in the Web development environment, another displays all members of the selected object, and the third describes the selected member. Using the Object Browser, you can view the list and description of the different classes and their objects and namespaces.

→ **CSS Styles**

Web page appearance can be customized by creating different styles in the cascading style sheet. These styles can then be applied globally to the different elements of the Web page. HTML server controls and Web server controls support CSS styles. You can use the CSS styles to easily maintain the uniformity in the Web pages.

→ **Framesets**

Framesets allow a Web page to be divided into multiple frames and these frames can be refreshed independent of each other. Consider a sports-based Web site displaying a scoreboard in a corner on the home page. Using framesets, the developer can cause the scoreboard to be dynamically refreshed without having the user refresh the entire page.

2.3.5 Compile-on-Demand

ASP.NET 2.0 provides the Compile-on-Demand feature that makes the debugging process easier for developers. Once a page has been compiled, if there are any changes made to the code, the developer does not have to explicitly recompile the page. Whenever the compiled version of the page is demanded by an application, IIS checks whether any changes have been made to the source code since the last compilation. If changes are detected, then the page is automatically recompiled and the earlier compiled version is discarded.

Dynamic compilation is applicable not just to ASP.NET pages but can also be used for Web services, user controls, HTTP handlers, and ASP.NET application files such as `Global.asax`.

The Compile-on-Demand feature is extended to many file types including class files and resource files.

Knowledge Check 3

1. Which of these following statements about Compile-on-Demand functionality and Coding Aids are false?

(A)	Compile-on-Demand feature simplifies debugging.
(B)	Class View displays descriptions of members of selected objects.
(C)	Framesets allows refreshing portions of a Web page.
(D)	CSS Styles allows customizing the appearance of a Web page.
(E)	The WYSIWYG editor supports the IntelliSense feature.

(A)	A, B, D	(C)	B, E
(B)	A, C	(D)	A, C, E

Module Summary

In this module, **Working with the IDE**, you learnt about:

→ **Working with the Visual Studio 2005 IDE**

Visual Studio 2005 IDE simplifies layout designing by providing the drag-and-drop feature through the WYSIWYG editor. The IDE also integrates an in-built Web browser to allow browsing from within the IDE. The Visual Web Developer provides support for creating IIS applications on local as well as remote computers.

→ **Configuring ASP.NET Applications with IIS**

ASP.NET 2.0 applications can use IIS as a Web server. IIS is very powerful and is used extensively to host and publish Web sites. You can configure IIS for ASP.NET 2.0 by using the IIS manager window (snap-in).

→ **Features of the new Web Development Environment**

The Web development environment of the IDE provides various tools and coding aids to help develop ASP.NET Web applications. The Compile-on-Demand feature of ASP.NET 2.0 simplifies the debugging process for developers.

Module - 3

Basics of ASP.NET

Welcome to the module, **Basics of ASP.NET**. This module begins with an introduction to the ASP.NET Web page model. The Web page model allows you to include the markup and programming code of an ASP.NET application either in a single file or in separate files. The markup file of an ASP.NET application consists of directives that tell the compiler how an ASP.NET page will be compiled and processed. The programming code of an ASP.NET application includes the application logic that consists of different event-handling techniques.

In this module, you will learn about:

- ➔ ASP.NET Web Page Code Model
- ➔ Directive Syntax
- ➔ Basic Event Handling in Web Pages

Web Development

http://www



3.1 ASP.NET Web Page Code Model

In this first lesson, **ASP.NET Web Page Code Model**, you will learn to:

- Describe the single-file page model.
- Describe the code-behind page model.
- Explain smart navigation in ASP.NET 2.0.
- Describe the use of code-behind files in Web applications.

3.1.1 Web Page Model

An ASP.NET Web page is created using markup and programming codes. When you open a Web page, you see the layout of the page on your browser screen. This layout is designed using markup tags. The programming code is written for Web pages to handle user interactivity or perform some actions.

The markup and the programming codes can either be in the same file or in different files. Based on this, there are two types of Web page models, namely the single-file page model and the code-behind page model.

The basic structure of the .aspx file for both the models has a common layout that consists of the page directives, script code, and the user interface code.

3.1.2 Single-File Page Model

The single-file page model is a Web page model that allows you to include both, the markup as well as the programming code, in a single .aspx file. This model is generally used for codes that are not very lengthy. For example, a single-file page model can be used for an application containing a couple of text boxes to enter user details and a submit button. Using single-file page model has its advantages as well as limitations.

→ Advantages

The single-file page model for Web page development is as follows:

- Easier to maintain since the markup and the programming code are both in one place.
- Easier to deploy since there is only one file.
- Easier to rename the single-file page since there is no dependency between files.

→ Limitations

The single-file page model has various limitations. Some of these are as follows:

- A single-file page cannot be directly created in Visual Studio IDE.

- The HTML editor that is used to create single-file pages has limited coding support as compared to the code editor.
- The event-handler cannot be created by double-clicking the control's event. You need to manually associate the events with the event handlers.

The following code demonstrates a single-file page model.

Code Snippet:

```
<%@ Page Language="C#" %>

<script runat="server">
    private void ShowMessage(object sender, EventArgs e)
    {
        lblStatus.Text = "Welcome, " + txtName.Text;
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Welcome...</title>
</head>
<body>
    <form id="frmInfo" runat="server">
        <div>
            <h1>Welcome to ASP .NET</h1>
            <asp:Label ID="lblName" Text="Enter Name :" runat="server" />
            <asp:TextBox ID="txtName" Text="" runat="server" />
            <asp:Button ID="btnSubmit" Text="Submit" OnClick="ShowMessage"
runat="server" />
            <br />
            <asp:Label ID="lblStatus" Text="" runat="server" />
        </div>
    </form>
</body>
</html>
```

The markup and programming code are included in the same file. The markup code designs a Web page consisting of a text box, two labels, and a button control. The programming code is written in the form of a method to display a message when the button is clicked.

3.1.3 Code-Behind Page Model

Code-behind page model is a Web page model that separates the markup and the programming code. When creating a new ASP.NET application, the Visual Studio 2005 IDE provides you with an option of placing the two codes in different files. If you select this option, the IDE creates two files, a markup file and a class file.

The markup file is saved with the extension `.aspx` and it stores the markup code used to design Web pages. The class file is the code-behind file that defines the programming logic and this file is saved with the extension `.aspx.cs`. The markup file contains a reference to the code-behind file. Thus, the two files are linked.

Visual Studio 2003 lacked the option to choose a particular Web page model and the default model always used to be the code-behind model.

3.1.4 Page Class

When you create an ASP.NET Web page, it is derived from the `Page` class. The `Page` class is a base class and it resides in the `System.Web.UI` namespace. This allows the Web page to inherit all the methods, properties, and events of the `Page` class.

→ Properties

Table 3.1 lists the properties of the `Page` class.

Property	Description
<code>ID</code>	Specifies or retrieves an identifier for an object (instance) of the <code>Page</code> class.
<code>Title</code>	Specifies or retrieves the title for the page.
<code>Server</code>	Retrieves an instance of the <code>Server</code> class, which is an instance of the <code>HttpServerUtility</code> class.
<code>Session</code>	Retrieves an instance of the <code>Session</code> class representing the current session.
<code>Controls</code>	Retrieves an instance of the <code>ControlCollection</code> class for any server control (which is a server side component).
<code>ErrorPage</code>	Specifies or retrieves an error page to which the requested ASP.NET page is redirected in case an exception occurs.

Table 3.1: Page Class Properties

→ Methods

Table 3.2 lists the methods of the `Page` class.

Method	Description
<code>HasControls</code>	Checks whether the server control consists of any child controls.
<code>LoadControl</code>	Loads an instance of the <code>Control</code> class.
<code>GetValidators</code>	Returns a set of validation objects that are associated with the specified validation group.
<code>MapPath</code>	Returns the path that a given virtual path maps to.
<code>Validate</code>	Instructs the controls to validate the information.

Table 3.2: Page Class Methods

The following code demonstrates how the properties of the `Page` class are used.

Code Snippet:

```
...
Page.Title = "Login";
Page.ErrorPage = "LoginPage.aspx";
...
```

The code uses the `Title` property of the `Page` class to specify the title of the Web page as `Login`. If any exception occurs, the `ErrorPage` property redirects the Web page to the `LoginPage.aspx` page.

The following code demonstrates how the methods of the `Page` class are used.

Code Snippet:

```
...
if (Page.HasControls())
{
    Response.Write("Welcome");
}
...
```

The code uses the `HasControls()` method that checks whether any child controls are present within a server control.

3.1.5 Partial Classes in Code-Behind Model

Code-behind files automatically creates a partial class. The partial class declaration consists of a `partial` keyword, which means that the class does not contain the complete implementation. This class is inherited from the `Page` class.

The partial class, by default, only consists of the necessary application code that generally includes event handlers. However, you can include different properties and methods to make up the full class for the Web page. Figure 3.1 displays the code.

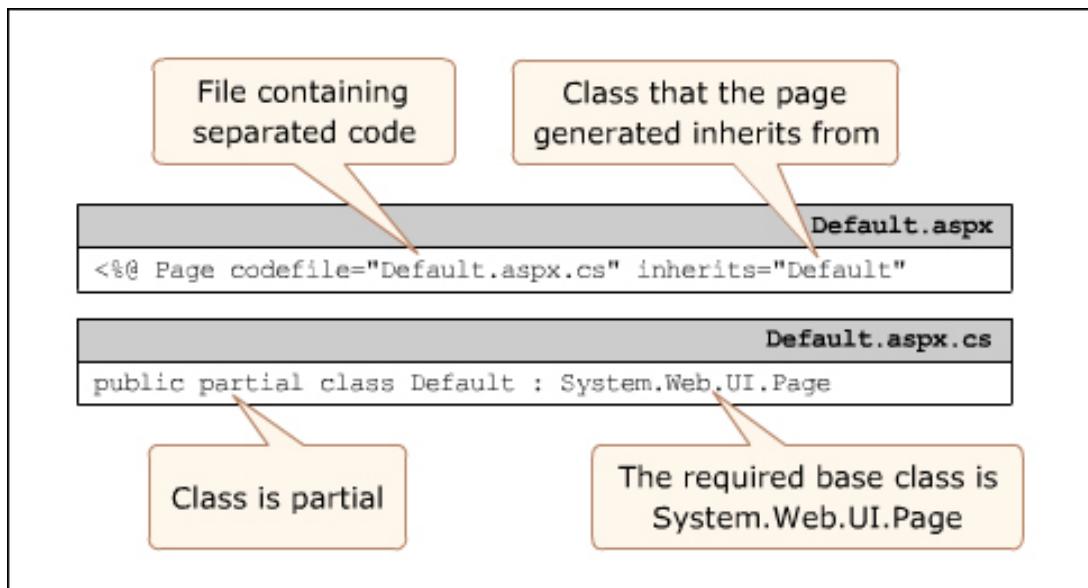


Figure 3.1: Code

3.1.6 Smart Navigation Feature

The scroll position or the focus of any element or control on a Web page changes whenever you access different pages or when you refresh the page. For example, when you submit a Web page, such as a page containing login details, to the server, a text box in the page might lose focus. To avoid this, you can use the smart navigation feature provided by ASP.NET.

Smart navigation feature allows you to retain the focus for an element or control even when a Web page is refreshed or a different Web page is accessed. This feature also maintains only the last page visit in the browser's history, which prevents the user from going back to the previously visited sites. Smart navigation feature also minimizes the flash that you might observe while navigating between different Web pages.

3.1.7 Implementing Smart Navigation

Smart navigation feature was implemented by using the `SmartNavigation` property. However, Microsoft has now deprecated the `SmartNavigation` property, which means it will be no longer used.

In ASP.NET 2.0, you can implement the smart navigation feature by using the `SetFocus()` method and `MaintainScrollPositionOnPostBack` property. The `SetFocus()` method maintains the focus of the element on a Web page while the `MaintainScrollPositionOnPostBack` property retains the scroll position on the Web page.

3.1.8 Advantages of Code-Behind Files

Code-behind files provide several advantages over single files.

Some of these are as follows:

- ➔ Improved presentation of code by separating the markup and programming code.
- ➔ Reusability of code across multiple pages.
- ➔ Hiding of application logic from the Web page designers or from outsiders.
- ➔ Elimination of problems arising due to browser incompatibility. This is because the file contains the programming code resides in the server and not in the browser.

3.1.9 Implementing Code-Behind Files

Code-behind page model implements multiple inheritance hierarchy. The code-behind class contained in the .aspx.cs file inherits from the Page class and the .aspx file inherits from the .aspx.cs file. This means all the properties, methods, and events of the Page class are available to the code-behind class and all the properties, methods, and events of the .aspx.cs file are available to the .aspx file.

Code-behind files can be implemented using the `Codebehind` and `Inherits` attributes of the `@ Page` directive. Directives are instructions given to the compiler that specify how an ASP.NET Web page should be compiled. The `Codebehind` and `Inherits` attributes are defined in the .aspx file. The `CodeFile` attribute specifies the reference of the code-behind file that is associated with the .aspx page. The `Inherits` attribute specifies the name of the code-behind class from which the .aspx file inherits.

Knowledge Check 1

- Which of the following statements about the Web page models are true?

(A)	Single-file page is saved with the extension .aspx.cs.		
(B)	Code-behind page model is used to eliminate the browser incompatibility problems		
(C)	The partial class is created automatically to include the markup code.		
(D)	Single-file page codes are created in Code Editor.		
(E)	Code-behind page model is used to create reusable code that can be used across multiple pages.		

(A)	A, B, D	(C)	B, E
(B)	A, C	(D)	A, B, E

2. Can you match the methods and properties of the Page class against their corresponding description?

Descriptions			Component
(A)	Verifies whether the server control contains any child controls.	(1)	GetValidators method
(B)	Retrieves the object of the ControlCollection class.	(2)	LoadControl method
(C)	Retrieves and returns a collection of validation objects.	(3)	ID property
(D)	Defines an identifier for an object of the Page class.	(4)	HasControls method
(E)	Loads an object of the Control class.	(5)	Controls property

(A)	(A)-(5), (B)-(3), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(1), (C)-(5), (D)-(2), (E)-(4)	(D)	(A)-(4), (B)-(5), (C)-(1), (D)-(3), (E)-(2)

3.2 Directive Syntax

In this second lesson, **Directive Syntax**, you will learn to:

- List and describe the popularly used directives in ASP.NET.
- State the syntax and use of the @ Page directive.
- State the syntax and use of the @ Import directive.

3.2.1 Directives

Directives are commands that describe how an ASP.NET application is compiled and processed. In ASP.NET, the directives are enclosed within <% %>. The directives are included in the .aspx file of the ASP.NET Web application. These directives instruct the compiler to process the .aspx and .aspx.cs files.

Table 3.3 lists the commonly used directives in ASP.NET.

Directive	Description	Example
@ Page	Defines various attributes for a Web page.	<%@ Page Language = "C#" AutoEventWireup = "true" %>
@ Import	Imports a namespace to a page that allows you to include the classes and interfaces within the namespace.	<%@ Import namespace = "System.Net" %>
@ Assembly	Associates an assembly to a page or any control.	<%@ AssemblyName = "MyAssembly" %>

@ Master	Defines various attributes for a master page, which is saved with the .master extension.	<%@ Master Language="C#" CodeFile="Login.master.cs" Inherits="Login" %>
@ Reference	Associates a page, control to the current page.	<%@ Reference Page="Information.aspx" %>

Table 3.3: Directives

3.2.2 @ Page Directive

The @ Page directive allows you to specify different attributes for a page. These attributes will be assigned values according to which the ASP.NET page will be compiled. The commonly used attributes of the @Page directive are as follows:

→ Language

The Language attribute specifies the language used while compiling the code.

→ CodeFile

The CodeFile attribute specifies the path for the code-behind file.

→ Inherits

The Inherits attribute specifies the name of the code-behind class. This attribute is used with the CodeFile attribute.

→ ErrorPage

The ErrorPage attribute specifies the URL for redirection in case any exception occurs.

→ Title

The Title attribute specifies the title for a Web page.

→ AutoEventWireUp

The AutoEventWireUp attribute allows automatic binding of events to the methods of the same .aspx file or the code-behind file. It takes a Boolean value and its default value is true.

→ MasterPageFile

The MasterPageFile attribute specifies the path of the master page file that is to be used in the ASP.NET application.

The following syntax demonstrates the use of the @ Page directive.

Syntax:

```
<%@ Page <attribute>="<value>" %>
```

where,

<attribute>: Defines the attribute applied to the page.

<value>: Specifies the value assigned to the attribute.

→ Language

The following code demonstrates the use of the Language attribute.

```
<%@ Page Language="C#" %>
```

The code uses the Language attribute to specify the programming language as C#.

→ CodeFile

The following code demonstrates the use of the CodeFile attributes.

```
<%@ Page Language="C#" CodeFile="Sample.aspx.cs" Inherits="Sample" %>
```

The code uses the CodeFile attribute to specify the file name of the code-behind file as Sample.aspx.cs.

→ Inherits

The following code demonstrates the use of the Inherits attributes.

```
<%@ Page Language="C#" CodeFile="Sample.aspx.cs" Inherits="Sample" %>
```

The Inherits attribute specifies the name of the code-behind class as Sample.

→ ErrorPage

The following code demonstrates the use of the ErrorPage attribute.

```
<%@ Page Language="C#" ErrorPage="ErrorPage.aspx" %>
```

The code uses the ErrorPage attribute that redirects the Web page to the ErrorPage.aspx page if any exception occurs.

→ Title

The following code demonstrates the use of the Title attribute.

```
<%@ Page Language="C#" AutoEventWireup="true" %>
```

The code uses the Title attribute that specifies the title for a Web page as Sample Page Title.

→ AutoEventWireUp

The following code demonstrates the use of the `AutoEventWireUp` attribute.

```
<%@ Page Language="C#" AutoEventWireup="true" %>
```

The code uses the `AutoEventWireUp` attribute that binds the events to the methods of the same `.aspx` file or the code-behind file.

→ MasterPageFile

The following code demonstrates the use of the `MasterPageFile` attribute.

```
<%@ Page Language="C#" MasterPageFile="Master1.master" %>
```

The code uses the `MasterPageFile` attribute specifies the name of the master page file as `Master1.master`.

3.2.3 @ Import Directive

The `@ Import` directive allows you to explicitly include in your Web page different functionalities that are declared in other namespaces. The `@ Import` directive allows one attribute, `namespace`. This attribute allows you to import a particular namespace in your ASP.NET Web page. The namespace can be either system-defined (.NET framework) or user-defined.

When you import a namespace in your ASP.NET page, you are making all the classes and interfaces of that namespace available to your ASP.NET Web page. Figure 3.2 displays the `Import` directive.



Figure 3.2 Import Directive

The following syntax demonstrates the use of the `@ Import` directive.

Syntax:

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

where,

`<value>` : Specifies the value assigned to the namespace.

The following code demonstrates how to import a system-defined namespace and a user-defined namespace.

Code Snippet:

```
<%@ Import namespace="System.IO" %>
<%@ Import namespace="Grocery" %>
```

The code uses the `@ Import` directive, which imports the classes and interfaces of the specified namespace. It imports the system-defined namespace, `System.IO`, and the user-defined namespace, `Grocery`.

Knowledge Check 2

1. Which of the following statements about directives in ASP.NET are true?

(A)	@ Reference directive associates a page or control to the current ASP.NET Web page.			
(B)	@ Import directive imports only system-defined namespaces.			
(C)	@ Master directive specifies various attributes for a master page saved with the .aspx extension.			
(D)	A directive indicates how an ASP.NET application is compiled and processed.			
(E)	@ Page directive specifies various attributes for a Web page.			

(A)	A, D, E	(C)	B, E
(B)	A, C	(D)	A, B, E

2. Can you match the attributes of the `@ Page` directive against their corresponding descriptions?

	Descriptions		Component
(A)	Defines the name of the code-behind class.	(1)	MasterPageFile
(B)	Defines the URL for redirection in case any exception occurs.	(2)	AutoEventWireUp
(C)	Defines the path that refers to the code-behind file.	(3)	Inherits
(D)	Binds events to the methods of the same .aspx file or the code-behind file.	(4)	ErrorMessage
(E)	Defines the location of the master page file to be used in the ASP.NET application.	(5)	CodeFile

(A)	(A)-(5), (B)-(3), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(4), (C)-(5), (D)-(2), (E)-(1)	(D)	(A)-(4), (B)-(5), (C)-(1), (D)-(3), (E)-(2)

3. Which one of the following syntax will help you declare the `@ Page` directive in an ASP.NET Web application?

(A)	<@ Page <attribute>=<value>>
(B)	<%@@ Page <attribute>=<value>%>
(C)	<%@ Page <attribute>=<value>%>

(D) <%@ Page <attribute>=<value> %>

(A)	A	(C)	B
(B)	C	(D)	D

3.3 Basic Event Handling in Web Pages

In this last lesson, **Basic Event Handling in Web Pages**, you will learn to:

- Explain how event handling takes place in Web pages.
- Describe the Page _ Load event.
- Identify the importance of IsPostBack property of Page object.

3.3.1 Event Handling

Events are actions that are fired while your application is running. For example, button clicks or key presses are some of the common events. Each of these events is handled by a block of code called **event handler**, which is executed when a particular event occurs.

ASP.NET is an event-driven programming model because everything that happens in a Web page is in response to some event. ASP.NET includes the event handlers in the server-side programming code and provides techniques to handle events manually and automatically.

You can manually associate the events with their respective event handlers using the += operator. These event handlers are specified as a delegate when using the += operator. Figure 3.3 displays a real-life example of event handling.

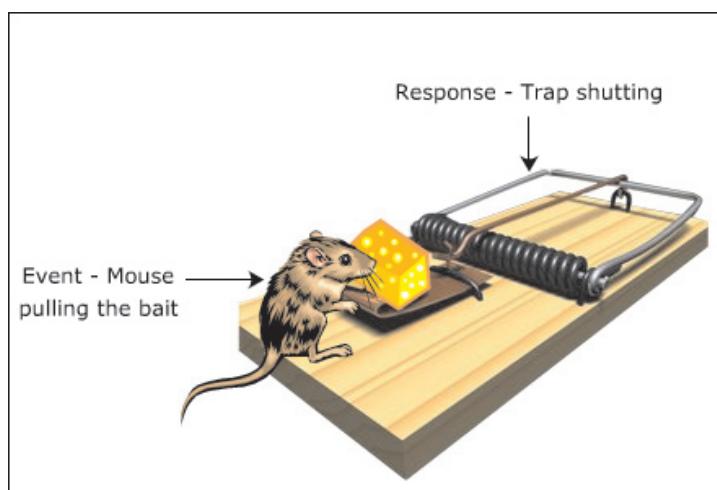


Figure 3.3: Event Handling

3.3.2 Automatic Event Handling

An ASP.NET Web page goes through a life cycle in which each stage can raise an event. These events are automatically processed by the appropriate event handlers to which they are associated. The life cycle of an ASP.NET Web page includes initialization, executing event-handling codes, rendering, and unloading processes. The different events in the Web page cycle are as follows:

→ **Init**

The `Init` event is the first event in the life cycle of the ASP.NET Web page. The `Init` event initializes all the controls of the Web page.

This event is also used when the variables need to be declared and initialized before the Web page is processed.

→ **Load**

The `Load` event is fired immediately after the `Init` event is fired. This event executes the event handling codes and is triggered every time a Web page is loaded.

→ **PreRender**

The `PreRender` event is triggered before the page is submitted to the user.

→ **UnLoad**

The `UnLoad` event is triggered once the page is submitted to the user.

Code Snippet:

→ **Init**

The following code demonstrates how to use the `Init` event.

```
protected void Page_Init(object sender, EventArgs e)
{
    Response.Write("Page initialization event occurs.");
}
```

The code handles the `Init` event that is fired when the page is initialized.

→ **Load**

The following code demonstrates how to use the `Load` event.

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("Page load event occurs.");
}
```

The code handles the `Load` event that is fired when the page is loaded for the first time.

→ PreRender

The following code demonstrates how to use the `PreRender` event.

```
protected void Page_PreRender(object sender, EventArgs e)
{
    Response.Write("Page prerender event occurs.");
}
```

The code handles the `PreRender` event that is fired before the page is submitted to the user.

→ UnLoad

The following code demonstrates how to use the `UnLoad` event.

```
protected void Page_Unload(object sender, EventArgs e)
{
    //This is the UnLoad event
}
```

The code handles the `UnLoad` event that when the page is submitted to the user.

3.3.3 Page_Load Event

The `Page_Load` event is triggered every time a Web page is requested. When a user requests for a Web page, the variables and the controls within the page are initialized and then, the page is loaded. For example, if you are creating a Web page and want certain part of the code to be executed every time a page is requested, you can include that code in the `Page_Load` event.

The `Page_Load` event allows you to define specific behavior on every page for an application. This means, you can customize each page of the application in different ways. It can also include code for database connectivity or binding data to the grid and so forth.

The following code demonstrates how to use the `Page_Load` event.

Code Snippet:

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("Welcome...");
```

The code demonstrates an event handler for the `Load` event which is fired every time a Web page is requested. The code will display a Web page with a Welcome message whenever the event is fired.

3.3.4 Postback

Postback is the information submitted by the browser to the server for processing. When a user submits information by clicking a button or by selecting an item from a grid, the browser passes the information to the server.

When a user interacts with the Web page by generating an event, the page is posted back to the server. The server processes the code present in the event handlers and then, the page is re-created. This page goes through the same cycle by firing the `Init()` and `Load()` events. To avoid firing these events, ASP.NET provides the `IsPostBack` property.

3.3.5 IsPostBack Property

The `IsPostBack` property checks whether the Web page is requested for the first time or is a result of a postback. If the page is requested for the first time, the `IsPostBack` property of the page is `false`. If the page is submitted back to the server, the `IsPostBack` property of the page is `true`.

This property allows you to assign different properties for different Web pages and allows you to control the behavior of the Web page.

Code Snippet:

The following code demonstrates how to use the `IsPostBack` property.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Page.IsPostBack)
    {
        Response.Write("Welcome...");
```

```
{
    txtName.Text = "Hello";
}
```

The code demonstrates the use of `IsPostBack` property in the `Load` event. The `IsPostBack` property determines whether the Web page is requested for the first time or is a result of a postback. If the user clicks the button on the page, it is a postback and the property returns true, therefore, the Web page displays the Welcome message.

Knowledge Check 3

1. Which of the following statements about event-handling and postbacks in ASP.NET are true?

(A)	The <code>PreRender</code> event is fired once the Web page is submitted to the user.		
(B)	The <code>Page_Load</code> event is fired for every request of the Web page.		
(C)	Postback is information submitted by the browser to the server.		
(D)	The <code>IsPostBack</code> property of a Web page is true if the page is requested for the first time.		
(E)	The <code>Page_Init</code> event is the first event that is fired in the life-cycle of the ASP.NET Web page.		

(A)	A, B	(C)	B, C
(B)	C, D	(D)	A, D

2. Describe the `Page_Load` event.

(A)	protected void Page_Load () { //Page load event }
(B)	protected Page_Load(object sender, EventArgs e) { //Page load event }
(C)	protected void PageLoad(object sender, EventArgs e) { //Page load event }

(D)	<pre>protected void Page_Load(object sender, EventArgs e) { //Page load event }</pre>		
-----	---	--	--

(A)	A	(C)	B
(B)	C	(D)	D

Module Summary

In this module, **Basics of ASP.NET**, you learnt about:

→ **ASP.NET Web Page Code Model**

ASP.NET Web page code model can be either a single-file page model or a code-behind model. The single-file page model allows you to include both the markup and the programming code in the same .aspx file. The code-behind page model separates the markup code and the programming code into different files. The markup code is saved in the .aspx file and the programming code is saved in the .aspx.cs file.

→ **Directive Syntax**

Directives are commands given to the .NET framework to decide the way in which an ASP.NET Web page is compiled. @ Page directive specifies different attributes for an ASP.NET Web page. @ Import directive imports the functionality of the classes and interfaces of a specified namespace into the current application.

→ **Basic Event Handling in Web pages**

An ASP.NET Web page undergoes a lot of processes like initialization, execution of error-handling code if any, rendering, and unloading. The Page_Load event is triggered every time a Web page is requested. The IsPostBack property checks whether the requested page is loaded for the first time or is a result of the feedback.

Module - 4

Request, Response, and Server Objects

Welcome to the module **Request, Response, and Server Objects**. The Request and Response objects in ASP.NET applications represent the information coming into and going out of the Web server respectively. The Server objects describe the utility methods which are used to move control among various Web pages.

In this module, you will learn about:

- Request Object
- Response Object
- Server Object

4.1 Request Object

In this first lesson, **Request Object**, you will learn to:

- ➔ Describe Request object and HttpRequest class.
- ➔ List and explain the methods and properties of HttpRequest class.

4.1.1 Request, Response, and Server Objects

ASP.NET maintains information regarding the current application such as the HTTP request and the user session whenever a Web application runs. This information is summarized and stored using in-built classes of ASP.NET. ASP.NET provides in-built objects for these classes and these objects can be accessed through code. Some of the important objects are as follows:

- ➔ Request Object
- ➔ Response Object
- ➔ Server Object

Figure 4.1 demonstrates this concept.

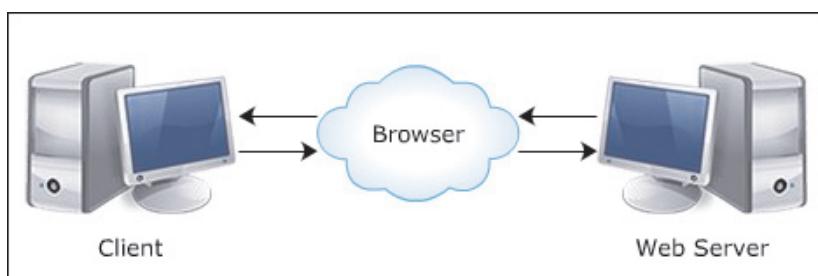


Figure 4.1: Request Object

4.1.2 Request Object and HttpRequest Class

A request is the interaction between the browser and the server in which the browser asks the server for a Web page. The URL sent by the browser identifies the requested page. Along with this, the URL also contains the information that identifies the source of the request and the path to the requested page. This information is stored as a set of properties of the `Request` object.

The `Request` object is the intrinsic object of `HttpRequest` class. The properties of the `Request` object can be accessed to retrieve information such as the type of browser used and the Internet Protocol (IP) address of the URL request.

Figure 4.2 demonstrates an example of a request operation.

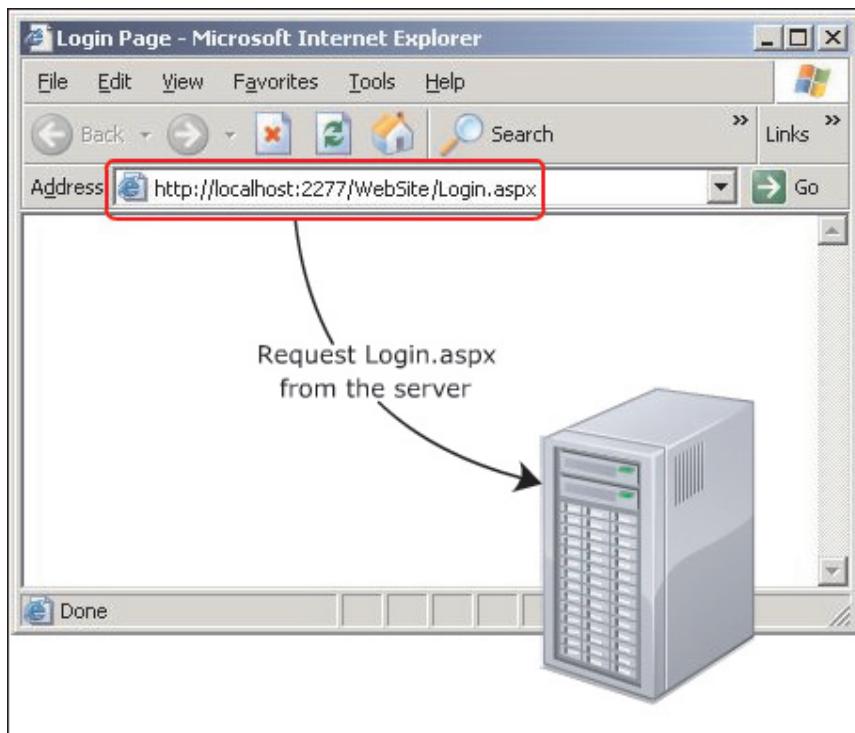


Figure 4.2: Example of a Request Operation

4.1.3 Properties of `HttpRequest` Class

The `HttpRequest` class is present in the `System.Web` namespace. The properties of this class are used to read the HTTP values sent by a client during a Web page request. Some of the commonly used properties of the `HttpRequest` class are as follows:

- ➔ ApplicationPath
- ➔ Browser
- ➔ ContentLength
- ➔ FilePath
- ➔ QueryString
- ➔ RequestType

Figure 4.3 shows some of the properties of the `HttpRequest` class.

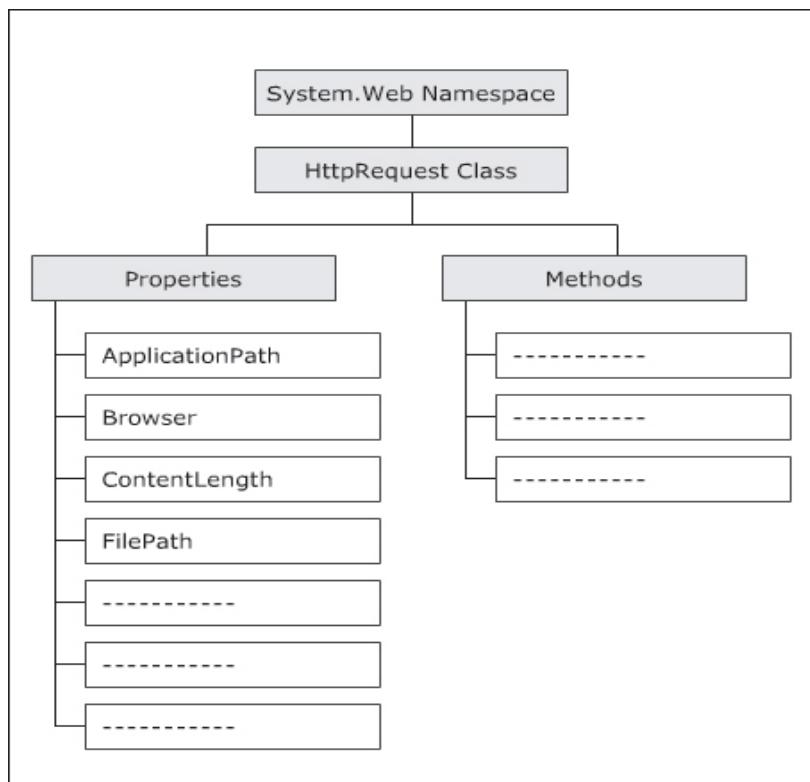


Figure 4.3: Properties of `HttpRequest` Class

These are described as follows:

→ **ApplicationPath**

The `ApplicationPath` property retrieves the ASP.NET application's virtual root path on the server.

→ **Browser**

The `Browser` property retrieves the attributes of the client's browser such as its name, version, and platform under which it is running.

→ **ContentLength**

The `ContentLength` property specifies the length of the entire content of the client request. This length is specified in bytes.

→ **FilePath**

The `FilePath` property retrieves the virtual path of the page requested by the browser. For example: /ASPNET2.0/HttpRequest/properties.aspx.

→ **QueryString**

The `QueryString` property returns a collection of name-value pairs that represent the elements of a form.

→ **RequestType**

The `RequestType` property specifies or retrieves the HTTP data transfer method, GET or POST, used by the client.

Syntax:

→ **ApplicationPath**

The following syntax is used to retrieve the virtual path of the application.

```
public string ApplicationPath { get; }
```

where,

`public`: Specifies that the property is accessible across the entire application.

`string`: Specifies that the return type of this property is `string`.

`get`: Is the accessibility type to retrieve `string` information.

→ **Browser**

The following syntax is used to retrieve the capabilities of the browser.

```
public HttpBrowserCapabilities Browser { get; set; }
```

where,

`HttpBrowserCapabilities`: Specifies that an object of this type is returned by the property and it allows the server to gather information about the capabilities of the browser.

`get`: Is the accessibility type to retrieve the browser capability.

`set`: Is the accessibility type to specify the browser capability.

→ **ContentLength**

The following syntax is used to retrieve the length of the client request in bytes.

```
public int ContentLength { get; }
```

where,

`int`: Specifies that the return type of the property is `integer`.

`get`: Is the accessibility type to retrieve the length of the content sent by the client.

→ **FilePath**

The following syntax is used to retrieve the virtual path of the current request.

```
public string FilePath { get; }
```

where,

`get:` Is the accessibility type to retrieve the virtual path of the current request.

→ **QueryString**

The following syntax is used to return the collection of HTTP query string variables sent by the client.

```
public NameValueCollection QueryString { get; }
```

where,

`NameValueCollection:` Is the return type of the property and represents the collection of query string variables.

→ **RequestType**

The following syntax is used to retrieve or specify HTTP data transfer method used by the client.

```
public string RequestType { get; set; }
```

Code Snippet:

→ **ApplicationPath**

The following code demonstrates how the `ApplicationPath` property is used to retrieve the virtual path of an application.

```
Response.Write("Application Path is : " + Request.ApplicationPath);
```

In this code, `Request.ApplicationPath` retrieves the virtual root path of the application from the server. `Response.Write` method writes the output to the Web page.

→ **Browser**

The following code demonstrates how the `Browser` property is used to retrieve information about the browser.

```
HttpBrowserCapabilities browserInfo = Request.Browser;
Response.Write("Name = " + browserInfo.Browser);
Response.Write("Version = " + browserInfo.Version);
```

In this code, `browserInfo` is an instance of the `HttpBrowserCapabilites` class. Information about the client's browser is stored in `browserInfo`.

The `Browser` and `Version` properties of `HttpBrowserCapabilities` class are referenced to display the name and version of the client's browser.

→ ContentLength

The following code demonstrates how to retrieve the length of a client request in bytes.

```
int length;
length = Request.ContentLength;
Response.Write("Content Length : " + length);
```

In this code, `length` is declared as an integer variable to store the length of the content sent by the client. The `ContentLength` property retrieves the length of the client request and assigns it to the `length` variable. The `Response.Write` method is used to display the length of the content.

→ FilePath

The following code demonstrates how to retrieve the virtual path of the current request.

```
Response.Write("FilePath is :" + Request.FilePath);
```

In this code, `Request.FilePath` retrieves the virtual path of the current request

→ QueryString

The following code demonstrates how to retrieve the value of a single form input element using `QueryString` property.

```
Request.QueryString["varName"]
```

In this code, the value of a single form input element is obtained using `QueryString` property.

The following code demonstrates how to retrieve the name and value pair of each form input element by traversing through the collection returned by `QueryString` property.

```
foreach ( string varNames in Request.QueryString )
{
    Response.Write ( Request.QueryString[varNames] + <br /> );
}
```

In this code, the values of all the input elements of the form are obtained using `QueryString` property. The `foreach` loop traverses through the collection and the values of input elements are displayed.

→ **RequestType**

The following code demonstrates how to retrieve or specify HTTP data transfer method used by client.

```
Response.Write ("Type of Request is : " + Request.RequestType);
```

In this code, `Request.RequestType` property is used to retrieve the HTTP data transfer method.

4.1.4 Methods of `HttpRequest` Class

`HttpRequest` class contains methods that are used to get path information and save requests on the disk. Some of the commonly used methods of `HttpRequest` class are as follows:

- `SaveAs`
- `MapImageCoordinates`
- `MapPath`

Figure 4.4 shows some of the methods of `HttpRequest` class.

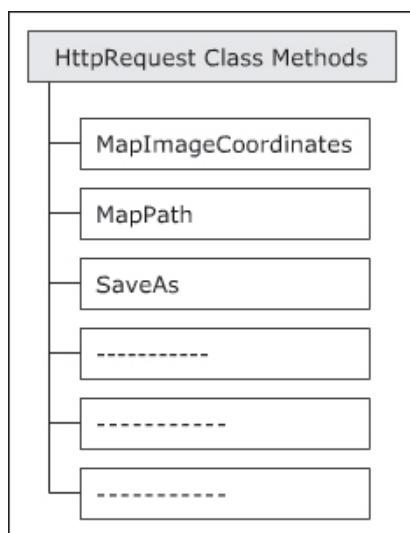


Figure 4.4:Methods of `HttpRequest` class

These are described as follows:

→ **SaveAs**

The `SaveAs()` method saves the entire request on the disk.

→ **MapImageCoordinates**

The `MapImageCoordinates` method maps the incoming image-field form parameter to its appropriate x and y coordinates.

→ **.MapPath**

The `.MapPath` method maps the virtual path in the requested URL to a physical path on the server for the current request.

Syntax:

→ **SaveAs**

The following syntax is used to save the HTTP request to the disk.

```
public void SaveAs (string filename, bool includeHeaders)
```

where,

`filename`: Is the first argument, of type `string`, specifying the disk location.

`includeHeader`: Is the second argument, of type `Boolean`. If the boolean value is true, then, the header is also saved to the disk.

→ **MapImageCoordinates**

The following syntax is used to map incoming image-field form parameter to its appropriate x and y coordinate values.

```
public int[] MapImageCoordinates (string imageFieldName)
```

where,

`imageFieldName`: Is the string reference to form image map.

`int[]`: Specifies that the return type of the method is a two-dimensional array of type integer.

→ **.MapPath**

The following syntax is used to map the virtual path in the requested URL to a physical path on the server for the current request.

```
public string MapPath (string virtualPath)
```

where,

`virtualPath`: Is the argument that specifies the virtual path.

→ SaveAs

The following code demonstrates how an HTTP request is saved onto the disk.

```
Request.SaveAs("c:\\HttpRequest.txt", true);
```

In this code, `c:\\HttpRequest.txt` is the disk location. Since the second argument is `true`, the header is also saved to the disk.

→ MapImageCoordinates

The following code maps the coordinates on the image and displays them on screen, when the image is clicked.

```
int[] coordinates = Request.MapPath("imgLogo");
Response.Write(coordinates[0].ToString() + ", " + coordinates[1].ToString());
```

In this code, the coordinates of the point where the mouse is clicked on the image are mapped and displayed.

→ MapPath

The code displays how to map the virtual path in the requested URL to a physical path on the server.

```
string fullPath = Request.MapPath("TestFile.txt");
Response.Write("Full path is :" + fullPath);
```

In this code, the `Request.MapPath` method maps the virtual path of the `TestFile.txt` file to its physical path.

Knowledge Check 1

- Which of the following statements about `Request` object and `HttpRequest` class are false?

(A)	The <code>FilePath</code> property retrieves the virtual path of the page requested by the browser.
(B)	The <code>Request</code> object is the in-built object of <code>HttpRequest</code> method.
(C)	The <code>RequestType</code> property specifies or retrieves the HTTP data transfer method, GET or POST, used by the client.
(D)	The <code>HttpRequest</code> class is present in the <code>System.Web</code> namespace.
(E)	The <code>MapPath</code> method maps the incoming image-field form parameter to its appropriate coordinates.

(A)	A, B	(C)	B, E
(B)	C, D	(D)	A, D

2. Can you match the properties and methods of `HttpRequest` class with their corresponding description?

Descriptions		Component	
(A)	Returns a collection of name-value pairs that represent the elements of a form.	(1)	ApplicationPath
(B)	Maps the incoming image-field form parameter to its coordinates.	(2)	FilePath
(C)	Retrieves the virtual root path on the server.	(3)	QueryString
(D)	Maps the virtual path in the requested URL.	(4)	MapImageCoordinates
(E)	Retrieves the virtual path of the page.	(5)	MapPath

(A)	(A)-(5), (B)-(3), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(4), (C)-(1), (D)-(5), (E)-(2)	(D)	(A)-(4), (B)-(3), (C)-(5), (D)-(1), (E)-(2)

4.2 Response

In this second lesson, **Response Object**, you will learn to:

- Explain `Response object` and `HttpResponse class`.
- List and explain the methods and properties of `HttpResponse class`.

4.2.1 Response Object and `HttpResponse Class`

A response is the interaction between the browser and the server in which the server responds to the HTTP request sent by the browser. `Response object` is the intrinsic object of `HttpResponse class` and stores information related to the server's response.

`HttpResponse class` is used to control the output of the HTTP request. It controls the various character sets used to encode the response. It also specifies whether the output will be buffered and sent all at once or it will be sent in parts as soon as a particular part is processed.

Figure 4.5 demonstrates this concept.

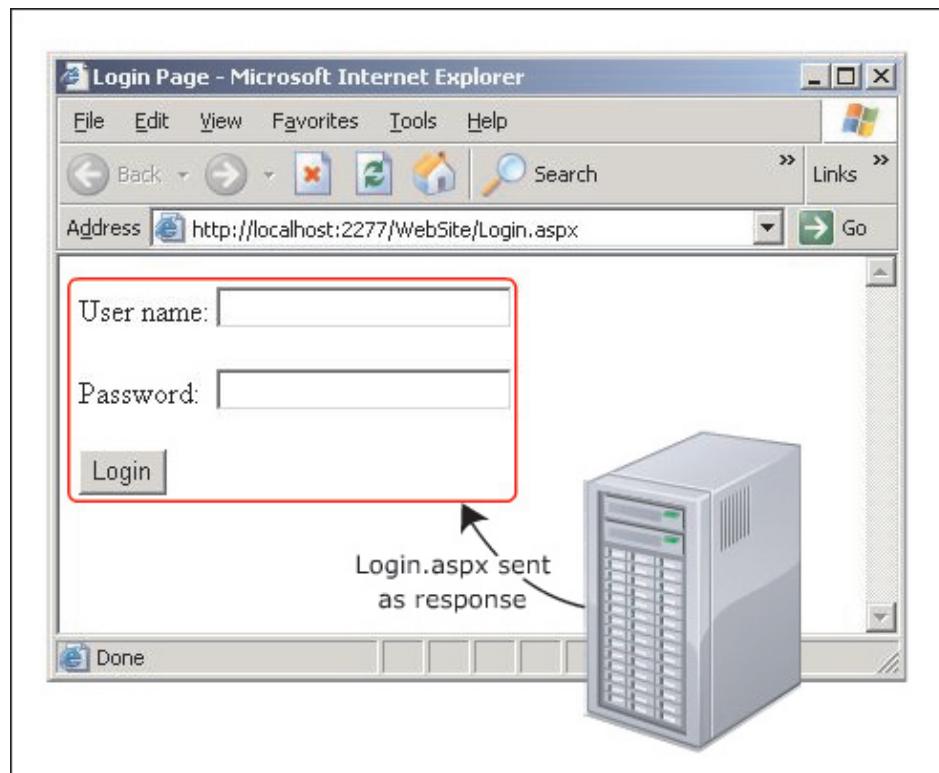


Figure 4.5: Example of a Response Mechanism

4.2.2 Properties of `HttpResponse` Class

`HttpResponse` class is present in the `System.Web` namespace. The properties of this class are used to specify or retrieve the HTTP status, check whether the client is connected to the server, and information about cookies to be sent.

Some of the commonly used properties of `HttpResponse` class are as follows:

- ➔ `BufferOutput`
- ➔ `Charset`
- ➔ `ContentEncoding`
- ➔ `ContentType`
- ➔ `Cookies`
- ➔ `IsClientConnected`

Figure 4.6 depicts some properties of the `HttpResponse` class.

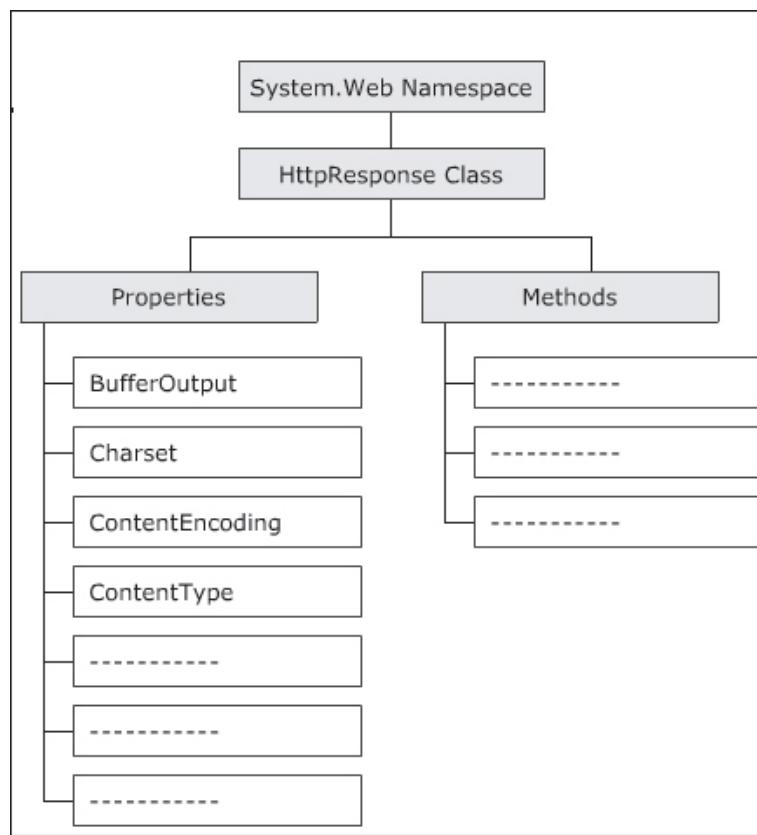


Figure 4.6: Properties of `HttpResponse` Class

These are described as follows:

→ **BufferOutput**

The `BufferOutput` property enables developers to specify whether the output of the HTTP request should be buffered and sent as a whole. Alternatively, the output can be sent in parts as it keeps getting processed.

→ **Charset**

The `Charset` property specifies or retrieves the character set of the response.

→ **ContentEncoding**

The `ContentEncoding` property specifies or retrieves the HTTP character set of the response.

→ **ContentType**

The `ContentType` property specifies or retrieves the HTTP Multipurpose Internet Mail Extension (MIME) type of response.

→ **Cookies**

The `Cookies` property contains a collection of cookies to be transmitted to the client in the response.

→ **IsClientConnected**

The `IsClientConnected` property retrieves a value indicating whether the client is still connected to the Web server during processing a response.

Syntax:

→ **BufferOutput**

The following syntax is used to enable a developer to specify whether the response should be held in memory until processing is finished and then, sent to the client or whether it should be sent as soon as it is processed.

```
[public bool BufferOutput { get; set; }]
```

where,

`bool`: Is the Boolean value, which is either true or false.

→ **Charset**

The following syntax is used to retrieve or specify the HTTP character set of the response.

```
[public string Charset { get; set; }]
```

→ **ContentEncoding**

The following syntax is used to specify or retrieve the HTML character set of the current response.

```
[public Encoding ContentEncoding { get; set; }]
```

→ **ContentType**

The following syntax is used to specify or retrieve the HTTP MIME (Multipurpose Internet Mail Extension) type of the current response.

```
[public string ContentType { get; set; }]
```

→ **Cookies**

The following syntax is used to retrieve a collection of all cookies that is to be transmitted to the client in the current response.

```
[public HttpCookieCollection Cookies { get; }]
```

where,

`HttpCookieCollection`: Is a way to manipulate HTTP cookies.

→ **IsClientConnected**

The following syntax is used to retrieve the boolean value indicating whether the client is still connected to the server.

```
public bool IsClientConnected { get; }
```

where,

`bool`: Is a boolean value. It returns true if the client is currently connected, else it returns false.

Code Snippet:

→ **BufferOutput**

The following code demonstrates how to specify a value indicating whether to buffer output.

```
if (Response.BufferOutput == true)
```

In the code, the property value is set to true. If this condition is satisfied, which means if the output to the client is buffered, then the block following the if statement is executed.

→ **Charset**

The following code specifies the HTTP character set of the response.

```
if (Response.Charset == "iso-1234-2")
```

In the code, the block following the if statement is executed only if the character set of response is set to a standard specified by ISO.

→ **ContentEncoding**

The following code retrieves the description of the character set encoding.

```
Response.Write(Response.ContentEncoding.EncodingName);
```

The code writes a readable description of the character set encoding to the output stream.

→ **ContentType**

The following code specifies the content type of the output.

```
Response.ContentType = "text/html";
```

In the code, the content type of the output is set as text/html.

→ **Cookies**

The following code creates a new cookie, sets the value of the cookie, and adds the cookie to the current cookie collection.

```
HttpCookie MyCookie = new HttpCookie("Name");
MyCookie.Value = "John Smith";
Response.Cookies.Add(MyCookie);
```

In the code, `HttpCookie MyCookie = new HttpCookie ("Name")` creates a new cookie named 'Name'. `MyCookie.Value` sets the value of the cookie as 'John Smith'. `Response.Cookies.Add (MyCookie)` adds the cookie to the current cookie collection.

→ **IsClientConnected**

The following code retrieves a boolean value that specifies whether or not the client is connected to the server.

```
if (Response.IsClientConnected == true)
{Response.Write("Client is still connected to the server");}
```

In the code, if the property value is true, which means that the client is currently connected, the `Response.Write` method is executed.

4.2.3 Methods of `HttpResponse` Class

Methods of the `HttpResponse` class are used to modify the content of the HTTP response including headers, cookies, as well as the actual text and graphical content. A response buffer is maintained to store the response before sending. Commonly used methods of `HttpResponse` class are as follows:

- Clear
- ClearContent
- Close
- End
- Redirect
- Write

Figure 4.7 depicts some of the methods of `HttpResponse` class.

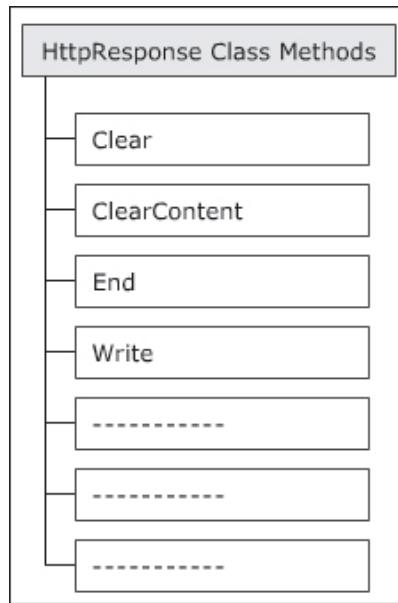


Figure 4.7: Methods of `HttpResponse` Class

These are described as follows:

→ **Clear**

The `Clear` method removes all content from the current buffer stream.

→ **ClearContent**

The `ClearContent` method removes all content from the buffer stream.

→ **Close**

The `Close` method closes the connection to the client.

→ **End**

The `End` method sends all buffered output to the client and terminates the execution of the current response.

→ **Redirect**

The `Redirect` method sends the client to a new URL.

→ **Write**

The `Write` method writes information to the HTTP response.

Syntax:**→ Clear**

The following syntax is used to clear the current response buffer.

```
public void Clear ()
```

→ ClearContent

The following syntax removes all the content from the response buffer.

```
public void ClearContent ()
```

→ Close

The following syntax is used to close the connection to the client.

```
public void Close ()
```

→ End

The following syntax is used to clear the response buffer and terminate the connection.

```
public void End ()
```

→ Redirect

The following syntax is used to send the client to a new URL.

```
public void Redirect (string url)
```

→ Write

The following syntax is used to write information to the output stream.

```
public void Write (char ch)  
public void Write (Object obj)  
public void Write (string s)
```

where,

char: Writes a character to an HTTP output stream.

Object: Writes an object to an HTTP output stream.

String: Writes a string to an HTTP output stream.

Code Snippet:**→ Clear**

The following code clears the current buffer stream.

```
Response.Clear();
```

→ ClearContent

The following code clears the buffer stream of the entire content.

```
Response.ClearContent();
```

→ Close

The following code closes the current connection.

```
Response.Close();
```

→ End

The following code clears the buffer and terminates the connection.

```
Response.End();
```

→ Redirect

The following code redirects the client to a new URL.

```
Response.Redirect("http://www.example.com");
```

In this code, the client is redirected to <http://www.example.com>.

→ Write

The following code writes the information to the output stream.

```
Response.Write("<h1>Welcome to ASP.Net</h1>");
```

In this code, `Response.Write` displays the heading "Welcome to ASP.Net" on the Web page.

Knowledge Check 2

1. Which of the following statements about Response object and HttpResponse class are false?

(A)	The End property sends all buffered output to the client and terminates the execution of the current response.
(B)	The ClearContent method removes all content from the buffer stream.
(C)	HttpResponse class is used to control the output of the HTTP request.
(D)	A response is the interaction between the browser and the server in which the client responds to the MIME request sent by the server.
(E)	The Cookies property contains a collection of cookies to be transmitted to the client in the request.

(A)	A, B	(C)	B, E
(B)	C, D	(D)	A, D, E

2. Which one of the following codes will help you switch from the current Web page to the Web page named "Welcome.html"?

(A)	<pre>Response.ClearContent(); if (Response.IsClientConnected) { Response.Redirect("Welcome.html"); } else { Response.Write("The browser is still not connected."); Response.End();</pre>
(B)	<pre>Response.ClearContent(); if (Response.IsClientConnected) { Response.redirect("Welcome.html"); } else { Response.Write("The browser is still not connected."); Response.End(); }</pre>

(C)	<pre>Response.ClearContent(); if (Response.IsClientConnected=true) { Response.Redirect("Welcome.html"); } else { Response.Write("The browser is still not connected."); Response.End(); }</pre>
(D)	<pre>Response.ClearContent(); if (Response.IsClientConnected==true) { Response.Redirect(Example.aspx); } else { Response.Write("The browser is still not connected."); Response.End(); }</pre>

(A)	A	(C)	B
(B)	C	(D)	D

4.3 Server Object

In this third lesson, **Server Object**, you will learn to:

- ➔ Describe `Server` object and `HttpServerUtility` class.
- ➔ List and explain the methods and properties of `HttpServerUtility` class.

4.3.1 Server Object and `HttpServerUtility` Class

`Server` object exposes various utility methods that can be used to transfer control between pages, decode HTML text, get error information, and so on. The `Server` object is the intrinsic object of the `HttpServerUtility` class.

`HttpServerUtility` class belongs to `System.Web` namespace. This class provides methods for tasks such as processing Web requests, performing utility functions for encoding and decoding strings for use in URLs, providing access to limited error information, and modifying the execution of the current request. The methods and properties of `HttpServerUtility` class can be accessed through the built-in `Server` object.

4.3.2 Properties of `HttpServerUtility` Class

`HttpServerUtility` class is present in the `System.Web` namespace. The properties of this class are used to get the server name and set the page-out time.

These properties are as follows:

- `MachineName`
- `ScriptTimeout`

Figure 4.8 shows some of the properties of the `HttpServerUtility` class.

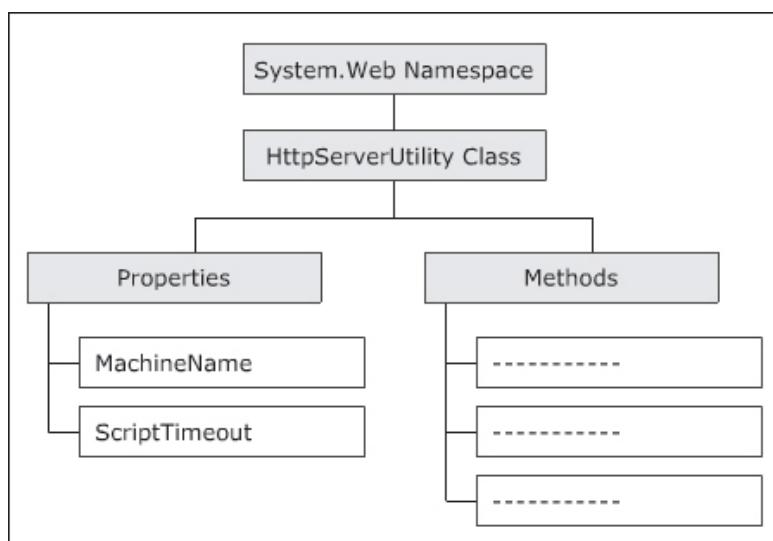


Figure 4.8: Properties of `HttpServerUtility` class

These are described as follows:

→ **MachineName**

The `MachineName` property retrieves the system name of the server that holds the Web application.

→ **ScriptTimeout**

The `ScriptTimeout` property specifies or retrieves the time (in seconds) in which the request is timed out.

Syntax:**→ MachineName**

The following syntax is used to retrieve the name of the server.

```
public string MachineName { get; }
```

→ ScriptTimeout

The following syntax is used to retrieve or specify the duration of time in which the request will be timed out.

```
public int ScriptTimeout { get; set; }
```

Code Snippet:**→ MachineName**

The code is used to retrieve the computer-name of the server.

```
String machineName;
machineName = Server.MachineName;
Response.Write("The server machine name is:" + machineName);
```

In this code, `machineName` is declared as a string variable. The system name of the server is retrieved using the `MachineName` property and is assigned to the variable `machineName`. The `Response.Write()` method then, displays this system name.

→ ScriptTimeout

The code is used to set the script time-out.

```
Server.ScriptTimeout = 60;
```

In this code, the script time-out is set to 60 seconds.

4.3.3 Methods of `HttpServerUtility` Class

`HttpServerUtility` class contains methods to encode strings, retrieve the file path, and execute requests. Commonly used methods of `HttpServerUtility` class are as follows:

- Execute
- HtmlEncode
- MapPath

→ **UrlEncode**

Figure 4.9 shows some of the methods of the `HttpServerUtility` class.

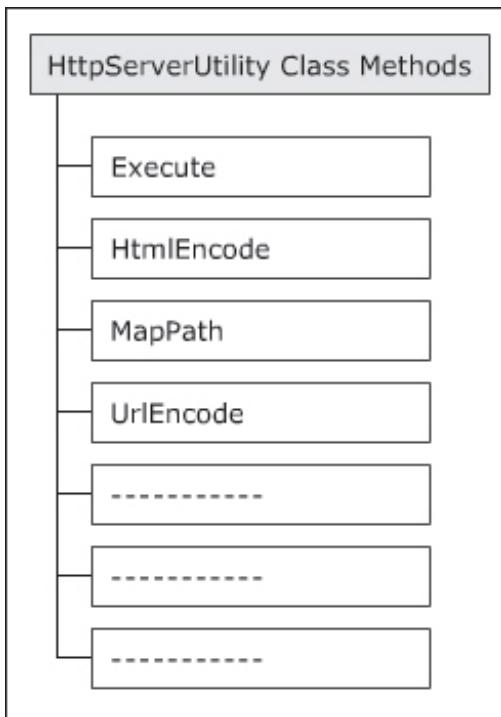


Figure 4.9: Methods of `HttpServerUtility` class

These are described as follows:

→ **Execute**

The `Execute` method processes the specified Web page.

→ **HtmlEncode**

The `HtmlEncode` method encodes a string to be displayed in a browser.

→ **.MapPath**

The `.MapPath` method returns the physical file path. This physical path corresponds to the specified virtual path on the Web server.

→ **UrlEncode**

The `UrlEncode` method encodes a string for reliable HTTP transmission from the Web server to a client via the URL.

Syntax:**→ Execute**

The following syntax is used to process another page using the specified URL path.

```
public void Execute(string path)
```

where,

path: Specifies the URL path of the new request

→ HtmlEncode

The following syntax is used to encode a string and return the encoded string.

```
public string HtmlEncode(string s)
```

The following syntax is used to encode a string and send the output to a `TextWriter` output stream.

```
public void HtmlEncode(string s, TextWriter output)
```

→ MapPath

The following syntax is used to retrieve the physical file path that corresponds to the virtual path of the Web server.

```
public string MapPath(string path)
```

where,

path: is the virtual path of Web server.

→ UrlEncode

The following syntax is used to encode a string for reliable HTTP transmission and return the encoded string.

```
public string UrlEncode(string s)
public void UrlEncode(string s, TextWriter output)
```

where,

s: Is the text to encode.

Code Snippet:**→ Execute**

The code explains how the `Execute` property works.

```
Server.Execute("http://www.example.com/loginexample.aspx");
```

In this code, `Server.Execute` executes the `.aspx` page, `loginexample.aspx`, on the server.

→ **HtmlEncode**

The code explains how HTML encodes a string for transmission.

```
String msg = "This is a <Message>.";  
String encodedString = Server.HtmlEncode(msg);
```

This code encodes the string named `msg` that contains the text “This is a `<Message>`.” and copies it into the string named `encodedString`.

→ **MapPath**

The code returns the physical path of the virtual directory that contains the specified Web site.

```
String filePath;  
filePath = Server.MapPath("/MyWebSite");
```

→ **UrlEncode**

The code encodes the URL before sending it to the browser.

```
String myURL;  
myURL = "http://www.example.com";  
Response.Write("<a href=" + Server.UrlEncode(myURL) + ">Example</a>");
```

In this code, the string `myURL` is encoded before sending it to the browser.

Knowledge Check 3

- Which one of the following codes will help you to encode the URL before sending it to the client browser?

(A)	<pre>String myURL; myURL = "http://www.example.com"; Response.Write ("Example");</pre>
(B)	<pre>String myURL; myURL = "http://www.example.com" Response.Write ("Example");</pre>

(C)	String myURL; myURL = "http://www.example.com"; Response.Write ("Example");
(D)	String myURL; myURL = "http://www.example.com"; Response.Write ("Example");

(A)	B	(C)	C
(B)	D	(D)	A

2. Which of the following statements about Server object and `HttpServerUtility` class are false?

(A)	The Server object serves as an in-built property of the <code>HttpServerUtility</code> class.
(B)	The <code>ScriptTimeout</code> property specifies or retrieves the time in milliseconds in which the request is timed out.
(C)	The <code>HtmlEncode</code> method encodes a string to be displayed in a browser.
(D)	Server object exposes utility methods that can be used to transfer control between pages and decode HTML text.
(E)	The <code>UrlEncode</code> method encodes a string for reliable HTTP transmission from the Web server to a client via the URL.

(A)	A, B	(C)	B, E
(B)	C, D	(D)	A, D

Module Summary

In this module, **Request, Response, and Server Objects**, you learnt about:

→ Request Object

`Request` object is the input object in an ASP.NET Web application and represents the information going to the Web server from the browser. `Request` object is the intrinsic object of `HttpRequest` class and is used to expose the methods and properties of this class.

→ Response Object

`Response` object is the output object in an ASP.NET Web application and represents the information going from the server to the browser. `Response` object is the intrinsic object of `HttpResponse` class and is used to expose the methods and properties of this class.

→ Server Object

`Server` object is the built-in object of the `HttpServerUtility` class and is used to expose the various utility methods. This method can be used to perform tasks such as transfer control between pages, get information about the most recent error, and encode and decode HTML text.

Module - 5

Basic Web Server Controls

Welcome to the module, **Basic Web Server Controls**. Web server controls are generally used to create the graphical user interface for Web applications. These controls are used to provide better appearance and functionality to Web Forms.

In this module, you will learn about:

- ➔ Introduction to Web Server Controls
- ➔ Basic Web Server Controls

Web Development

http://www



5.1 Introduction to Web Server Controls

In this first lesson, **Introduction to Web Server Controls**, you will learn to:

- Describe **Web Server Controls**.
- Describe briefly the `System.Web.UI.WebControls` namespace and list the commonly used classes.

5.1.1 Web Server Controls

Controls are used to build a Graphical User Interface (GUI) for applications. The Visual Studio 2005 IDE has four categories of controls for Web applications, namely HTML server controls, Web server controls, Validation controls, and User Controls.

Web server controls are controls that allow user interaction by performing an action, displaying information, or setting values on Web Forms. These controls are used to design the layout of the Web Form. The end user can perform various actions on these controls such as clicking them or entering text into them. These actions cause events to occur.

Web Server controls provide more built-in features than HTML server controls. In addition, they also include special-purpose controls such as Calendar, Menu, and TreeView controls.

Figure 5.1 demonstrates this concept.

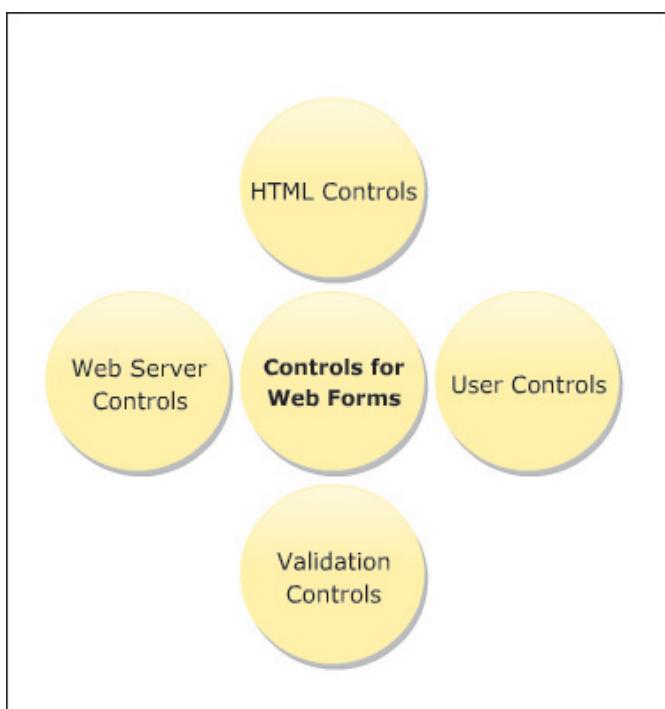


Figure 5.1: Controls for Web Forms

5.1.2 Features of Web Server Controls

Web server controls provide complex functionality with minimum code. This allows you to more easily develop Web applications. Some important features of Web server controls include:

- **Browser detection:** The Web Server controls can automatically detect the browser capabilities such as its type and version number and accordingly provide the markup.
- **Template usage:** Some Web server controls allow you to define a custom layout for the control using templates.
- **Theme support:** Web server controls support themes. This enables you to define a standard appearance for controls throughout the Web site.

5.1.3 System.Web.UI.WebControls Namespace

The `System.Web.UI.WebControls` namespace contains classes that allow you to create Web server controls on an ASP.NET Web page. The controls created using these classes run on the server.

Classes of the `System.Web.UI.WebControls` namespace such as `TextBox` and `ListBox` are rendered as HTML tags on the Web page. Certain classes, such as `SqlDataSource` and `ObjectDataSource`, support operations on data; however, these classes are not rendered on the Web page. The `GridView` and `DetailsView` classes are used to record and compare the changes in the data.

Some commonly used classes of the `System.Web.UI.WebControls` namespace are listed as follows:

- Label
- TextBox
- Button
- Image and ImageButton
- LinkButton
- Panel

5.1.4 WebControl Class

The `WebControl` class is included in the `System.Web.UI.WebControls` namespace. It acts as the base class for most Web server controls.

Table 5.1 lists the commonly used properties, methods, and events of the `WebControl` class, which are also inherited by its derived controls.

Name	Description
<code>BackColor</code> property	Specifies or retrieves the background color of the Web server control.
<code>ForeColor</code> property	Specifies or retrieves the foreground color of the Web server control.
<code>Height</code> property	Specifies or retrieves the height of the Web server control.
<code>ID</code> property	Specifies or retrieves the unique identifier assigned to the Web server control.
<code>Visible</code> property	Specifies or retrieves a boolean value indicating whether the Web server control is displayed on the page or not.
<code>Width</code> property	Specifies or retrieves the width of the Web server control.
<code>Focus()</code> method	Sets the focus on the Web server control.
<code>Load</code> event	Occurs when the server control is loaded into the memory.

Table 5.1: Properties of WebControl Class

The following code demonstrates the use of some properties, methods, and events of the `WebControl` class.

Code Snippet:

```
WebControl wctrlTextBox = new TextBox();
wctrlTextBox.ID = "wctrlExample";
wctrlTextBox.BackColor = System.Drawing.Color.Yellow;
wctrlTextBox.ForeColor = System.Drawing.Color.Blue;
wctrlTextBox.Visible = true;
wctrlTextBox.Focus();
frmDetails.Controls.Add(wctrlTextBox);
...
void wctrlExample_Load(object sender, EventArgs e)
{
    Response.Write("WebControl is loaded on the page.");
}
```

In the code, a Web server control named `wctrlTextBox` is created. `wctrlExample` is the value specified to the `ID` property of this control. The background and the foreground colors of the `TextBox` control are set to yellow and blue respectively using the `BackColor` and `ForeColor` properties. The `Visible` property value is set to `true`, which indicates that the `TextBox` control is visible on the Web page. `Focus()` method sets the focus on the Text box control. The control is added to the form `frmDetails` using `Add()` method. The event handler for `Load()` event loads the control on the Web page.

Knowledge Check 1

1. Which of the following statements about Web server controls are true?

(A)	System.Web.UI.Controls namespace contains classes that allow you to create Web server controls on a Web page.		
(B)	Validation controls include special-purpose controls such as a Calendar, Menu, and TreeView controls.		
(C)	Web server controls automatically detect the browser capabilities and provide the markup accordingly.		
(D)	<code>SqlDataSource</code> and <code>ObjectDataSource</code> classes are rendered on the Web page.		
(E)	Web server controls support defining a standard appearance for controls throughout the Web site using themes.		

(A)	A, B	(C)	C, E
(B)	C, D	(D)	A, D

2. Can you match the properties of the `WebControl` class with their corresponding descriptions?

Descriptions		Property	
(A)	Specifies or retrieves the background color of the Web server control.	(1)	<code>ForeColor</code>
(B)	Specifies or retrieves the foreground color of the Web server control.	(2)	<code>Width</code>
(C)	Specifies or retrieves the unique identifier assigned to the Web server control.	(3)	<code>BackColor</code>
(D)	Specifies or retrieves a boolean value indicating whether or not the control is displayed on the page	(4)	<code>ID</code>
(E)	Specifies or retrieves the width of the Web server control.	(5)	<code>Visible</code>

(A)	(A)-(5), (B)-(3), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(1), (C)-(4), (D)-(5), (E)-(2)	(D)	(A)-(4), (B)-(3), (C)-(5), (D)-(1), (E)-(2)

5.2 Basic Web Server Controls

In this second lesson, **Basic Web Server Controls**, you will learn to:

- Describe the `Label` control.
- Describe the `TextBox` control.
- Explain the `Button` control.
- Describe `Image` and `ImageButton` controls.
- Explain the `LinkButton` control.
- Explain the `Panel` control.

5.2.1 Label Control

The `Label` control is a Web server control that allows you to display a label on a Web Form. For example, consider a simple login page where you enter your user name and password. There are two boxes provided for you to enter the data. However, to identify in which box to enter what data, the boxes need to be labeled. Hence, the boxes are given labels such as “User Name” and “Password” to prompt the user to enter the relevant data. Such labels are created using `Label` controls.

The `Label` control is a primary output control. This control is used to display static text. It can also be used as a dynamic output area that is controlled by scripts. For example, you can display the system’s date and time dynamically at the click of a button.

Figure 5.2 shows this concept.

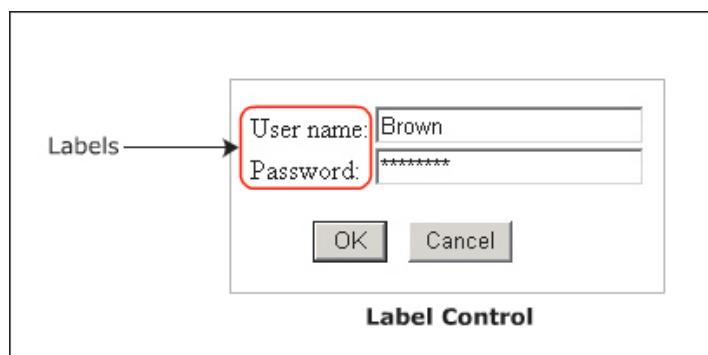


Figure 5.2: Label Control

Syntax:

The following syntax is used to create the Label control.

```
Label <objectname> = new Label();
```

where,

`Label`: Is the built-in class used to create the Label control.

The following code is used to create a Label control named `lblFirstName`.

Code Snippet:

```
Label lblFirstName = new Label();
Page.Controls.Add(myLabel);
```

The label is created and added to the Web Page but it will not display anything as of now on the page unless other properties for it have been set.

5.2.2 Properties and Methods of Label Class

The `Label` class represents the `Label` control. Table 5.2 lists the most commonly used properties and methods of the `Label` class.

Name	Description
AccessKey property	Specifies or retrieves the shortcut key that allows quick navigation to the Label control.
AssociatedControlID property	Specifies or retrieves the ID of the associated control.
BorderStyle property	Specifies or retrieves the border style of the Label control.
EnableViewState property	Specifies or retrieves a value indicating whether the Label control maintains its view state.
Page property	Retrieves a reference to the Page instance containing the Label control.
Text property	Specifies the text to be displayed in the label.
HasControls() method	Determines whether or not the Label control has any child controls.
ToString() method	Returns a string representing the current object.

Table 5.2:Properties of Label Class

The following code assigns values to various properties of the `Label` control as well as sets the focus on the control.

Code Snippet:

```
Label lblFirstName = new Label();
lblFirstName.AccessKey = "F";
lblFirstName.Text = "First Name:";
//assuming that a control named txtFirstName has been created, //set it to be the
associated control
lblFirstName.AssociatedControlID = "txtFirstName";
lblFirstName.EnableViewState = true;
frmStudentDetails.Controls.Add(lblFirstName);
```

In the code, a `Label` control named `lblFirstName` is created. `F`, `First Name:` and `txtFirstName` are the values specified for the `lblFirstName.AccessKey`, `lblFirstName.Text` and `lblFirstName.AssociatedControlID` properties of the `Label` control respectively. `EnableViewState` property value is set to `true`, which maintains the view state of the `Label` control. The control is added to the form `frmStudentDetails` using `Add()` method.

5.2.3 TextBox Control

The `TextBox` control allows you to display a text box on a Web page. The text box is generally used to accept the input from the user. For example, in a simple login page, the rectangular areas in which you enter the user name and password are text boxes created using the `TextBox` control.

Even though text boxes are mostly used for accepting input, they can also be used to display output. The `TextBox` control can display the text either in a single line or in multiple lines. This is based on the value set for the `TextMode` property of this control. You can set the property of the `TextBox` control to a single line for displaying short text entries such as name, age, and so on. For displaying long text entries such as addresses, you can use the multi-line text box.

Syntax:

The following syntax is used to create the `TextBox` control.

```
TextBox <objectname> = new TextBox();
```

where,

`TextBox`: Is the built-in class used to create the Text box control.

The following code is used to create a `TextBox` control named `txtAddress`.

Code Snippet:

```
TextBox txtAddress = new TextBox();
```

5.2.4 Properties, Methods, and Events

The `TextBox` class allows you to include `TextBox` control on the Web page. Table 5.3 lists the commonly used properties, methods and events of the `TextBox` class.

Name	Description
<code>AccessKey</code> property	Specifies or retrieves the shortcut key that allows quick navigation to the <code>TextBox</code> control.
<code>AutoPostBack</code> property	Specifies or retrieves a value indicating whether the post back occurs automatically when the user presses the ENTER or the TAB key.
<code>MaxLength</code> property	Specifies or retrieves the maximum length of characters allowed in the <code>TextBox</code> control.
<code>Text</code> property	Specifies or retrieves the text to be displayed in the <code>TextBox</code> control.
<code>TextMode</code> property	Specifies or retrieves the behavior mode, namely single-line, multi-line, or password of the <code>TextBox</code> control.
<code>ToString()</code> method	Returns a string representing the current object.
<code>TextChanged</code> event	Occurs when there is a change in the content of the <code>TextBox</code> control between posts.

Table 5.3: Properties of `TextBox` Class

The following code assigns values to various properties of the `TextBox` control, sets focus on the control, as well as uses the `TextChanged` event to indicate if the text in the control has been changed.

Code Snippet:

```

...
TextBox txtAddress = new TextBox();
txtAddress.AccessKey = "A";
txtAddress.Text = "14, Riverside";
txtAddress.MaxLength = 10;
txtAddress.TextMode = TextBoxMode.MultiLine;
frmEmployeeDetails.Controls.Add(txtAddress);
...
void txtAddress_TextChanged(object sender, EventArgs e)
{
    Response.Write("Hello!");
}

```

In this code, a TextBox control named txtAddress is created. A and 14, River side are the values assigned to the AccessKey and Text properties respectively. The maximum character length of the text is set to 10. The behavior mode of TextBoxMode is set to MultiLine. The control is added to the form frmEmployeeDetails using the Add() method. TextChanged event occurs when the content in the text box is changed.

5.2.5 Button Control

Consider a simple login page. Once you have entered the user name and password, you click the 'Submit' button in order to log on to the Web site. Buttons such as 'Submit' are created using the `Button` control of ASP.NET. The `Button` control allows you to add a button on the Web page.

The `Button` control is a clickable control that is primarily used to submit information to the Web server.

A button can either be a submit button or a command button. The submit button is used to trigger a PostBack to the Web server whereas a command button is used to perform actions on the client side.

Syntax:

The following syntax is used to create the `Button` control.

```
Button <objectname> = new Button();
```

where,

`Button`: Is the built-in class used to create the `Button` control.

Code Snippet:

The following code is used to create a `Button` control named `btnSubmit`.

```
Button btnSubmit = new Button();
```

5.2.6 Properties, Methods, and Events

The `Button` class represents the `Button` control. Table 5.4 lists the most commonly used properties, methods and events of the `Button` class.

Name	Description
<code>AccessKey</code> property	Specifies or retrieves the short-cut keys used in Web pages for quick navigation of the buttons.
<code>CausesValidation</code> property	Specifies or retrieves a value indicating whether validation is performed when the <code>Button</code> control is clicked.
<code>Enabled</code> property	Specifies or retrieves a Boolean value indicating whether or not the <code>Button</code> control is enabled.
<code>Text</code> property	Specifies or retrieves the text to be displayed on the <code>Button</code> control.

Name	Description
Click event	Occurs when the button is clicked.
Command event	Occurs when the button is clicked.

Table 5.4: Properties of Button Class

The following code assigns values to various properties and the Click event of the Button control.

Code Snippet:

```
Button btnSubmit = new Button();
btnSubmit.Text = "Submit";
btnSubmit.AccessKey = "S";
btnSubmit.Enabled = true;
btnSubmit.CausesValidation = true;
frmRegistration.Controls.Add(btnSubmit);
...
void btnSubmit_Click(object sender, EventArgs e)
{
    Response.Write("You clicked the Submit button.");
}
```

In this code, a Button control named btnSubmit is created. Submit and S are values assigned to Text and AccessKey properties of the Button control respectively. The Enabled and CausesValidation properties of the Button control are set to true. This specifies that the Button control is enabled on the Web page and the validation is performed when the Button control is clicked. The control is added to the form frmRegistration using the Add() method. The Click event is fired when the button is clicked.

5.2.7 Image Control

Often, when you create GUIs, you will need to use some form of pictorial data such as logos, images and photographs.

The Image control in ASP.NET allows you to include graphic images on Web pages. The Image class represents Image Web server control.

Table 5.5 lists the most commonly used properties and methods of the `Image` class.

Name	Description
<code>AlternateText</code> property	Specifies or retrieves the alternate text to be displayed in place of the <code>Image</code> control when the image cannot be displayed.
<code>AlternateText</code> property	Specifies or retrieves a value indicating whether the <code>Image</code> control maintains its view state.
<code>ImageAlign</code> property	Specifies or retrieves the alignment of the <code>Image</code> on the Web page.
<code>ImageUrl</code> property	Specifies or retrieves the location of the image to be displayed in the <code>Image</code> control.
<code>ToolTip</code> property	Specifies or retrieves the text to be displayed when the mouse pointer is hovered over the <code>Image</code> control.
<code>GetType()</code> method	Retrieves the type of the current instance.
<code>ToString()</code> method	Returns a string representing the current object.

Table 5.5: Properties of `Image` Class

The following syntax is used to create the `Image` control.

Syntax:

```
Image <objectname> = new Image();
```

where,

`Image`: Is the built-in class used to create the `Image` control.

The following code assigns values to various properties of the `Image` control.

Code Snippet:

```
Image imgFlower = new Image();
imgFlower.ImageUrl = "~\\Rose.jpg";
imgFlower.AlternateText = "Rose";
imgFlower.EnableViewState = true;
imgFlower.ImageAlign = ImageAlign.Middle;
imgFlower.ToolTip = "A king of flower, Rose";
frmProductDetails.Controls.Add(imgFlower);
```

In the code, an `Image` control named `imgFlower` is created. `~\\Rose.jpg` and `Rose` are the values assigned to `ImageUrl` and `AlternateText` properties of the `Image` control. `EnableViewState` property value is set to `true`, which indicates that the `Image` control maintains its view state. The `Image` control is centre-aligned using the `ImageAlign` property. The `ToolTip` property sets the tool tip for the `Image` control. The control is added to the form `frmProductDetails` using the `Add()` method.

5.2.8 ImageButton Control

The `ImageButton` control allows you to add clickable images on the Web page. These clickable images act like buttons by responding to mouse-click events.

The `ImageButton` class represents the `ImageButton` control. Table 5.6 lists the commonly used properties, methods, and events of the `ImageButton` class.

Name	Description
<code>AlternateText</code> property	Specifies or retrieves the alternate text to be displayed in place of the <code>ImageButton</code> control when the image cannot be displayed.
<code>ImageAlign</code> property	Specifies or retrieves the alignment of the <code>ImageButton</code> control.
<code>ImageUrl</code> property	Specifies or retrieves the URL of an image to display in the <code>ImageButton</code> control.
<code>OnClientClick</code> property	Specifies or retrieves the client-side script that is executed when a click event occurs on an <code>ImageButton</code> control.
<code>GetType()</code> method	Retrieves the type of the current instance.
<code>ToString()</code> method	Returns a string representing the current object.
<code>Click</code> event	Occurs when the <code>ImageButton</code> control is clicked.
<code>Command</code> event	Occurs when the <code>ImageButton</code> control is clicked.

Table 5.6: Properties of `ImageButton` Class

The following syntax is used to create the `ImageButton` control.

Syntax:

```
ImageButton <objectname> = new ImageButton();
```

where,

`ImageButton`: Is the built-in class used to create the `ImageButton` control.

The following code assigns values to various properties of the `ImageButton` control and sets the focus on the control.

Code Snippet:

```
ImageButton ibtnNext = new ImageButton();
ibtnNext.ImageUrl = "~\\Next.gif";
ibtnNext.AlternateText = "Next";
ibtnNext.ImageAlign = ImageAlign.Middle;
```

```

frmProductDetails.Controls.Add(ibtnNext);

...
...

void ibtnNext_Click(object sender, ImageClickEventArgs e)
{
    Response.Redirect("CustomerDetails.aspx");
}

```

In the code, an `ImageButton` control named `ibtnNext` is created. `~\\Next.gif` and `Next` are the values assigned to `ImageUrl` and `AlternateText` properties of the `ImageButton` control respectively. The property `ImageAlign` is assigned value `Middle`, which specifies that the `ImageButton` control is centrally aligned. The control is added to the form `frmProductDetails` using the `Add()` method. When the `ImageButton` is clicked, the `Click` event is triggered and the user is redirected to the `CustomerDetails.aspx` page.

5.2.9 LinkButton Control

The `LinkButton` control is used to display, on the Web page, a button that acts as a hyperlink. This button is not displayed as a button but appears as a text link.

The `LinkButton` control should not be confused with the `HyperLink` control. The `HyperLink` control is simply used to display Web pages when clicked but the `LinkButton` control performs the same task as the `Button` control. When you click the `LinkButton` control, a `PostBack` is triggered to the Web server.

The following syntax is used to create the `LinkButton` control.

Syntax:

```
LinkButton<objectname> = new LinkButton();
```

where,

`LinkButton`: Is the built-in class used to create the `LinkButton` control.

The following code is used to create a `LinkButton` control named, `lnkbtnClickHere`.

Code Snippet:

```
LinkButton lnkbtnClickHere = new LinkButton();
```

5.2.10 Properties, Methods, and Events

The `LinkButton` class represents the `LinkButton` control.

Table 5.7 lists the commonly used properties and events of the LinkButton class.

Name	Description
CausesValidation property	Specifies or retrieves a value indicating whether validation is performed when the LinkButton control is clicked.
Enabled property	Specifies or retrieves a value indicating whether or not the LinkButton control is enabled.
Font property	Specifies or retrieves a value indicating whether or not the LinkButton control is enabled.
PostBackUrl property	Specifies or retrieves the URL of the page to which the data has to be posted when the LinkButton is clicked.
Click event	Occurs when the LinkButton control is clicked.
Command event	Occurs when the LinkButton control is clicked.

Table 5.7: Properties of LinkButton Class

The following code assigns values to various properties of the LinkButton control.

Code Snippet:

```
LinkButton lnkbtnClickHere = new LinkButton();
lnkbtnClickHere.Text = "Click Here";
lnkbtnClickHere.Enabled = true;
lnkbtnClickHere.CausesValidation = true;
frmMagazineDetails.Controls.Add(lnkbtnClickHere);
...
void lnkbtnClickHere_Click(object sender, EventArgs e)
{
    Response.Redirect("UserRegister.aspx");
}
```

In the code, a LinkButton control named lnkbtnClickHere is created. Click is the value assigned to the Text property of the LinkButton control. The Enabled and CausesValidation properties of the LinkButton control are set to true. It specifies that the LinkButton control is enabled on the Web page and the validation is performed when the LinkButton control is clicked. The control is added to the form frmMagazineDetails using the Add() method. When the LinkButton control is clicked, the Click event is triggered and the user is redirected to the UserRegister.aspx page.

5.2.11 Panel Control

A Panel control can be described as a container of different Web server controls.

Panel controls are usually used to display static information and output values of the script written for each Web server control. For example, a login Panel control on the Web page consists of Label controls to display the labels, TextBox controls to accept data, and a Submit Button.

The styles applied to the Panel control are also applied on the controls included in the panel.

The following syntax is used to create the Panel control.

Syntax:

```
Panel<objectname> = new Panel();
```

where,

Panel : Is the built-in class used to create the Panel control.

The following code is used to create a Panel control named, panCustomerDetails.

Code Snippet:

```
Panel panCustomerDetails = new Panel();
```

5.2.12 Properties and Methods of the Panel Class

The Panel class represents the Panel control. Panel controls are usually used to display static information and output values of the script written for each Web server control.

Table 5.8 lists the most commonly used properties of the Panel class.

Name	Description
Controls property	Retrieves a ControlCollection object representing the controls included in the Panel control.
EnableViewState property	This property specifies or retrieves a value indicating whether or not the Panel control maintains its view state.
GroupingText property	Specifies or retrieves the caption for the group of controls contained in the Panel control.
HorizontalAlign property	Specifies or retrieves the alignment of the controls in the Panel control horizontally.
Wrap property	Specifies or retrieves a value indicating whether the content wraps within the Panel control.

Table 5.8: Properties of Panel Class

The following code assigns values to various properties of the Panel control.

Code Snippet:

```
Panel panCustomerDetails = new Panel();
panCustomerDetails.EnableViewState = true;
panCustomerDetails.Controls.Add(txtAddress);
panCustomerDetails.Visible = true;
panCustomerDetails.Wrap = true;
panCustomerDetails.GroupingText = "Customer Details";
frmRegistration.Controls.Add(panCustomerDetails);
```

In this code, a Panel control named panCustomerDetails is created. EnableViewState, Visible, and Wrap properties of the Panel control are set to true. The Wrap property determines whether the content wraps within the Panel control. Customer Details value is assigned to the GroupingText property. The control is added to the form frmRegistration using the Add() method.

Knowledge Check 2

- Can you match the properties of various Web server controls with their corresponding description?

Descriptions		Component	
(A)	Retrieves a ControlCollection object representing controls included in the panel.	(1)	AssociatedControlID
(B)	Retrieves the ID of the associated control.	(2)	AccessKey
(C)	Specifies or retrieves the foreground color of the LinkButton control.	(3)	OnClientClick
(D)	Specifies or retrieves the short-cut keys for quick navigation of the buttons.	(4)	ForeColor
(E)	Specifies or retrieves the client-side script when a click event occurs.	(5)	Controls

(A)	(A)-(5), (B)-(1), (C)-(4), (D)-(2), (E)-(3)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(1), (C)-(4), (D)-(5), (E)-(2)	(D)	(A)-(4), (B)-(3), (C)-(5), (D)-(1), (E)-(2)

2. Which of the following statements about the basic Web server controls are true?

(A)	TextBox control allows accepting input as well as displaying output.
(B)	Label control acts as a dynamic output area that is controlled by scripts.
(C)	The Image control allows inclusion of clickable images on the Web page.
(D)	The Submit button allows triggering a PostBack to the Web server.
(E)	The LinkButton control performs the same task as the Hyperlink control.

(A)	A, B, D	(C)	C, E
(B)	C, D	(D)	A, C

3. Which of the following codes will help you create text boxes to accept user Id and password and also include OK and Cancel buttons?

(A)	<pre>Label lblId = new Label(); Label lblPassword = new Label(); TextBox txtId = new TextBox(); TextBox txtPassword = new TextBox(); Button btnOk = new Button(); Button btnCancel = new Button(); txtPassword.Mode = TextBoxMode.Password; txtId.MaxLength = 20; btnCancel.AccessKey = "O"; btnCancel.BackColor = System.Drawing.Color.Beige;</pre>
(B)	<pre>Label lblId = new Label(); Label lblPassword = new Label(); TextBox txtId = new TextBox(); TextBox txtPassword = new TextBox(); Button btnOk = new Button(); Button btnCancel = new Button(); txtPassword.TextMode = TextBoxMode.Password; txtId.MaxLength = 20; btnCancel.AccessKey = "O"; btnCancel.BackColor = System.Drawing.Color.Beige;</pre>

(C)	<pre> Label lblId = new Label(); Label lblPassword = new Label(); TextBox txtId = new TextBox(); TextBox txtPassword = new TextBox(); Button btnOk = new Button(); Button btnCancel = new Button(); txtPassword.TextMode = TextMode.Password; txtId.MaxLength = 20; btnCancel.AccessKey = "O"; btnCancel.BackColor = System.Drawing.Color.Beige; </pre>
(D)	<pre> Label lblId = new Label(); Label lblPassword = new Label(); TextBox txtId = new TextBox(); TextBox txtPassword = new TextBox(); Button btnOk = new Button(); Button btnCancel = new Button(); txtPassword.TextMode = TextBoxMode.Password; txtId.MaximumLength = 20; btnCancel.AccessKey = "O"; btnCancel.BackColor = System.Drawing.Color.Beige; </pre>

(A)	A	(C)	C
(B)	B	(D)	D

Module Summary

In this module, **Basic Web Server Controls**, you learnt about:

→ **Introduction to Web Server Controls**

The Visual Studio IDE provides four categories of controls, namely HTML server controls, Web server controls, validation controls, and user controls. Web server controls provide easy interactivity with users by allowing them to perform actions such as clicking buttons and entering values. The `System.Web.UI.WebControls` namespace contains classes representing the various Web server controls.

→ **Basic Web Server Controls**

`Label`, `TextBox`, `Button`, `Image`, `ImageButton`, `LinkButton`, and `Panel` are some of the basic Web server controls. The methods, properties, and events of the classes representing these controls can be used to control the behavior of these Web server controls.

Module - 6

More Web Server Controls

Welcome to the Session, **More Web Server Controls**.

The module introduces you to selection Web server controls such as RadioButton, CheckBox and ListBox. Special controls like the Calendar and AdRotator controls are also explained. Each of these controls assist in making Web Form pages more interactive.

In this module, you will learn about:

- Selection Web Server Controls
- Other Controls

Web Development

http://www



6.1 Selection Web Server Controls

In this first lesson, **Selection Web Server Controls**, you will learn to:

- Identify the classes for Selection Web server controls.
- Explain the `CheckBox` control.
- Describe the `RadioButton` control.
- Explain the `CheckBoxList` control.
- Describe the `RadioButtonList` control.
- Explain the `ListBox` control.
- Describe the `DropDownList` control.

6.1.1 Introduction

Consider a passport application form where applicants have to not only enter data but also select items from the given options by placing tick marks against appropriate items. Now if such a form is to be provided online, the user should be able to make similar selections on the online form. This functionality to make text entries as well as select options from given lists is provided by ASP.NET using Web server controls.

Selection Web server controls are part of Web server controls and belong to `System.Web.UI.WebControls` namespace. These controls include `CheckBox`, `RadioButton`, `CheckBoxList`, `RadioButtonList`, `ListBox`, and `DropDownList`.

Figure 6.1 shows this concept.

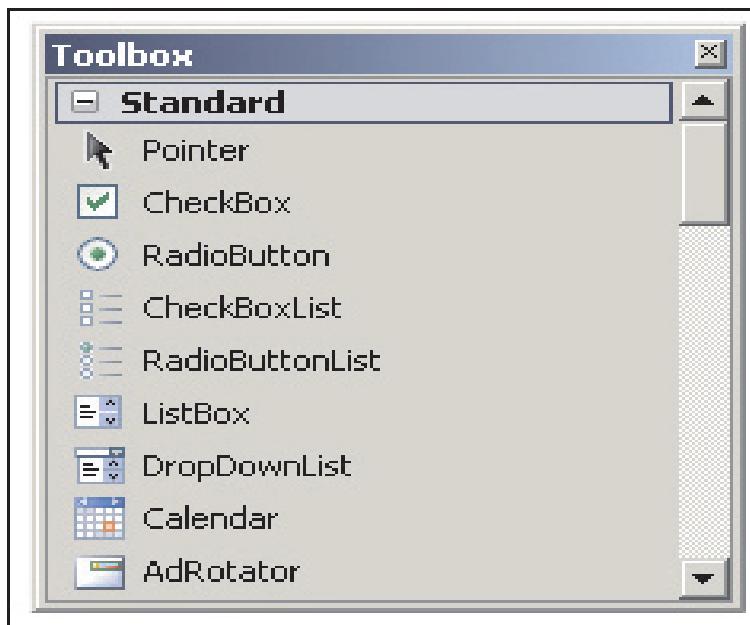


Figure 6.1: Web Server Controls

6.1.2 Selection Web Server Controls

Selection Web server controls are used when the user has to select or deselect a particular option or choose one or more options from a given list. All selection Web server controls are derived from either the `WebControls` class in the `System.Web.UI` namespace or one of the derived classes of the `WebControls` class. Table 6.1 lists the various selection Web server controls and their respective base classes.

Selection Web Server Control Class	Base Class
CheckBox	<code>System.Web.UI.WebControls</code>
CheckBoxList	<code>System.Web.UI.WebControls.ListControl</code>
DropDownList	<code>System.Web.UI.WebControls.ListControl</code>
ListBox	<code>System.Web.UI.WebControls.ListControl</code>
RadioButton	<code>System.Web.UI.WebControls.CheckBox</code>
RadioButtonList	<code>System.Web.UI.WebControls.ListControl</code>

Table 6.1: Base Classes

6.1.3 CheckBox Control

The CheckBox control is used when the user has to either select or deselect a particular option. For example, the login page of certain Web sites might have a line of text stating “Remember me on this computer”. A small box is provided alongside this text which can be selected by a mouse click. This box is created using the CheckBox control. When this box is selected, a small tick mark appears in the box to indicate its selection.

CheckBox controls allow users to select one or more options from the list of the options.

Figure 6.2 demonstrates this concept.

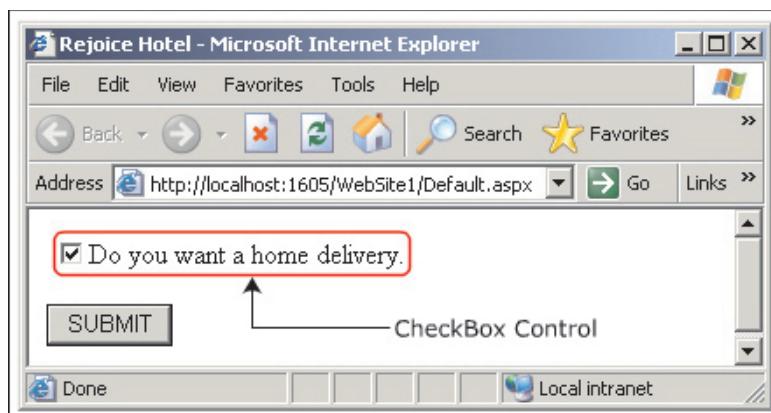


Figure 6.2: CheckBox Control

The following syntax is used to create a CheckBox control.

Syntax:

```
CheckBox <objectname> = new CheckBox();
```

where,

CheckBox: Is the in-built class used to create CheckBox control.

The following code is used to create a CheckBox control named chkEnglish.

Code Snippet:

```
CheckBox chkEnglish = new CheckBox();
```

6.1.4 Properties and Events

The CheckBox control is derived from the WebControl class and thus, inherits the various in-built properties and events of this class.

→ Properties

The properties of the CheckBox class handle the appearance and the layout of the CheckBox control.

Table 6.2 lists the commonly used properties of this class.

Property	Description
AutoPostBack	Specifies or retrieves the value of the control. Depending on the value, it automatically posts back to the server when clicked.
Checked	Specifies or retrieves the state of the control.
Text	Specifies or retrieves the text that is assigned to the control.
Width	Specifies or retrieves the width of the control.

Table 6.2: Properties of CheckBox

The following code assigns properties to a newly created instance of the CheckBox control.

Code Snippet:

```
CheckBox chkEnglish = new CheckBox();
chkEnglish.ID = "chkEnglish";
chkEnglish.AutoPostBack = true;
chkEnglish.Text = "English";
chkEnglish.Checked = true;
frmStudentDetails.Controls.Add(chkEnglish);
```

In this code, `chkEnglish` is an object of `CheckBox` class. The `ID` property is assigned value `chkEnglish`. `AutoPostBack` property is set to `true` so that there is automatic post back to the server when change events occur. The `Text` property is assigned value `English`. The `Checked` property is set to `true` to maintain the checked state of the `CheckBox` control. The control is added to the form `frmStudentDetails` using `Add()` method. Here, `frmStudentDetails` is not the class name or file name, but the HTML form element name.

→ Events

The `CheckBox` control has in-built events that can be used to trigger different actions. Table 6.3 shows an important event that is associated with the `CheckBox` control.

Event	Description
CheckedChanged	It takes place when the boolean value of the control changes.

Table 6.3: Events of CheckBox

The following code demonstrates the use of CheckedChanged event of the CheckBox control. The event is associated with the event handler at runtime by using the += operator.

Code Snippet:

```
CheckBox chkEnglish = new CheckBox();  
...  
...  
//associating the event to the handler method at runtime chkEnglish.  
CheckedChanged += new EventHandler(chkEnglish_CheckedChanged);  
...  
void chkEnglish_CheckedChanged(object sender, EventArgs e)  
{  
    if (chkEnglish.Checked)  
    {  
        Response.Write("You selected the option.");  
    }  
}
```

In this code, `chkEnglish` is an object of `CheckBox` class. `CheckedChanged` is an event that takes place when the boolean value of the `CheckBox` control is changed. During this event, it displays a message using `Response.Write()` method if `Checked` property of `chkEnglish` object is set to true.

6.1.5 RadioButton Control

A `RadioButton` control is used in a Web Form when the user has to select only one of the provided options. For example, if the gender is to be selected, the user can select either male or female, never both. In such a case, if any one option is selected from the list, all other options should automatically get deselected. This functionality of deselecting an option when another is selected is provided by an associated group of radio buttons. The association of multiple radio buttons in a single group is done using the `GroupName` property of the `RadioButton` class.

Figure 6.3 illustrates this concept.



Figure 6.3: RadioButton Control

The following syntax is used to create a RadioButton control.

Syntax:

```
RadioButton <objectname> = new RadioButton();
```

where,

RadioButton: Is the in-built class used to create the RadioButton control.

The following code is used to create a RadioButton control named rdoMale.

Code Snippet:

```
RadioButton rdoMale = new RadioButton();
```

6.1.6 Properties and Events

The RadioButton class inherits the properties, methods, and events of the CheckBox class.

→ **Properties**

The properties of the RadioButton class handle the appearance and the layout of the RadioButton control. Table 6.4 lists some of the commonly used properties.

Property	Description
AutoPostBack	It specifies or retrieves the state of the control and automatically posts back to the server when the option is selected.

Property	Description
Checked	It specifies or retrieves the state of the control.
GroupName	It specifies or retrieves the name of the group to which the control belongs.
Text	It specifies or retrieves the text that is associated with the control.

Table 6.4: RadioButton Members

The following code creates a RadioButton control and assigns values to its relevant properties.

Code Snippet:

```
RadioButton rdoMale = new RadioButton();
rdoMale.Text = "Male";
rdoMale.ID = "rdoMale";
rdoMale.AutoPostBack = false;
rdoMale.Checked = false;
rdoMale.GroupName = "Gender";
frmEmployeeDetails.Controls.Add(rdoMale);
```

In this code, `rdoMale` is an object of `RadioButton` class. The `Text` property is assigned value `Male`. The `ID` property is assigned value `rdoMale`. `AutoPostBack` property is set to true to ensure automatic postback to the server when change events occur. `Checked` property is set to true to maintain the checked state of the `RadioButton` control. `GroupName` property is set to `Gender`. This property identifies the group to which the radio button belongs. The control is added to the form `frmEmployeeDetails` using `Add()` method.

→ Events

Table 6.5 shows a commonly used event of the `RadioButton` control.

Event	Description
CheckedChanged	It takes place when the boolean value of the control changes.

Table 6.5: RadioButton Events

The following code demonstrates how to handle the `CheckedChanged` event of the `RadioButton` control.

Code Snippet:

```
RadioButton rdoMale = new RadioButton();
//code to associate the event handler
```

```
...
void rdoMale_CheckedChanged(object sender, EventArgs e)
{
if (rdoMale.Checked)
{
Response.Write("You selected the option Male.");
}
}
```

In this code, `rdoMale` is an object of `RadioButton` class. `CheckedChanged` is an event that takes place when the boolean value of the `RadioButton` control is changed. When this event occurs, if `Checked` property of `rdoMale` object is set to `true`, a message is displayed using the `Response.Write()` method.

6.1.7 CheckBoxList Control

Consider an admission form that is given to students at the time of school admissions. This form may list various languages offered to students. The student has the option to select the languages he/she wishes to take for the next semester. To select the languages, the student has to simply tick the check boxes provided alongside the languages. A similar functionality of selecting multiple options from a given list can be provided in online forms using the `CheckBoxList` control.

The `CheckBoxList` control can be considered as a group of logically related `CheckBox` controls. This control is used when the user has the option to select more than one option from a given list and the listed options are not mutually exclusive.

Figure 6.4 demonstrates this concept.

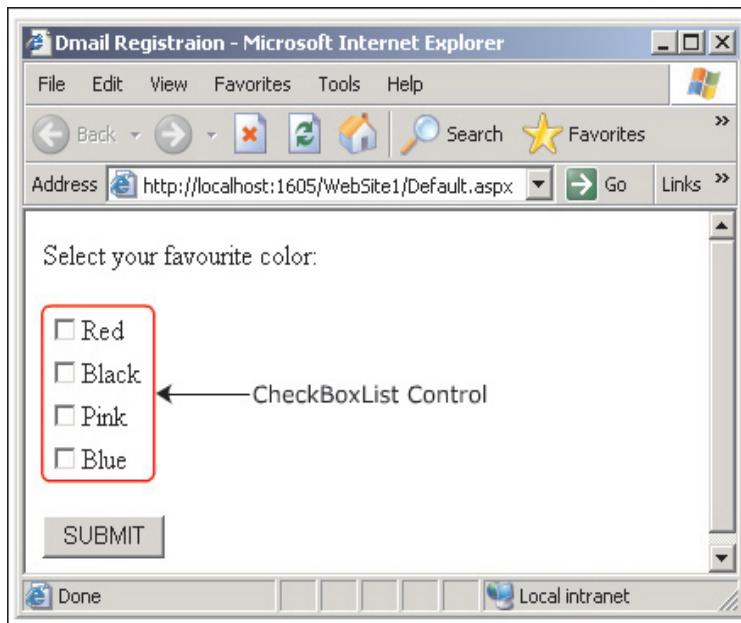


Figure 6.4: CheckBoxList Control

The following syntax is used to create a CheckBoxList control.

Syntax:

```
CheckBoxList <objectname> = new CheckBoxList();
```

where,

CheckBoxList: Is the in-built class used to create CheckBoxList control.

The following code is used to create a CheckBoxList control named chklstColors.

Code Snippet:

```
CheckBoxList chklstColors = new CheckBoxList();
```

6.1.8 Properties, Methods, and Events

The CheckBoxList class inherits all properties, methods, and events of the ListControl class.

→ Properties

The properties of the CheckBoxList class handle the appearance and the layout of the CheckBoxList control. Table 6.6 lists the commonly used properties of this class.

Property	Description
AutoPostBack	It specifies or retrieves the state of the control and automatically posts back to the server when the option is selected.

Property	Description
Items	It retrieves the items that are saved in the control.
SelectedIndex	From the selected list of items, it specifies or retrieves the index number of the item having the lowest index number. For example, if you select the fifth, seventh, and eighth items from the list, it will specify or retrieve the index number of the fifth item.
SelectedValue	It retrieves the value of the items that are selected in the list.
Text	It specifies or retrieves the text that is associated with the control.

Table 6.6: CheckBoxList Properties

The following code demonstrates use of the CheckBoxList control.

```
CheckBoxList chklistColors = new CheckBoxList();
chklistColors.AutoPostBack = true;
chklistColors.Items.Add("Red");
Response.Write("You selected" + chklistColors.SelectedValue);
frmProductDetails.Controls.Add(chklistColors);
```

In this code, `chklistColors` is an object of `CheckBoxList` class. `AutoPostBack` property is set to true so that changes can be posted back to the server as soon as a change event occurs. The `Add` method of `Items` property adds an item named `Red` to the `CheckBoxList` control. The value of the selected item is displayed using `.SelectedValue` property of the control. The control is added to the form `frmProductDetails` using `Add()` method.

→ Methods and Events

Table 6.7 lists the commonly used methods and events of the `CheckBoxList` control.

Method and Event	Description
ClearSelection() Method	It erases the list of the items that have been selected and sets the <code>selected</code> property of all the items to false.
SelectedIndexChanged Event	It occurs if a different item on the list is selected than what was selected at the last post to the server.

Table 6.7: CheckBoxList Members

The following code calls the `ClearSelection` method and demonstrates the use of the `SelectedIndexChanged` event of `CheckBoxList` control.

Code Snippet:

```
CheckBoxList chklistColors = new CheckBoxList();
chklistColors.ClearSelection();
...
...
void chklistColors_SelectedIndexChanged (object sender, EventArgs e)
{
    Response.Write("You selected the item.");
}
```

In this code, `chklistColors` is an object of `CheckBoxList` class. `ClearSelection` is a method used to clear the list of the items that have been selected. `SelectedIndexChanged` is an event that takes place when the selections in the `CheckBoxList` control changes. During this event, it displays a message on the Web page using `Response.Write` method.

6.1.9 RadioButtonList Control

Radio buttons are generally never used individually. Two or more buttons are always grouped together using a single group name. The `RadioButtonList` control on the other hand provides a single control that can act as a group of radio buttons. This control is derived from the `ListControl` class. Thus, working with this control is similar working with other list controls such as the `CheckBoxList` control.

`RadioButtonList` control allows easier data-binding than multiple `RadioButton` controls grouped together. It also makes it easier to detect which radio button on the list has been selected. You do not need to check each button individually to determine its selection status. You can simply retrieve the index number of the selected item on the list using the `SelectedIndex` property of the control.

Figure 6.5 demonstrates this concept.

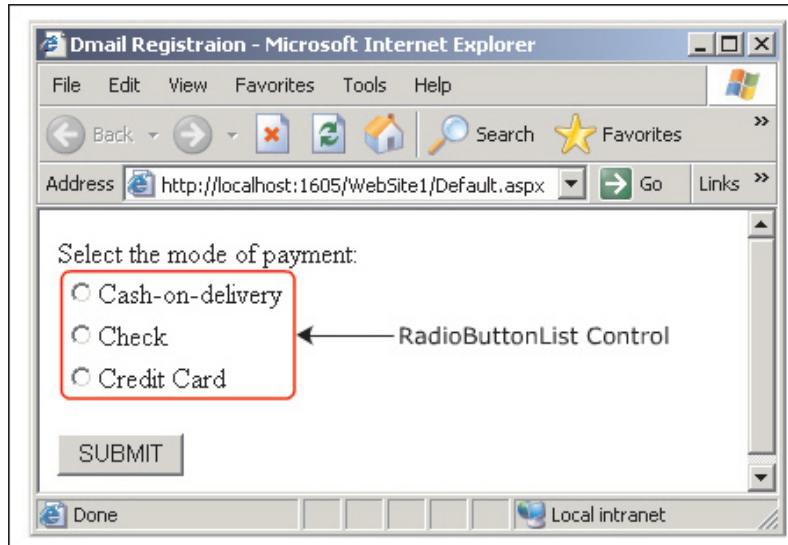


Figure 6.5: RadioButtonList Control

The following syntax is used to create RadioButtonList control.

Syntax:

```
RadioButtonList <objectname> = new RadioButtonList();
```

where,

RadioButtonList: Is the in-built class used to create RadioButtonList control.

The following code is used to create a RadioButtonList control named rdolstGender.

Code Snippet:

```
RadioButtonList rdolstGender = new RadioButtonList();
```

6.1.10 Properties, Methods, and Events

The RadioButtonList class inherits the properties, methods, and events of the ListControl class.

→ **Properties**

The properties of the RadioButtonList class handle the appearance and the layout of the RadioButtonList control.

Table 6.8 lists the commonly used properties of this class.

Property	Description
AutoPostBack	It specifies or retrieves the state of the control and automatically posts back to the server when the option is selected.
Items	It retrieves the items that are saved in the control.
SelectedIndex	It specifies or retrieves the index number of the selected item.
SelectedValue	It retrieves the value of the item selected from the list.
Text	It specifies or retrieves the text that is associated with the control.

Table 6.8: Properties of RadioButtonList

The following code assigns properties to RadioButtonList control.

Code Snippet:

```
RadioButtonList rdolstGender = new RadioButtonList();
rdolstGender.ID = "rdolstGender";
rdolstGender.Items.Add("Male");
rdolstGender.AutoPostBack = true;
Response.Write("You selected" + rdolstGender.SelectedIndex);
Response.Write("You selected" + rdolstGender.SelectedValue);
frmStudentDetails.Controls.Add(rдолстGender);
```

In this code, rdolstGender is an object of RadioButtonList class. ID property of rdolstGender is assigned value rdolstGender. The Add method of Items property is used to add an item named Male. AutoPostBack property is set to true so that it can post back automatically to the server when a change event occurs. The values of SelectedIndex and SelectedValue properties are displayed using the Response.Write() method. The control is added to the form frmStudentDetails using Add() method.

→ **Methods and Events**

Table 6.9 lists the commonly used methods and events of the RadioButtonList control.

Method and Event	Description
ClearSelection() Method	It sets the Selected property of all the items to false.

Method and Event	Description
SelectedIndexChanged Event	It occurs if a different item on the list is selected than what was selected at the last post to the server.
TextChanged Event	It occurs when the values of the Text property and the SelectedValue property of the control change.

Table 6.9: RadioButtonList Members

The following code is used to call the `ClearSelection` method and demonstrate the use of the `SelectedIndexChanged` event of `RadioButtonList` control.

Code Snippet:

```
RadioButtonList rdolstGender = new RadioButtonList();
rdolstGender.ClearSelection();
...
void rdolstGender_SelectedIndexChanged(object sender, EventArgs e)
{
    Response.Write("You have selected the item.");
}
```

In this code, `rdolstGender` is an object of `RadioButtonList` class. `ClearSelection` is a method used to uncheck the selected item on the list. `SelectedIndexChanged` is an event that takes place when the selection in the `RadioButtonList` control is changed. When this event occurs, the message "You have selected the item." is displayed using the `Response.Write()` method.

6.1.11 ListBox Control

The `ListBox` control is used to allow the user to select multiple items from a list. The items of the list are displayed in a scrollable box. For example, consider a list of countries of the world from which you have to select the names of only those countries that play international rugby. To allow the user to make such a selection, the list of countries can be provided in a list box. Here, the user can select one or more items from the list of items.

Figure 6.6 demonstrates this concept.

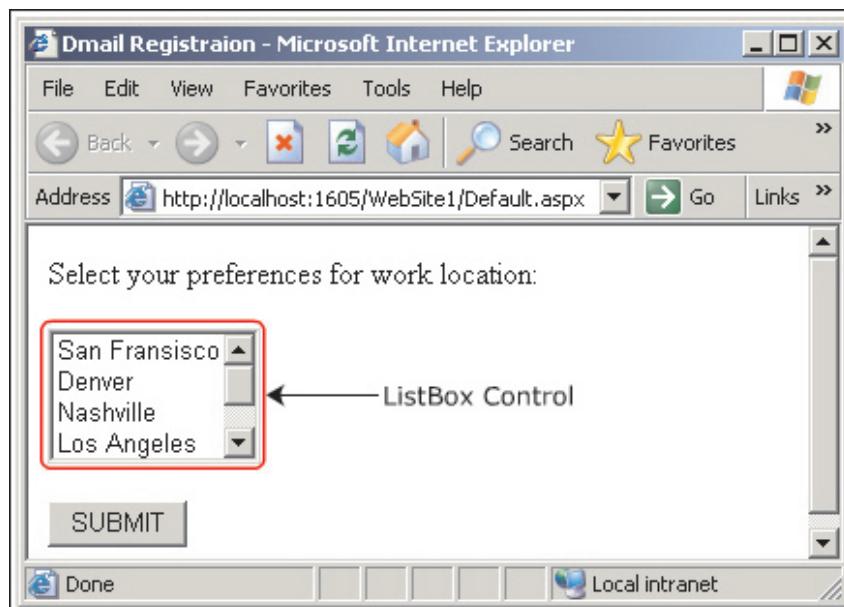


Figure 6.6: ListBox Control

The following syntax is used to create a `ListBox` control.

Syntax:

```
ListBox <objectname> = new ListBox();
```

where,

`ListBox`: Is the in-built class used to create `ListBox` control.

The following code is used to create a `ListBox` control named, `lstLanguage`.

Code Snippet:

```
ListBox lstLanguage = new ListBox();
```

6.1.12 Properties, Methods, and Events

The `ListBox` class inherits the properties, methods, and events of the `ListControl` class.

→ Properties

The properties of the `ListBox` class handle the appearance and the layout of the `ListBox` control.

Table 6.10 lists the commonly used properties of this class.

Property	Description
AutoPostBack	It specifies or retrieves the state of the control and automatically posts back to the server when the option is selected.
Items	It retrieves the items that are saved in the control.
SelectedIndex	From the selected list of items, it specifies or retrieves the index number of the item having the lowest index number. For example, if you select the fifth, seventh, and eighth items from the list, it will specify or retrieve the index number of the fifth item.
SelectedItem	It retrieves the item having the lowest index number from the selected list of items.
SelectionMode	It specifies or retrieves the mode of selection in the control.

Table 6.10: ListBox Properties

The following code is used to create an instance of `ListBox` control and assign values to its properties.

Code Snippet:

```
ListBox lstLanguage = new ListBox();
lstLanguage.ID = "lstLanguage";
lstLanguage.Items.Add("English");
//further code to populate the list
...
lstLanguage.SelectionMode = ListSelectionMode.Multiple;
lstLanguage.AutoPostBack = true;
frmEmployeeRegistration.Controls.Add(lstLanguage);
```

In this code, `lstLanguage` is an object of `ListBox` class. The `ID` property is assigned the value `lstLanguage`. The `Add` method of `Items` property is used to add an item named `English`. `SelectionMode` is the property used to set the mode of selection as `Multiple`. `AutoPostBack` property is set to `true`. The control is added to the form `frmEmployeeRegistration` using `Add()` method.

→ Methods and Events

Table 6.11 lists the commonly used methods and events of the `ListBox` control.

Method and Event	Description
<code>ClearSelection()</code> Method	It erases the list of the items that have been selected and sets the <code>Selected</code> property of all the items to false.
<code>GetSelectedIndices()</code> Method	It retrieves the array of the index numbers for the currently selected items in the control.
<code>SelectedIndexChanged</code> Event	It occurs if there is a change in the selections from the list between posts to the server.
<code>TextChanged</code> Event	It takes place when the <code>Text</code> and <code>SelectedValue</code> properties of the control are changed.

Table 6.11: ListBox Members

The following code is used to set `GetSelectedIndices` method and demonstrate `SelectedIndexChanged` event of `ListBox` control.

Code Snippet:

```
ListBox lstLanguage = new ListBox();
//code to populate the list
...
void lstLanguage_SelectedIndexChanged(object sender, EventArgs e)
{
    if (lstLanguage.GetSelectedIndices().Length > 1)
    {
        Response.Write("You have selected two or more languages.");
    }
    else
        Response.Write("You have selected the language." + lstLanguage.
SelectedItem);
}
```

In this code, `lstLanguage` is an object of `ListBox` class. `Length` is a property of `GetSelectedIndices` that retrieves a length of array of currently selected items in the control. If a length of the array is greater than 1, it displays an appropriate message on the Web page. `SelectedIndexChanged` is an event that takes place when the selections in the `ListBox` control are changed. When this event occurs, the message "You have selected the item." is displayed using the `Response.Write()` method.

6.1.13 DropDownList Control

The `DropDownList` control is used to create a drop-down list. Each item in a drop-down list is a selectable item. However, unlike the `ListBox` control, the `DropDownList` control supports only a single selection. Hence, the `DropDownList` control is used to list selectable items that are mutually exclusive. The listed items are displayed in a drop-down manner. The drop-down control is seen as a single-line rectangular box with a small downward-pointing clickable arrow at the right end of the box. When the arrow is clicked, a list of items is displayed. When you select an item from this list, the selected item is displayed in the box.

For example, to allow the user to enter date of birth, you can provide three `DropDownList` controls, one each for the day, month, and the year selections. Each of these lists will allow only a single item to be selected.

Figure 6.7 demonstrates this concept.



Figure 6.7: DropDownList Control

The following syntax is used to create a `DropDownList` control.

Syntax:

```
DropDownList <objectname> = new DropDownList();
```

where,

`DropDownList`: Is the in-built class used to create the `DropDownList` control.

The following code is used to create a DropDownList control named `ddlEducation`.

Code Snippet:

```
DropDownList ddlEducation = new DropDownList();
```

6.1.14 Properties, Methods, and Events

The DropDownList class inherits all methods, properties, and events of the ListControl class.

→ **Properties**

The properties of the DropDownList class handle the appearance and the layout of the DropDownList control. Table 6.12 lists the commonly used properties of this class.

Property	Description
AutoPostBack	It specifies or retrieves the state of the control and automatically posts back to the server when the option is selected.
Items	It retrieves the items that are saved in the control.
SelectedIndex	It specifies or retrieves the index number of the selected item.
SelectedItem	It retrieves the selected item.
Text	It specifies or retrieves the SelectedValue property of the control.

Table 6.12: DropDownList Properties

The following code assigns properties to the DropDownList control.

Code Snippet:

```
DropDownList ddlEducation = new DropDownList();
ddlEducation.ID = "ddlEducation";
ddlEducation.Items.Add("Graduate");
ddlEducation.AutoPostBack = true;
ddlEducation.Text = "Select anyone..";
Response.Write("You selected " +
    ddlEducation.SelectedIndex.ToString());
frmStudentAdmission.Controls.Add(ddlEducation);
```

In this code, `ddlEducation` is an object of DropDownList class. The `ID` is assigned value `ddlEducation`. The `Add` method of `Items` property is used to add an item named `Graduate`. The `AutoPostBack` property is set to `true`. The `Text` property is used to provide a label for the control.

The `Response.Write()` function displays the index number of the selected item using the `SelectedIndex` property of the control. The control is added to the form `frmStudentAdmission` using `Add()` method.

→ Methods and Events

Table 6.13 lists the commonly used methods and events of the `DropDownList` control.

Method and Event	Description
<code>ClearSelection()</code> Method	It sets the <code>Selected</code> property of all the items to false.
<code>VerifyMultiSelect()</code> Method	It throws an <code>HttpException</code> as multiple selections are not supported by the control.
<code>SelectedIndexChanged</code> Event	It occurs if the selected item at the current post back is different from the item that was selected at the previous post back.

Table 6.13: DropDownList Members

The following code calls the `ClearSelection` method and demonstrates the use of the `SelectedIndexChanged` event of `DropDownList` control.

Code Snippet:

```
DropDownList ddlEducation = new DropDownList();
ddlEducation.ClearSelection();
...
void ddlEducation_SelectedIndexChanged(object sender, EventArgs e)
{
    Response.Write("You selected the item.");
}
```

In this code, `ddlEducation` is an object of `DropDownList` class. `ClearSelection` is the method used to deselect the selected item on the list. The `SelectedIndexChanged` event occurs when the selection status of the items on the list is changed. When this event occurs, a message is displayed using the `Response.Write()` method.

Knowledge Check 1

1. Which of the following statements about selection Web server controls are true?

(A)	RadioButtonList control belongs to System.Web.UI.RadioButton Class.		
(B)	DropDownList control supports only single selection.		
(C)	CheckBoxList control allows only single selections.		
(D)	RadioButton control inherits the properties of CheckBoxList class.		
(E)	ListBox control supports single as well as multiple selections.		

(A)	A, B, D	(C)	B, E
(B)	A, C	(D)	A, C, E

2. Can you match the properties of various selection Web server controls with their corresponding description?

Description		Property	
(A)	Specifies or retrieves the lowest index number of the items that are selected in the list.	(1)	AutoPostBack
(B)	Specifies or retrieves the state of the control and automatically posts back to the server.	(2)	SelectionMode
(C)	Specifies or retrieves the program identifier that is assigned to the control.	(3)	SelectedIndex
(D)	Specifies or retrieves the mode of selection in the control.	(4)	SelectedValue
(E)	Retrieves the value of the selected item.	(5)	ID

(A)	(A)-(5), (B)-(3), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(2), (C)-(5), (D)-(2), (E)-(4)	(D)	(A)-(4), (B)-(3), (C)-(5), (D)-(1), (E)-(2)

6.2 Other Controls

In this last lesson, **Other Controls**, you will learn to:

- State the use of the Calendar control.
- State the use of AdRotator control.

6.2.1 Calendar Control

The Calendar control is used to display a calendar in the browser. This control displays only a single-month calendar and allows the user to select dates. By default, this control displays the calendar for the current month. It displays the name of the month, the day headings for the days of the week, and the dates under corresponding days.

It also displays directional arrows that can be used for navigating to the previous or the next month.

Figure 6.8 demonstrates this concept.

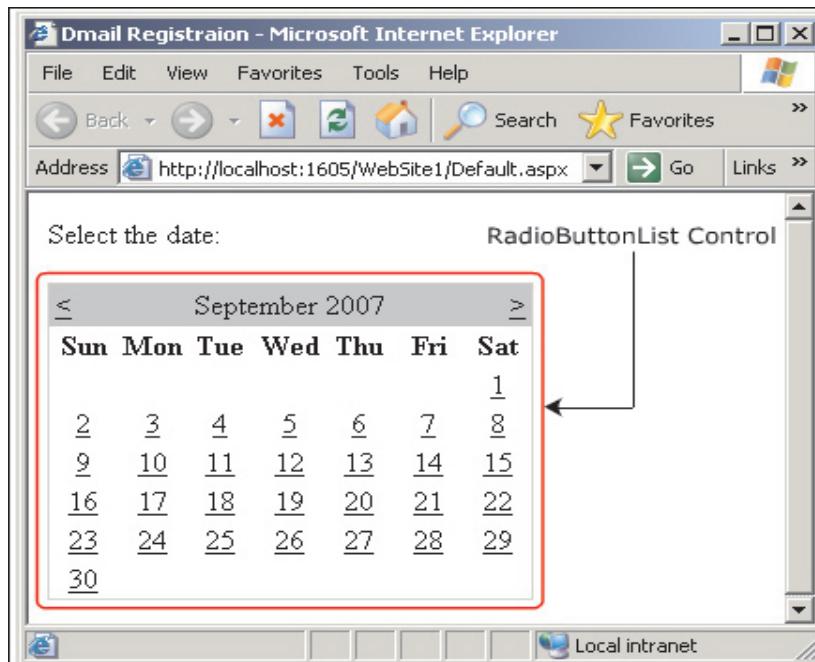


Figure 6.8: Calendar Control

The following syntax is used to create the `Calendar` control.

Syntax:

```
Calendar <objectname> = new Calendar();
```

where,

`Calendar`: Is the in-built class used to create the `Calendar` control.

The following code is used to create a `Calendar` control named `calCompany`.

Code Snippet:

```
Calendar calCompany = new Calendar();
```

6.2.2 Properties and Events

The `Calendar` class represents the `Calendar` control. This class inherits the properties, methods, and events of the `WebControls` class.

→ **Properties**

The properties of the `Calendar` class handle the appearance and the layout of the `Calendar` control.

Table 6.14 lists the commonly used properties of this class.

Property	Description
DayHeaderStyle	It retrieves the style properties for the section that displays the week days.
DayNameFormat	It specifies or retrieves the name format of the days of the week.
DayStyle	It retrieves the style properties of the days in the displayed month.
SelectedDate	It specifies or retrieves the date that is selected.
SelectionMode	It specifies or retrieves the selection mode of the date that enables the user to either select a day, a week, or an entire month.
TodaysDate	It specifies or retrieves the present date from the calendar.

Table 6.14: Calendar Properties

The following code assigns properties to Calendar control.

Code Snippet:

```
Calendar calCompany = new Calendar();
calCompany.SelectionMode = CalendarSelectionMode.DayWeek;
calCompany.DayNameFormat = DayNameFormat.Full;
calCompany.DayHeaderStyle.Font.Bold = true;
calCompany.DayStyle.BackColor = System.Drawing.Color.Blue;
Response.Write("Selected date is: " + calCompany.SelectedDate);
frmCompanyDetails.Controls.Add(calCompany);
```

In this code, `calCompany` is an object of `Calendar` class. The `SelectionMode` property is used to set the selection mode as `DayWeek`, which allows the user to select either a single day or select an entire week. The `DayNameFormat` property is used to set the name format for the days of the week as `Full`. `Bold` is a property of `Font` class of `DayHeaderStyle` and is set to `true`. `BackColor` is a property of `DayStyle` that is set `Blue`. When the control is clicked, it displays the values of `SelectedDate` using `Response.Write()` method. The control is added to the form `frmCompanyDetails` using `Add()` method.

→ Events

Table 6.15 lists the commonly used events of the Calendar control.

Event	Description
DayRender	It occurs when the day is created in the control.
SelectionChanged	It occurs when the user clicks the date-selector control for selecting the day, week, or the month.
VisibleMonthChanged	It occurs when the navigation controls for the previous or next month are selected.

Table 6.15: Calendar Events

The following code demonstrates the use of SelectionChanged and VisibleMonthChanged events of Calendar control.

Code Snippet:

```
Calendar calCompany = new Calendar();
void calCompany_SelectionChanged(object sender, EventArgs e)
{
    Response.Write("You selected the date.");
}
void calCompany_VisibleMonthChanged(object sender, MonthChangedEventArgs e)
{
    Response.Write("The month has been changed.");
}
```

In this code, `calCompany` is an object of `Calendar` class. The `SelectionChanged` event takes place when the date-selector is clicked. When this event occurs, it displays the message 'You have selected the date.' using `Response.Write()` method. The `VisibleMonthChanged` event takes place when navigation controls for the previous or next month are selected. When this event occurs, it displays the message 'The month has been changed.'

6.2.3 AdRotator Control

The `AdRotator` control is a special control in ASP.NET. This control is used to display flashing banner advertisements. The control can be used to display these advertisements randomly or sequentially depending on the properties that have been set by the user. A new advertisement is displayed each time the page is refreshed or reloaded.

The `AdRotator` control uses an XML file to store information regarding the advertisements. This XML file begins and ends with the `<Advertisement>` tag.

Within these tags, there may be multiple <Ad> tags, each of them representing an ad. Figure 6.9 demonstrates the concept.

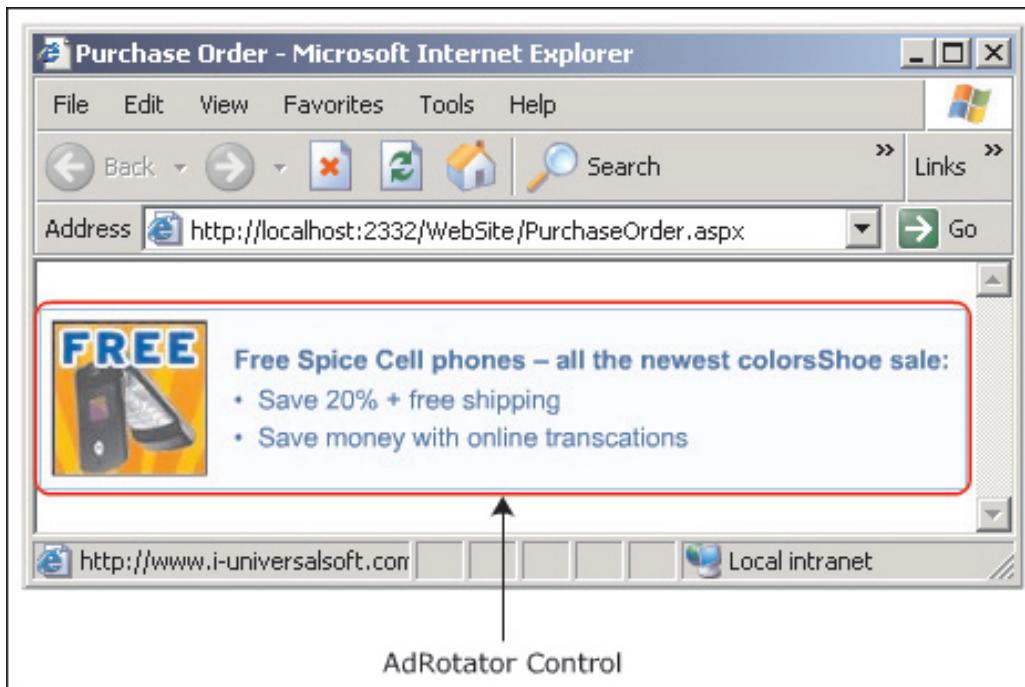


Figure 6.9: AdRotator Control Sample

The <Ad> tag consists of predefined elements such as:

→ **ImageUrl**

`ImageUrl` sets the path for the image that appears in the advertisement.

→ **NavigateUrl**

`NavigateUrl` sets the path of the page to which the user should be taken when the image is clicked.

→ **AlternateText**

`AlternateText` sets the text for the image when the user rolls the mouse on it. `AlternateText` is an optional element.

→ **Keyword**

`Keyword` sets the specific keyword for the ad. `Keyword` is an optional element.

→ **Impressions**

`Impressions` specifies the number of the ads and their order of rotation with respect to other ads in the file.

The following syntax is used to create `AdRotator` control.

Syntax:

```
AdRotator<objectname> = new AdRotator();
```

where,

`AdRotator`: Is the in-built class used to create `AdRotator` control.

The following code is used to create the `AdRotator` control named `adrBanner`.

Code Snippet:

```
AdRotator adrBanner = new AdRotator();
```

6.2.4 Properties, Methods, and Events

The `AdRotator` class represents the `AdRotator` control. This class is derived from the `WebControls` class in the `System.Web.UI` namespace. Hence, it inherits all the methods, properties, and events of the `WebControls` class.

→ **Properties**

The properties of the `AdRotator` class handle the appearance of the `AdRotator` control. Table 6.16 lists the commonly used properties of this class.

Property	Description
<code>AdvertisementFile</code>	It specifies or retrieves the path of the XML file that contains the advertisement information.
<code>AlternateTextField</code>	It specifies or retrieves a data field that can be used in place of the <code>AlternateText</code> element of the <code><Ad></code> tag.
<code>ImageUrlField</code>	It specifies or retrieves a data field that can be used in place of the <code>ImageUrl</code> element of the <code><Ad></code> tag.
<code>KeywordFilter</code>	It specifies or retrieves keywords for specific types of advertisements in the XML advertisement file.
<code>NavigateUrlField</code>	It specifies or retrieves a data field that can be used in place of the <code>NavigateUrl</code> element of the <code><Ad></code> tag.

Table 6.16: `AdRotator` Properties

The following code assigns properties to AdRotator control.

Code Snippet:

```
AdRotator adrBanner = new AdRotator();
adrBanner.AdvertisementFile = "~\\AdBanner.xml";
adrBanner.KeywordFilter = "TravelSites";
adrBanner.ImageUrlField = "Pic";
adrBanner.AlternateTextField = "Alt";
adrBanner.NavigateUrlField = "Nav";
frmMagazineDetails.Controls.Add(adrBanner);
```

In this code, `adrBanner` is an object of `AdRotator` class. `AdvertisementFile` property is used to set the path of the XML file that contains the advertisement information. `KeywordFilter` property is set to `TravelSites`. It specifies the type of advertisements in the XML advertisement file. `ImageUrlField` property is set to `Pic` and can be used in place of the `ImageUrl` element of the `<Ad>` tag. `AlternateTextField` property is set to `Alt` and is used in place of the `AlternateText` element of the `<Ad>` tag. `NavigateUrlField` property is set to `Nav` and can be used in place of `NavigateUrl` element of the `<Ad>` tag. The control is added to the form `frmMagazineDetails` using `Add()` method.

→ **Methods and Events**

Table 6.17 lists the commonly used methods and events of the `AdRotator` control.

Method and Event	Description
<code>Equals()</code> Method	It specifies whether the occurrences of the objects are equal.
<code>Focus()</code> Method	It sets the focus on the control when the control is selected by the user.
<code>ToString()</code> Method	It returns the string that represents the current object in the Web Form.
<code>AdCreated</code> Event	It occurs when the control is created but before the page is rendered on the browser.

Table 6.17: AdRotator Members

The following code demonstrates the use of `AdCreated` event of `AdRotator` control.

Code Snippet:

```
AdRotator adrBanner = new AdRotator();
...
void adrBanner_AdCreated(object sender, AdCreatedEventArgs e)
{
    Response.Write("URL:" + e.NavigateUrl);
}
```

In this code, `adrBanner` is an object of `AdRotator` class. `AdCreated` is an event that occurs when the advertisement is created. When this event occurs, it displays a URL string using `Response.Write()` method.

Knowledge Check 2

- Can you match the properties of the `Calendar` control with their corresponding description?

Description		Property	
(A)	Specifies or retrieves the present date from the calendar.	(1)	SelectedDate
(B)	Specifies or retrieves the name format of the days of the week.	(2)	DayStyle
(C)	Specifies or retrieves the date that is selected.	(3)	TodaysDate
(D)	Retrieves the style properties of the days in the displayed month.	(4)	DayHeaderStyle
(E)	Retrieves the style properties for the section that displays the week days.	(5)	DayNameFormat

(A)	(A)-(5), (B)-(3), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(5), (C)-(2), (D)-(1), (E)-(4)	(D)	(A)-(4), (B)-(3), (C)-(5), (D)-(1), (E)-(2)

2. Which of the following statements about AdRotator control are true?

(A)	The <code>AlternateText</code> property sets a text for the image when the mouse rolls over the image.
(B)	The <code>AdRotator</code> control displays the ads only in a sequential manner.
(C)	The <code>AdRotator</code> control uses an XML file to store information about the advertisement.
(D)	The <code>NavigateUrlField</code> method is used to move between advertisements.
(E)	The <code>AdRotator</code> control inherits the properties from <code>Calendar Class</code> .

(A)	A, B, D	(C)	B, E
(B)	A, C	(D)	A, C, E

Module Summary

In this module, **More Web Server Controls**, you learnt about:

→ Selection Web Server Controls

CheckBox, RadioButton, CheckBoxList, RadioButtonList, ListBox, and DropDownList

are some of the commonly used selection Web server controls. These controls are derived from the WebControl class or its derived classes in the System.Web.UI.WebControls namespace.

Selection Web server controls allow users to select items from provided options.

→ Other Controls

Calendar control displays a single-month calendar on the browser. The AdRotator control is used to display advertisements on the Web page either sequentially or randomly. Both these controls are derived from the WebControl class in the System.Web.UI.WebControls namespace.

Module - 7

Advanced Web Controls

Welcome to the module, **Advanced Web Controls**.

Advanced Web controls are the Web controls that help a user navigate Web pages in a convenient manner. These controls also help in maintenance of Web sites.

In this module, you will learn to:

- Navigation Web Controls
- ImageMap control and HotSpot classes
- Advanced Web server controls

Web Development

http://www



7.1 Navigation Web Controls

In this first lesson, **Navigation Web Controls**, you will learn to:

- List the Navigation Web Controls in ASP.NET 2.0.
- Describe the **Menu** Web server control.
- Explain the **SiteMapPath** Web server control.
- Describe **TreeView** control.

7.1.1 Introduction to Navigation Web Controls

A Web site can contain one or more Web pages. Multiple Web pages in a Web site are linked to each other through hyperlinks. However, managing too many hyperlinks sometimes becomes difficult. Consider a really large Web site such as amazon or ebay, which has hundreds of Web pages logically linked to each other. If these sites were managed using only hyperlinks, the pages would be completely cluttered with links and there would be no logical path to access a page of interest.

ASP.NET 2.0 provides various in-built navigation features that eliminate the need for excessive use of hyperlinks. These features allow defining all links in a single location, which is usually an XML file. The links can then be displayed on the Web pages through navigation menu that are accessed using navigation controls.

Figure 7.1 demonstrates this concept.



Figure 7.1: Navigation Controls Sample

The navigation controls provided by ASP.NET are as follows:

→ **Menu Control**

Consider a simple text editor such as Microsoft Word. When you open Word, somewhere towards the top of the window you can see the **Menu** bar listing various menus such as **File**, **Edit**, and **View**.

A menu can be defined as a list of choices that is represented in textual, graphical, or both textual as well as graphical formats.

ASP.NET 2.0 enables creation of menus using **Menu** control. The menus created in ASP.NET Web applications can have multiple levels of sub-menus.

Menu control allows you to create static as well as dynamic menus. The dynamic menus are also referred to as fly-out menus.

Figure 7.2 demonstrates the concept.

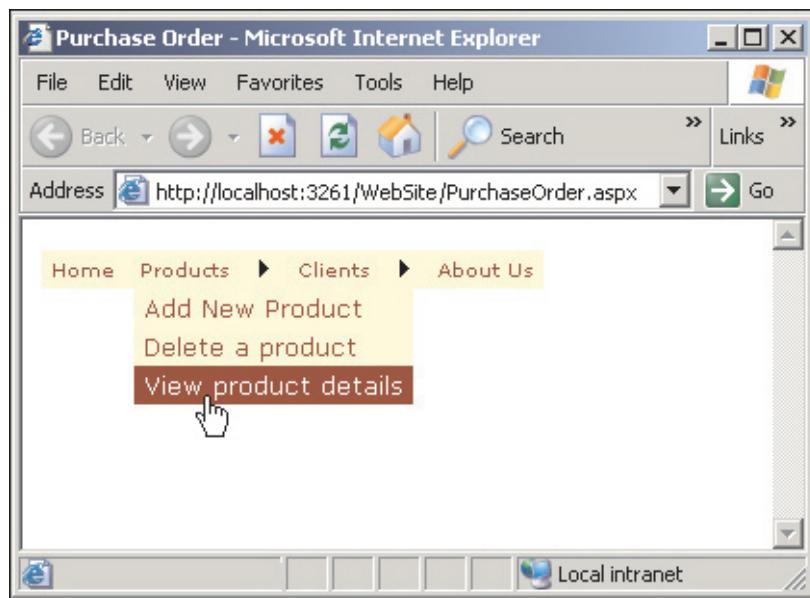


Figure 7.2: Menu Control

→ **SiteMapDataSource Control**

Menu items in a large Web site can be moved to an external XML file for ease of maintenance. By doing this, the menus can be maintained separately from the Web pages that use them.

An XML menu file can be linked to a **Menu** control and used in exactly the same way as a menu coded internally as part of the **Menu** control. The external menu is made available to the **Menu** control through **SiteMapDataSource** control. The **SiteMapDataSource** control looks for a special file named `web.sitemap`. This file contains the XML-encoded menu. The **Menu** control is linked to the **SiteMapDataSource** control, which in turn is linked to the `web.sitemap` file. Any changes made in the menu structure in the `web.sitemap` file, is automatically reflected in the **Menu** control.

Figure 7.3 demonstrates the concept.

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap>
    <siteMapNode url="Hello.aspx" title="Home">
        <siteMapNode url="Prod.aspx" title="Products">
            <siteMapNode url="BuyPrd.aspx" title="Buy"/>
            <siteMapNode url="SelPrd.aspx" title="Sell"/>
        </siteMapNode>
        <siteMapNode url="Clients.aspx" title="Clients"/>
        <siteMapNode url="AbtUs.aspx" title="About Us"/>
    </siteMapNode>
</siteMap>
```

Figure 7.3: SiteMapDataSource Control

7.1.2 Properties, Methods, and Events

The `Menu` class represents the Menu Web server control and belongs to the `System.Web.UI.WebControls` namespace. This class is used to create and configure menus in Web pages at runtime. The properties, methods, and events of this class are used to give static and dynamic menus, a customized appearance by using various images and styles.

The properties, methods, and events of the `Menu` class are described as follows:

→ Items Property

`Item` property is used to retrieve a `MenuItemCollection` object that contains all menu items present in the `Menu` control. This property is useful when adding or removing items to/from a menu.

The following code creates a menu at runtime and adds items to it.

Code Snippet:

```
Menu mnuCompany = new Menu();  
  
MenuItem mnuitemCareers = new MenuItem("Careers");  
  
mnuCompany.Items.Add(mnuitemCareers);
```

In this code, `mnuCompany` is an object of `Menu` class. A menu item named `mnuitemCareers` is created using `MenuItem` class. To get the existing menu item collection present within `mnuCompany`, the `Items` property is used.

→ Orientation Property

Orientation property is used to specify or retrieve the direction in which to display the Menu control. For example, you can make a menu horizontal or vertical using this option.

The following code demonstrates how to set the layout of the menu items in `Menu` control.

Code Snippet:

```
mnuCompany.Orientation=Orientation.Horizontal;
```

In this code, direction of menu `mnuCompany` is set to horizontal using the `Orientation` property.

→ PathSeparator Property

`PathSeparator` property is used to specify or retrieve the character used to restrict the path of a menu item.

The following code demonstrates how to specify the path separator for menu items.

Code Snippet:

```
mnuCompany.PathSeparator='>';
```

In this code, the path separator for menu items is set to '`>`', which will be the delimiter character separating the menu items.

→ Target Property

`Target` property is used to specify or retrieve the target window or frame in which the Web page content needs to be displayed.

The following code demonstrates how to set the target window in which the Web page needs to be displayed.

Code Snippet:

```
mnuCompany.Target="_blank";
```

In this code, `_blank` specifies the target window as a new window.

→ FindItem Method

`FindItem` method retrieves the menu item at the specified value path. The menu path is a string of delimited values that form the path from a root menu item to the current menu item.

The following code demonstrates how to use the `FindItem` property for searching a menu item.

Code Snippet:

```
MenuItem mnuitemFind = mnu.FindItem("Careers>HR");
if (mnuitemFind != null)
{
    Response.Write("Item is found.");
}
```

In this code, the `FindItem()` method is used to retrieve the menu item, HR, using the specified value path. If the menu item is found, the message `Item is found` is displayed on the Web page using `Response.Write()` method.

→ MenuItemClick Event

`MenuItemClick` event is triggered when a menu item in a `Menu` control is clicked.

The following code demonstrates how the `MenuItemClick` event is triggered when a menu item is clicked.

Code Snippet:

```
void mnuCompany_MenuItemClick(object sender, MenuEventArgs e)
{
    Response.Write("You clicked " + mnuCompany.SelectedItem.Value);
}
```

In this code, an object of `MenuEventArgs` class is passed to the event handler that allows accessing the properties of the menu item which raised the event. The name of the particular menu item is displayed using `Response.Write()` method when the menu item is clicked.

7.1.3 TreeView Control

A tree view can be considered as an expanding and contracting menu such as the one you see in the left pane of the Explorer window.

ASP.NET 2.0 provides a `TreeView` control to represent the links between pages in a tree structure. The links are represented in a hierarchical manner. The tree view structure is made of multiple levels of nodes, namely the root node, the leaf nodes, and one or more levels of intermediate nodes. These nodes are represented by the `TreeNode` object.

The `TreeNode` object has four UI elements. These are as follows:

→ **Expand/Collapse image**

Expand/Collapse image indicates whether the node is in its expanded or collapsed state. The plus (+) sign indicates that the node is collapsed and can be expanded whereas the minus (-) sign indicates that the node is expanded and can be collapsed.

→ **Checkbox**

Checkboxes are used to allow the user to select a particular node.

→ **Node Text**

Node text is the text that is displayed on the `TreeNode` object. These text can also be used as hyperlink for navigation.

→ **Node Image**

Node image is the image displayed near the node text.

Figure 7.4 depicts the TreeView control.

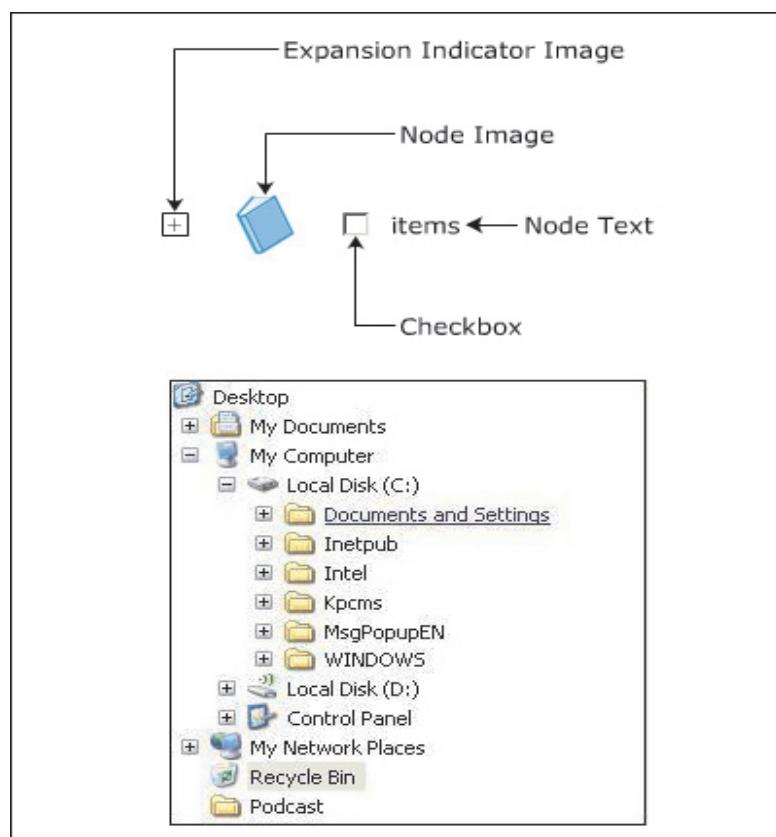


Figure 7.4: TreeView Control

7.1.4 SiteMapDataSource Control

The TreeView control can also bind to a SiteMapDataSource control. The SiteMapDataSource control looks for a special file named `web.sitemap` stored in the root Web directory.

This `web.sitemap` file provides input to the TreeView control to produce a site menu. The TreeView control is linked to the SiteMapDataSource control, which in turn is linked to the `web.sitemap` file. Any changes made to the hierarchical structure in the `web.sitemap` file are automatically reflected in the TreeView control.

7.1.5 Properties

The TreeView class belongs to `System.Web.UI.WebControls` namespace. This class represents the TreeView control that displays a hierarchical collection of labeled items. Each of these labeled items is represented by a tree node.

Figure 7.5 depicts some of the properties of the TreeView class.

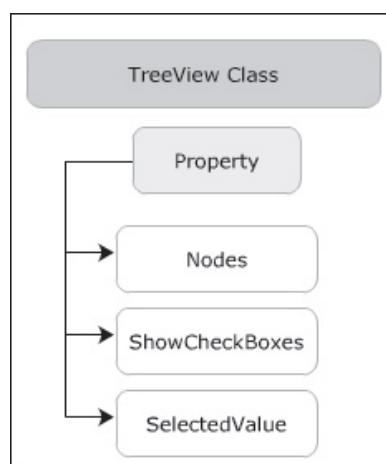


Figure 7.5: Properties of TreeView Class

The properties of TreeView class are as follows:

→ **Nodes**

`Nodes` property retrieves the collection of tree nodes that are allocated to the TreeView control. This property is useful when adding or removing nodes to/from a node collection.

→ **ShowCheckBoxes**

`ShowCheckBoxes` property specifies or retrieves a value that indicates which node types will display a checkbox.

→ **SelectedValue**

`SelectedValue` property returns the value of the node which is selected.

→ **NodeStyle**

`NodeStyle` property specifies or retrieves the default style of the nodes.

→ **ShowLines**

`ShowLines` property specifies or retrieves a value indicating whether lines are drawn between tree nodes in the tree view control.

→ **SelectedNode**

`SelectedNode` property specifies or retrieves the tree node that is currently selected.

The syntax for creating a `TreeView` control is:

Syntax:

```
<asp:TreeView>
  <Nodes>
    <asp:TreeNode>
    </asp:TreeNode>
  </Nodes>
</asp:TreeView>
```

where,

`<asp:TreeView> </asp:TreeView>`: ASP.NET declarative syntax used to create a `TreeView` control.

`<Nodes> </Nodes>`: Tags used to add a node to the tree view.

`<asp:TreeNode> </asp:TreeNode>`: ASP.NET declarative syntax used to create tree nodes.

→ **Node**

The following code demonstrates how to create a tree node and add it to an existing `TreeView` control.

Code Snippet:

```
TreeNode tnWild = new TreeNode("Wild");
TreeNode tnLion = new TreeNode("Lion");
tnWild.ChildNodes.Add(tnLion);
tvwAnimals.Nodes.Add(tnWild);
```

In this code, `tnWild` and `tnLion` are the objects of `TreeNode` class. The `Add()` method adds `tnWild` node to the `TreeView` control named `tvwAnimals`.

→ **ShowCheckboxes**

The following code demonstrates how to display a checkbox in a node.

Code Snippet:

```
tvwBank.ShowCheckboxes = TreeNodeTypes.Leaf;
```

In this code, `tvwBank` is configured to show leaf nodes with a checkbox displayed alongside each node to allow its selection. So if the tree view has three leaf nodes, each of the nodes will have a checkbox beside it.

→ **SelectedValue**

The following code demonstrates the use of `SelectedValue` property.

Code Snippet:

```
Response.Write("You selected: " + tvwBank.SelectedValue);
```

In this code, `SelectedValue` retrieves the value of the node `tvwBank` and displays it on the Web page.

→ **NodeStyle**

The following code demonstrates the use of `NodeStyle` property.

Code Snippet:

```
tvwBank.NodeStyle.BackColor = System.Drawing.Color.Yellow;
```

In this code, the background color of the tree node is set to yellow using the `NodeStyle` property.

→ **ShowLines**

The following code demonstrates the use of the `ShowLines` property.

Code Snippet:

```
tvwAnimals.ShowLines = true;
```

In this code, the value of `ShowLines` property is set to `true`. Setting the value to `true` draws lines between the tree nodes.

→ **SelectedNode**

The following code demonstrates the use `SelectedNode` property.

Code Snippet:

```
Response.Write("You clicked: " + tvwAnimals.SelectedNode.Value);
```

In this code, the value of the selected node from the TreeView control named tvwAnimals is retrieved using the SelectedNode property and displayed using Response.Write() method.

7.1.6 Methods and Events

The important methods and events of the TreeView control are as follows:

- FindNode Method
- SelectedNodeChanged Event
- TreeNodeCollapsed Event
- TreeNodeExpanded Event
- TreeNodeCheckChanged Event

Figure 7.6 depicts some of the methods and events of the TreeView class.

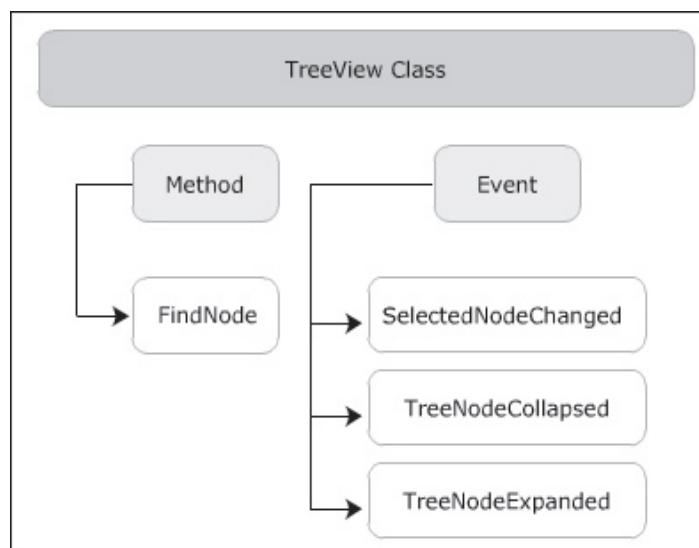


Figure 7.6: Methods and Events of TreeView Class

→ **FindNode Method**

FindNode () method is used to retrieve a node from the TreeView control at the specified value path.

The following code demonstrates the use of `FindNode()` method.

Code Snippet:

```
TreeNode findnode = tvwAnimals.FindNode ("Tame>Dog");
if (findnode != null)
{
    Response.Write ("Item found.");
}
```

In this code, the `FindNode()` method is used to retrieve a node from the `TreeView` control. If the node is found, the message `Item found` is displayed on the Web page using `Response.Write()` method.

→ **SelectNodeChanged Event**

The `SelectedNodeChanged` event occurs when a node is selected in the `TreeView` control.

The following code demonstrates the use of `SelectedNodeChanged` event.

Code Snippet:

```
void tvwAnimals_SelectedNodeChanged (object sender, EventArgs e)
{
    Response.Write ("You clicked: " + tvwAnimals.SelectedNode.Value);
}
```

In this code, the `SelectedNodeChanged` event occurs when a node is selected. Within the event handler, the value of the selected node is displayed.

→ **TreeNodeCollapsed Event**

The `TreeNodeCollapsed` event occurs when a node is collapsed.

The following code demonstrates the use of `TreeNodeCollapsed` event.

Code Snippet:

```
void tvwBank_TreeNodeCollapsed (object sender, TreeNodeEventArgs e)
{
    Response.Write ("Tree node has been collapsed.");
}
```

In this code, `TreeNodeCollapsed` event is triggered and `Tree node has been collapsed` message is displayed when the tree node is collapsed.

→ **TreeNodeExpanded Event**

The **TreeNodeExpanded** event occurs when a node is expanded.

The following code demonstrates the use of **TreeNodeExpanded** event.

Code Snippet:

```
void twvBank_TreeNodeExpanded(object sender, TreeNodeEventArgs e)
{
    Response.Write("Tree node has been expanded.");
}
```

In this code, **TreeNodeExpanded** event is triggered and **Tree node has been expanded** message is displayed when the tree node is expanded.

→ **TreeNodeCheckChanged Event**

The **TreeNodeCheckChanged** event occurs if the state of the checkbox is changed between post backs to the server.

The following code demonstrates the use of **TreeNodeCheckChanged** event.

Code Snippet:

```
void twvBank_TreeNodeCheckChanged(object sender, TreeNodeEventArgs e)
{
    Response.Write("You have changed the checked status of tree node.");
}
```

In this code, **TreeNodeCheckChanged** event is triggered and the message **You have changed the checked status of tree node** is displayed when the checked status of the tree node is changed since the last post back.

7.1.7 SiteMapPath Control

The logical structure of a Web application is stored in an XML file and is called a site map. Site maps define the layout and the relationship between pages in a Web application.

ASP.NET 2.0 provides the **SiteMapPath** server control to work with site map files and create a breadcrumb navigation mechanism. A breadcrumb is a linear path from the home page to the user's current location.

Consider a Web site with several Web pages. When browsing through the site, chances are that users will lose track of how they reached a particular page. To allow users to recollect the path they have taken to reach a particular page, breadcrumb navigation is used. Breadcrumb links allow backtracking through the pages. They show the user the current page location and display the path from the home page to the current page. For example, **Home → Automobiles → Cars → Ford → Mustang** can be a breadcrumb.

Figure 7.7 demonstrates the concept.

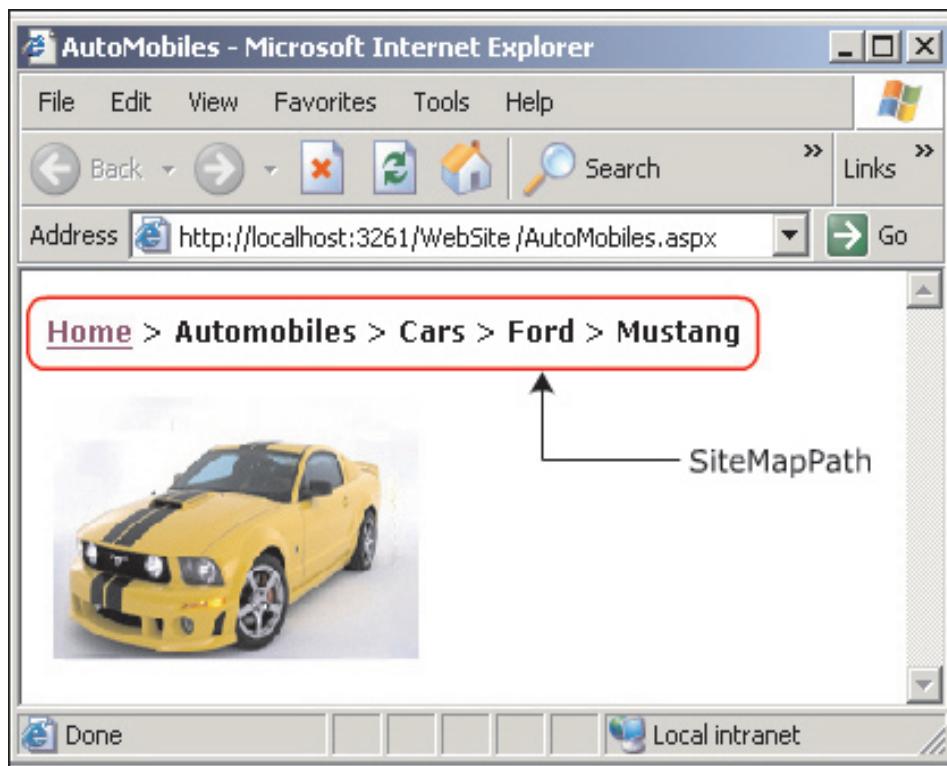


Figure 7.7: SiteMapPath Control

7.1.8 Properties, Methods, and Events

SiteMapPath class belongs to `System.Web.UI.WebControls` namespace. SiteMapPath class displays text or image as hyperlinks. This enables the users to navigate more easily through a Web site.

The properties and events of SiteMapPath class are as follows:

- ➔ PathSeparator Property
- ➔ CurrentNodeStyle Property
- ➔ PathDirection Property
- ➔ RootNodeStyle Property
- ➔ ItemCreated Event

Figure 7.8 depicts some of the methods and Properties of the SiteMapPath control..

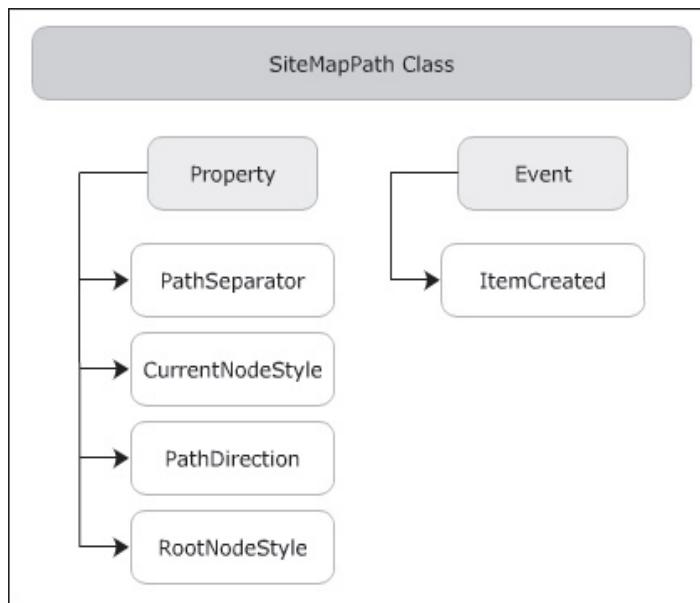


Figure 7.8: Methods and Properties of SiteMapPath Control

These are described as follows:

→ **PathSeparator Property**

The `PathSeparator` property specifies or retrieves the string that delimits `SiteMapPath` nodes in the rendered navigation path.

The following code demonstrates the use of `PathSeparator` property.

Code Snippet:

```

SiteMapPath sitemphome = new SiteMapPath();
...
sitemphome.PathSeparator = "::";
  
```

In this code, an object `sitemphome` of `SiteMapPath` class is created. The `PathSeparator` property is set to `::`. This represents a delimiter for the nodes in the navigation path.

→ **CurrentNodeStyle Property**

The `CurrentNodeStyle` property retrieves the style used for the display text for the current node.

The following code demonstrates the use of `CurrentNodeStyle` Property.

Code Snippet:

```

sitemphome.CurrentNodeStyle.ForeColor = System.Drawing.Color.AliceBlue;
  
```

In this code, the foreground color of the SiteMapPath control named sitempHome is set to AliceBlue using the ForeColor property of the CurrentNodeStyle property.

→ **PathDirection Property**

The PathDirection property specifies or retrieves the order that the navigation path nodes are displayed in.

The following code demonstrates the use of PathDirection property.

Code Snippet:

```
sitempHome.PathDirection = PathDirection.RootToCurrent;
```

In this code, the PathDirection property of SiteMapPath control is set to RootToCurrent. This value displays the order in which the navigation path is rendered.

→ **RootNodeStyle Property**

The RootNodeStyle property retrieves the style of the display text in root node.

The following code demonstrates the use of RootNodeStyle property.

Code Snippet:

```
sitempHome.RootNodeStyle.ForeColor = System.Drawing.Color.Aqua;
```

In this code, the foreground color of the root node of the navigation path is set to Aqua using the ForeColor property of the RootNodeStyle property.

→ **ItemCreated Event**

The ItemCreated event occurs when a new SiteMapNodeItem is created by the SiteMapPath control.

The following code demonstrates the use of ItemCreated property.

Code Snippet:

```
void sitempHome_ItemCreated(object sender, SiteMapNodeEventArgs e)
{
    Response.Write("Item created: " + e.Item);
}
```

In this code, the ItemCreated event fires when a node is created by the SiteMapPath control and the node item is displayed using the Response.Write() method.

Knowledge Check 1

1. Which of the following statements about Menu Web server control, SiteMapPath Web server control and TreeView control are true?

(A)	Nodes property retrieves the collection of tree nodes that are allocated to the TreeView control.
(B)	The site navigation system of ASP.NET 2.0 allows defining all links in a single location, which is usually a .css file.
(C)	Target property is used to specify or retrieve the target window or frame in which the Web page content needs to be displayed.
(D)	FindNode() method is used to retrieve a node from the TreeView control at the specified value path.
(E)	The PathSeparator() method specifies or retrieves the string that delimits SiteMapPath nodes in the rendered navigation path.

(A)	A, B, D	(C)	B, E
(B)	A, C	(D)	A, C, D

2. Can you match the properties and methods of Menu Web server control, SiteMapPath Web server control, and TreeView control with their corresponding descriptions

Description		Property and Method	
(A)	Specifies or retrieves the delimiter string that is used by the tree node path.	(1)	FindItem
(B)	Specifies or retrieves the order in which the navigation path nodes are delivered.	(2)	Target
(C)	Retrieves the menu item at the specified value path.	(3)	PathSeparator
(D)	Retrieve a node from the TreeView control at the specified value path.	(4)	PathDirection
(E)	Specifies or retrieves the target window where the Web page content needs to be displayed.	(5)	FindNode

(A)	(A)-(5), (B)-(3), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(4), (C)-(1), (D)-(5), (E)-(2)	(D)	(A)-(4), (B)-(3), (C)-(5), (D)-(1), (E)-(2)

7.2 ImageMap Control and HotSpot Classes

In this second lesson, **ImageMap Control and HotSpot Classes**, you will learn to:

- Describe the ImageMap control and HotSpot class.
- Explain how to use the different types of HotSpot classes.

7.2.1 ImageMap control and HotSpot Class

Certain areas on an image on a Web page can cause the mouse cursor to become clickable. When you click the mouse after moving the cursor to such an area, you are directed to another page. Such clickable regions on images are referred to as hot spots.

ASP.NET 2.0 provides the `ImageMap` control to create images containing clickable hot spot regions. When a user clicks a hot spot region, a sub-program can be called or the user can be directed to another page.

`ImageMap` control works like a combination of `Image` and `ImageButton` controls.

Figure 7.9 demonstrates the concept.

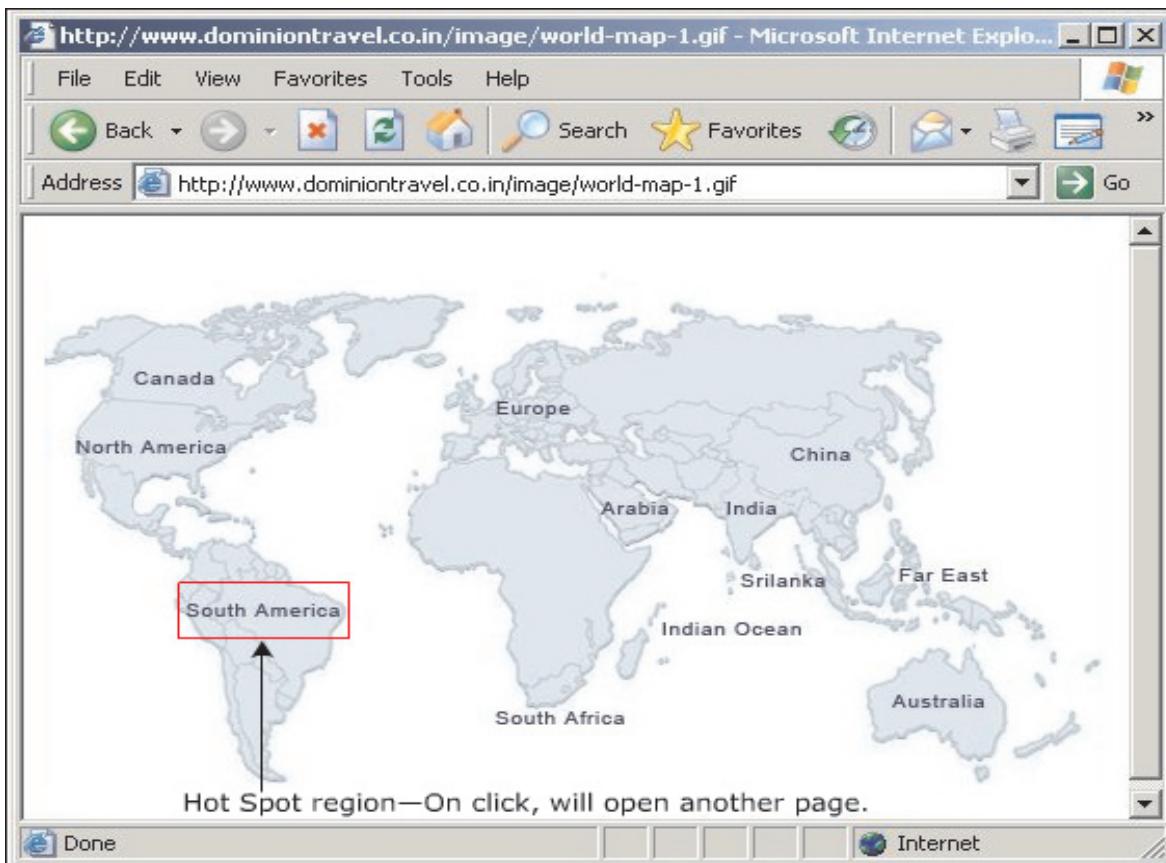


Figure 7.9: ImageMap Control

7.2.2 Properties and Events of ImageMap Class

`ImageMap` class represents the `ImageMap` control and belongs to `System.Web.UI.WebControls` namespace.

The properties and events of the `ImageMap` class that enable you to define the hot spot regions and navigate to another URL.

Figure 7.10 demonstrates the concept.

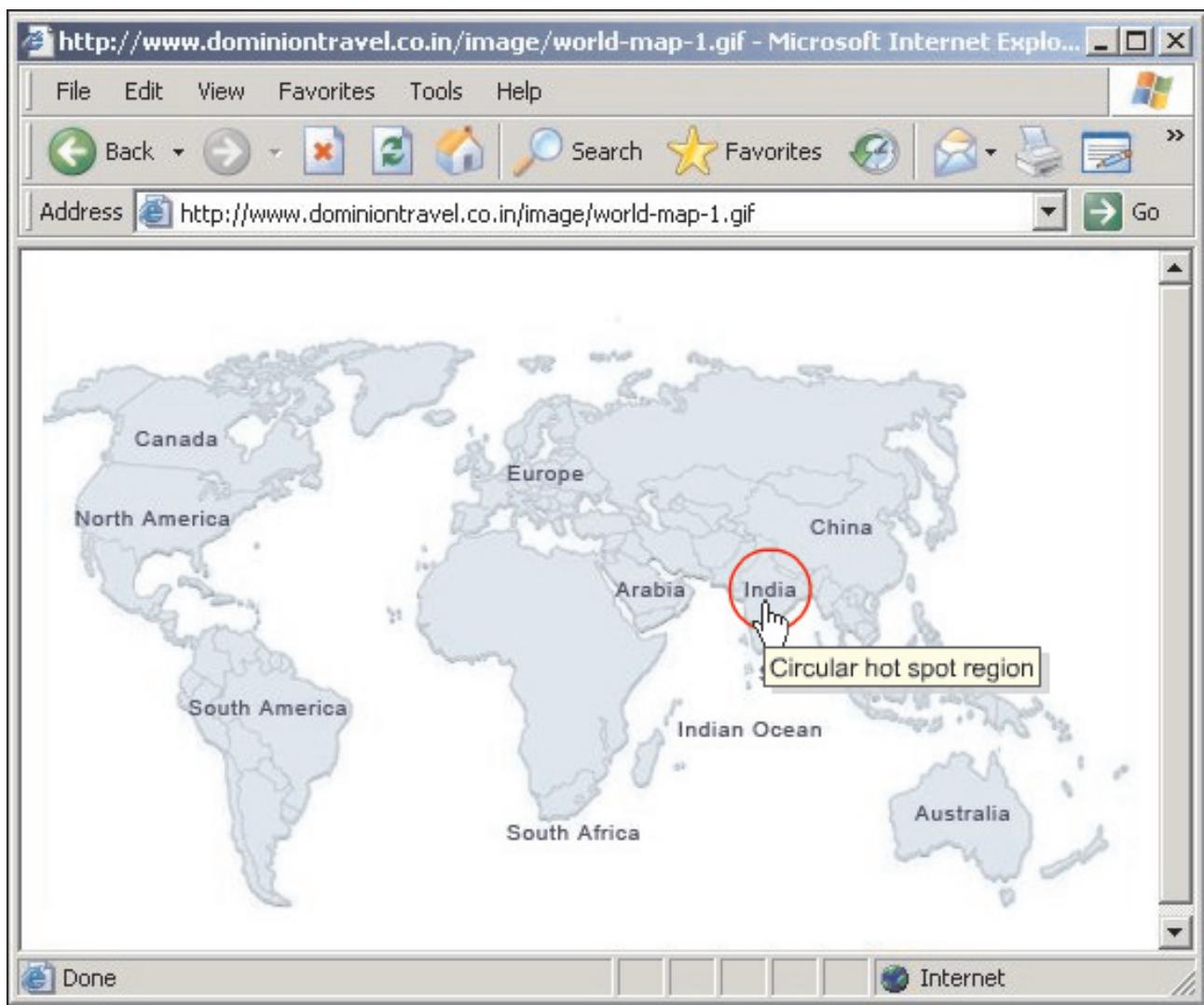


Figure 7.10: HotSpot Property of ImageMap

The properties and event of the `ImageMap` class are as follows:

→ **HotSpotMode Property**

The `HotSpotMode` property specifies or retrieves the behavior for the `HotSpot` object.

→ **HotSpots Property**

The `HotSpots` property retrieves a collection of `HotSpot` objects. These `HotSpot` objects represent the defined hot spot regions in an `ImageMap` control.

→ **ImageUrl Property**

The `ImageUrl` property specifies or retrieves the location of an image to display in the `Image` control.

→ **AlternateText Property**

The `AlternateText` property specifies or retrieves the alternate text to be displayed when the image is unavailable. If the browser supports tool tip feature, then the alternate text will be displayed as `ToolTip`.

→ **Click Event**

The `Click` event occurs when a `HotSpot` object in an `ImageMap` control is clicked.

Syntax:

The syntax for creating an `ImageMap` control is:

```
<asp:ImageMap>  
...  
</asp:ImageMap>
```

This is the skeleton syntax used to create `ImageMap` controls. However, as it is very cumbersome to create the `ImageMap` controls using the complete syntax at runtime, they are created at design time itself to avoid complications.

→ **HotSpotMode Property**

The following code demonstrates how to set the behavior of the `HotSpot` object.

Code Snippet:

```
imgmpWorld.HotSpotMode = HotSpotMode.PostBack;
```

In this code, `HotSpotMode` property value is specified as `PostBack`. When the hot spot region is clicked, `HotSpot` objects generate a postback to the server.

→ **HotSpots Property**

The following code demonstrates the use of the `HotSpots` property.

Code Snippet:

```
PolygonHotSpot phspotUSA = new PolygonHotSpot();
phspotUSA.Coordinates = "28,85,33,15,40,22,40,40,122,200";
imgmpWorld.HotSpots.Add(phspotUSA);
```

In this code, an object, `phspotUSA`, is created of the `PolygonHotSpot` class. The coordinates for the vertices of the polygon are specified using the `Coordinates` property. The `HotSpots` property retrieves the collection of hot spots and adds a polygonal hot spot object `phspotUSA` to the collection.

→ **ImageUrl Property**

The following code demonstrates how to specify the image location.

Code Snippet:

```
imgmpWorld.ImageUrl = "images/WorldMap.jpg";
```

In this code, `images/WorldMap.jpg` is the location of the image to be displayed.

→ **AlternateText Property**

The following code demonstrates how to specify the alternate text to be displayed.

Code Snippet:

```
imgmapWorld.AlternateText = "This is an image.";
```

In this code, `This is an image` is the text that will be displayed when the image is unavailable.

→ **Click Event**

The following code demonstrates the use of `Click` event.

Code Snippet:

```
void imgmpWorld_Click(object sender, ImageMapEventArgs e)
{
    Response.Write("You clicked on the HotSpot.");
}
```

In this code, the `Click` event occurs when the `HotSpot` object is clicked and the `Response.Write()` method is used to display an appropriate message.

→ Properties and Methods of HotSpot Class

HotSpot class belongs to `System.Web.UI.WebControls` namespace. This class is used to create hot spot regions on an image. Hot spot regions can be either circular, rectangular, or polygonal.

Some important properties and methods of `HotSpot` class are described as follows:

- `AccessKey` Property
- `PostBackValue` Property
- `GetCoordinates` Method

Figure 7.11 depicts some of the properties of the `HotSpot` class.

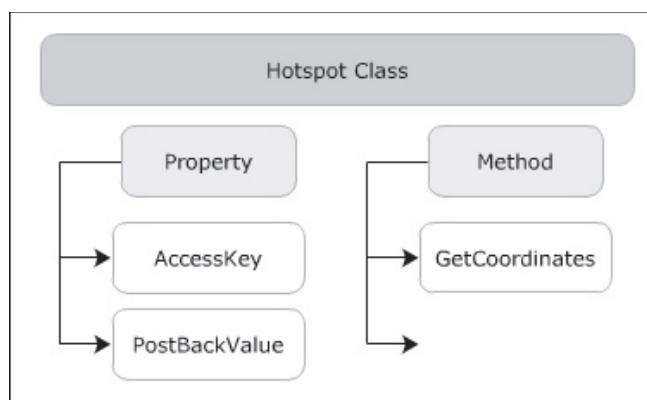


Figure 7.11: Properties of Hotspot Class

These are described as follows:

→ AccessKey Property

The `AccessKey` property specifies or retrieves the access key. This access key allows quick navigation to the hot spot region.

The following code demonstrates how to specify the access key.

Code Snippet:

```
phspotItem.AccessKey = "H";
```

In this code, the access key value is specified as `H`. This allows quick navigation to the region by pressing ALT key and the access key, which in this case is `H`.

The `src` attribute is mandatory, t

→ PostBackValue Property

The `PostBackValue` property specifies or retrieves the name of the `HotSpot` object to pass in the event data when the hot spot is clicked.

The following code demonstrates how to specify the PostBackValue property.

Code Snippet:

```
phspotItem.PostBackValue = "ItemName";
```

In this code, ItemName is the name of the HotSpot object and it is passed in the event data when the hot spot is clicked.

→ **GetCoordinates Method**

The GetCoordinates() method retrieves a string that represents the coordinates of hot spot region.

The following code demonstrates how to retrieve the coordinates of the hot spot region.

Code Snippet:

```
string coordinates = phspotItem.GetCoordinates();  
Response.Write("Co-ordinates are : " + coordinates + ".");
```

In this code, coordinates of phspotItem object are retrieved into a string variable. Response. Write() method displays the coordinates on the Web page.

7.2.3 CircleHotspot Class

The CircleHotSpot class is used to define a circular hot spot region on the ImageMap control. To define a circular hot spot region on the ImageMap control, you have to provide the x and y coordinates of the pixel representing the center of the circle as well as need to provide the length of the radius. These values can be provided using the properties of the CircleHotSpot class.

Figure 7.12 demonstrates the concept.

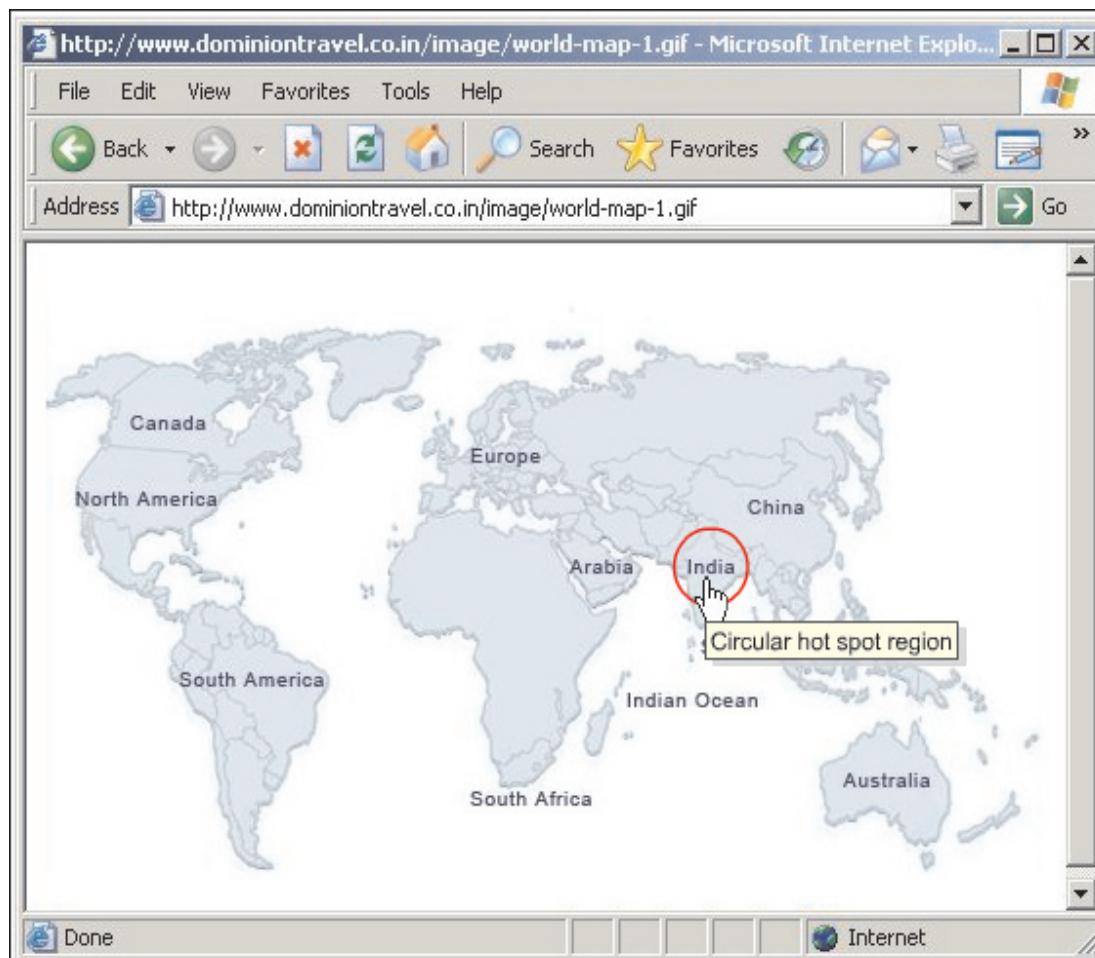


Figure 7.12: Circle HotSpot Class

The important properties of CircleHotSpot class are as follows:

→ **NavigateUrl**

`NavigateUrl` property specifies or retrieves the URL to which the user should be redirected when the circular hot spot region is clicked.

The following code demonstrates how to specify a URL for navigation.

Code Snippet:

```
chspotProduct.NavigateUrl = "http://www.productexample.com";
```

In this code, `http://www.productexample.com` is specified as the URL to which the control will be traversed on click of `HotSpot` object.

→ **Radius**

`Radius` property specifies or retrieves the distance from the center of the circular hot spot region

to the edge of the region.

The following code demonstrates how specify the radius of the circular region.

Code Snippet:

```
chspotProduct.Radius = 50;
```

In this code, value from the center to the edge of the circle is set to 50.

→ **x**

x property specifies or retrieves the x coordinate of the center of the circular region.

The following code demonstrates how specify the x-coordinate of the center of the circular region.

Code Snippet:

```
chspotProduct.X = 120;
```

In this code, value of x-coordinate is specified as 120.

→ **y**

y property specifies or retrieves the y coordinate of the center of the circular region.

The following code demonstrates how specify the y-coordinate of the center of the circular region.

Code Snippet:

```
chspotProduct.Y = 75;
```

In this code, value of y-coordinate is specified as 75.

7.2.4 PolygonHotspot Class

The `PolygonHotSpot` class is used to define a polygonal hot spot region on the `ImageMap` control. To define a polygonal hot spot region on the `ImageMap` control, you have to provide the x and y coordinates of the pixels representing corners of the polygon. The `PolygonHotSpot` class is useful when you need to define irregularly-shaped hot spot regions such as areas on a country's map.

Figure 7.13 demonstrates the concept.

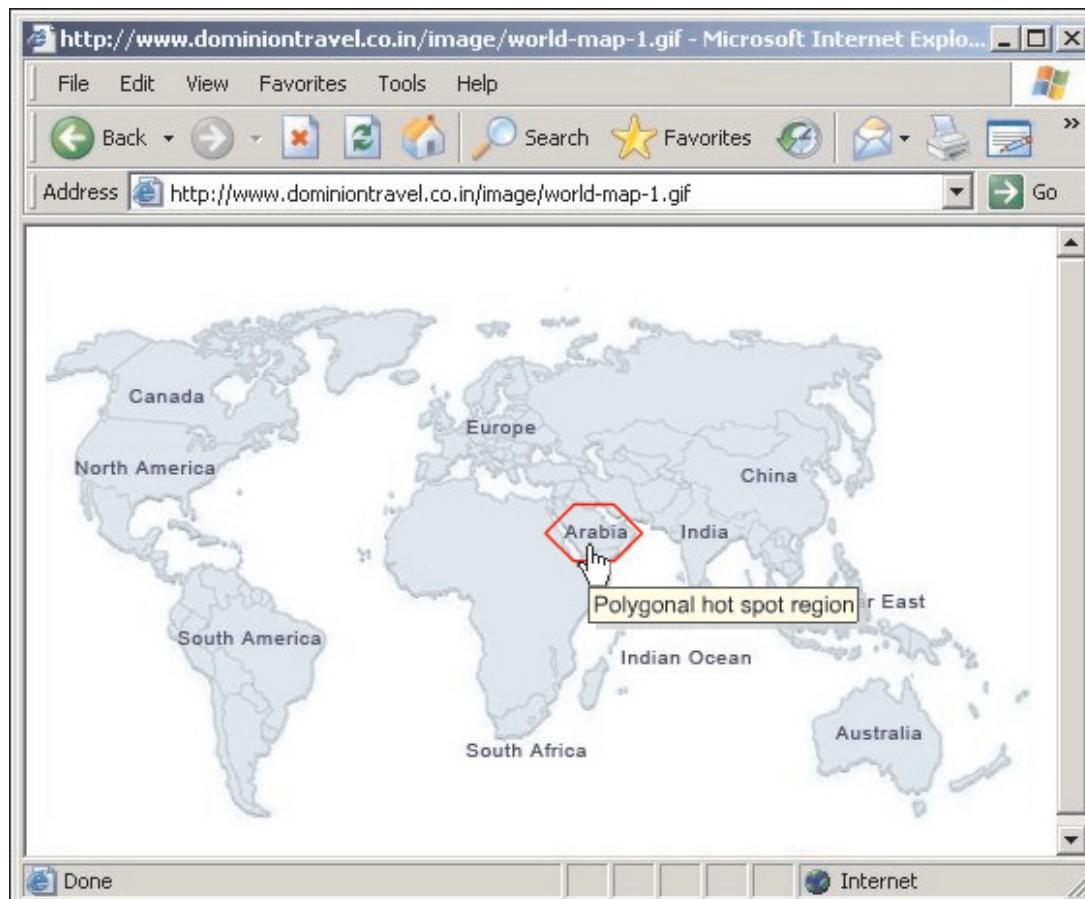


Figure 7.13: PolygonHotspot Class

The important properties of `PolygonHotSpot` class are as follows:

→ **Coordinates** Property

`Coordinates` property specifies a string of x and y coordinates of pixels representing the vertexes of the polygonal hot spot region.

The following code demonstrates how to specify the coordinates for a `PolygonHotSpot` object.

Code Snippet:

```
phspotItem.Coordinates = "28,85,33,15,40,22,40,40,122,200";
```

In this code, 28, 85, 33, 15, 40, 22, 40, 40, 122, and 200 are the coordinates of the polygonal region. 28, 85, 33, 15, 40, 22, 40, 40, 122, and 200 represent the x1, y1, x2, y2, x3, y3, x4, y4, x5, and y5 coordinates respectively.

→ Target Property

Target property specifies or retrieves the target window or frame in which the linked Web page is to be displayed.

The following code demonstrates how to set the target window.

Code Snippet:

```
phspotItem.Target = "_blank";
```

In this code, `_blank` specifies the target as a new window.

7.2.5 RectangleHotspot Class

The `RectangleHotSpot` class is used to define a rectangular hot spot region on the `ImageMap` control. To define a rectangular hot spot region on the `ImageMap` control, you have to provide the x and y coordinates of the pixels representing the top left and bottom right corners of the rectangle.

Figure 7.14 demonstrates the concept.

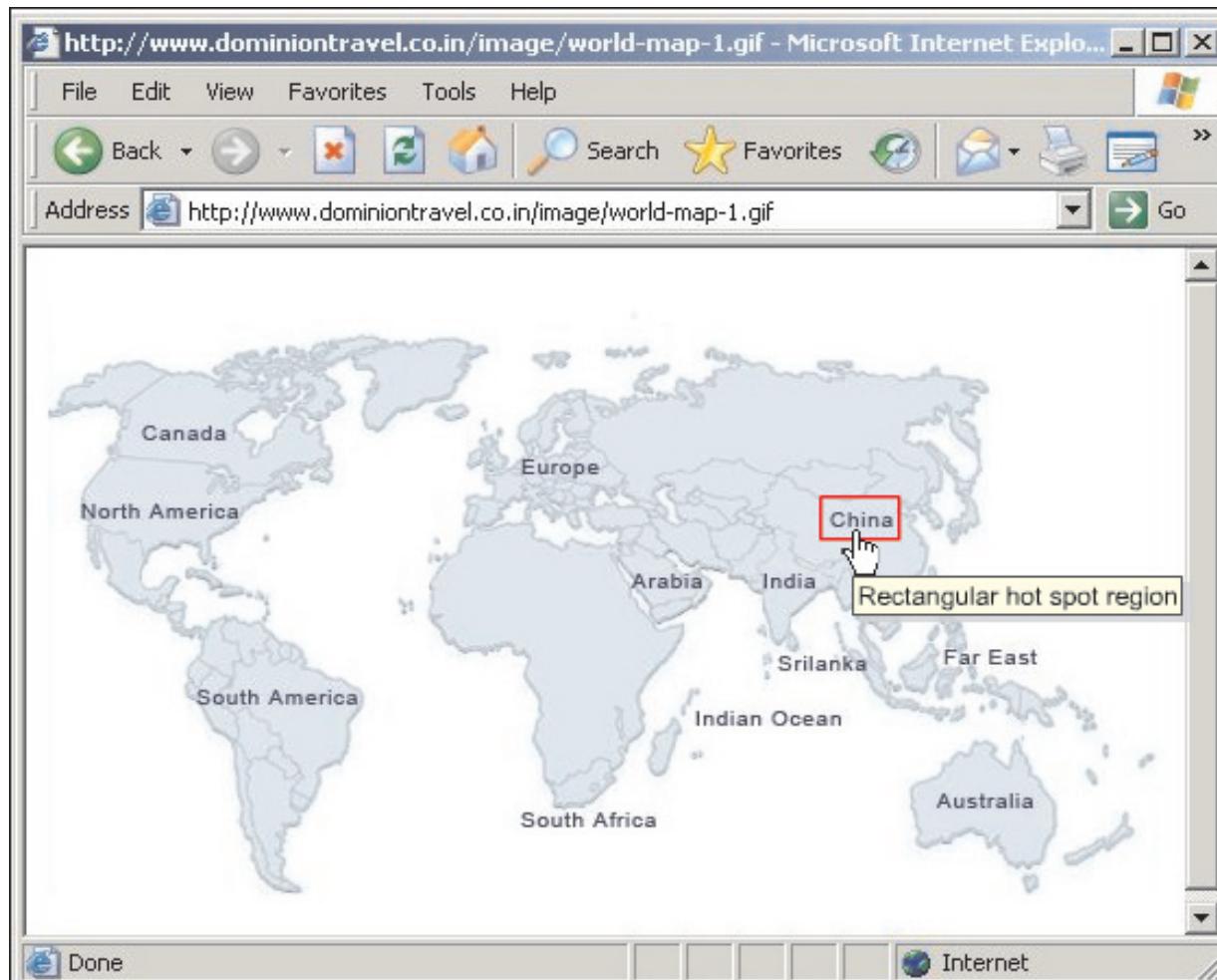


Figure 7.14: Rectangular Hotspot Class

The important properties of RectangleHotSpot class are as follows:

→ **Bottom**

Bottom property specifies or retrieves the y-coordinate of the bottom side of the rectangular region.

The following code demonstrates how to specify the y-coordinate of the bottom side of the rectangular region.

Code Snippet:

```
rhspotBook.Bottom = 200;
```

In this code, the bottom side of the rectangular region is specified as 200.

→ **Left**

Left property specifies or retrieves the x-coordinate of the left side of the rectangular region.

The following demonstrates how to specify the x-coordinate of the left side of the rectangular region.

Code Snippet:

```
rhspotBook.Left = 10;
```

In this code, the left side of the rectangular region is specified as 10.

→ **Right**

Right property specifies or retrieves the x-coordinate of the right side of the rectangular region.

The following demonstrates how to specify the x-coordinate of the right side of the rectangular region.

Code Snippet:

```
rhspotBook.Right = 100;
```

In this code, the right side of the rectangular region is specified as 100.

→ **Top**

Top property specifies or retrieves the y-coordinate of the top side of the rectangular region.

The following demonstrates how to specify the y-coordinate of the top side of the rectangular region.

Code Snippet:

```
rhspotBook.Top = 10;
```

In this code, the top side of the rectangular region is specified as 10.

Knowledge Check 2

1. Which of the following statements about `ImageMap` control and `HotSpot` class are true?

(A)	<code>ImageMap</code> control works exactly as <code>Image</code> control.		
(B)	<code>ImageMap</code> class belongs to <code>System.Web.UI.WebControls</code> namespace.		
(C)	<code>The RectangleHotSpot</code> class is used to define a rectangular hot spot region.		
(D)	<code>HotSpot</code> class belongs to <code>System.Web.Forms</code> namespace.		
(E)	<code>The AccessKey()</code> method specifies or retrieves the access key.		

(A)	A, B, D	(C)	B, E
(B)	B, C	(D)	A, C, D

2. Which one of the following codes creates a `PolygonHotSpot` region in an image map on a Web page and opens a new window when the region is clicked?

(A)	<pre>PolygonHotSpot phspotItem = new PolygonHotSpot(); phspotItem.AlternateText = "Item"; phspotItem.HotSpotMode = HotSpotMode.Navigate; phspotItem.Target = "_blank"; phspotItem.Coordinates = "28,85,33,15,40,22,40,40,122,200"; imgmapWorld.HotSpots.Add(phspotItem); frmCompany.Controls.Add(imgmapWorld);</pre>
(B)	<pre>PolygonHotSpot phspotItem = new PolygonHotSpot(); phspotItem.AlternateText = "Item"; phspotItem.HotSpotMode = HotSpotMode.PostBack; phspotItem.Target = "_blank"; phspotItem.Coordinates = "28,85,33,15,40,22,40,40,122,200"; imgmapWorld.HotSpots.Add(phspotItem); frmCompany.Controls.Add(imgmapWorld);</pre>

(C)	<pre>PolygonHotSpot phspotItem = new PolygonHotSpot(); phspotItem.AlternateText = "Item"; phspotItem.HotSpotMode = HotSpotMode.Inactive; phspotItem.Target = "_blank"; phspotItem.Coordinates = "28,85,33,15,40,22,40,40,122,200"; imgmapWorld.HotSpots.Add(phspotItem); frmCompany.Controls.Add(imgmapWorld);</pre>
(D)	<pre>PolygonHotSpot phspotItem = new PolygonHotSpot(); phspotItem.AlternateText = "Item"; phspotItem.HotSpotMode = HotSpotMode.NotSet; phspotItem.Target = "_blank"; phspotItem.Coordinates = "28,85,33,15,40,22,40,40,122,200"; imgmapWorld.HotSpots.Add(phspotItem); frmCompany.Controls.Add(imgmapWorld);</pre>

(A)	A	(C)	B
(B)	C	(D)	D

7.3 FileUpload Control

In this last lesson, **FileUpload Control**, you will learn to:

- Describe the FileUpload control.
- Explain how to use View control.
- Explain how to use MultiView control.

7.3.1 FileUpload Control

FileUpload control is provided by ASP.NET for locating and uploading files. For example, when sending an e-mail, if you want to attach a file to your mail, you have to browse to the specific file and then, attach it to the mail. Another example could be that you may want the user to browse for an image on his local folder and then, have it displayed on the Web page.

The FileUpload control creates a text box along with a browse button for locating files on the user's local folders.

The file types that can be uploaded as well as the maximum allowable size can be defined in the code. For example, sometimes you are only allowed to upload an image file in the jpeg format whose size cannot exceed 500 kilobytes.

The full path to the directory in which to save the uploaded file should be specified using SaveAs() method.

Figure 7.15 demonstrates the concept.

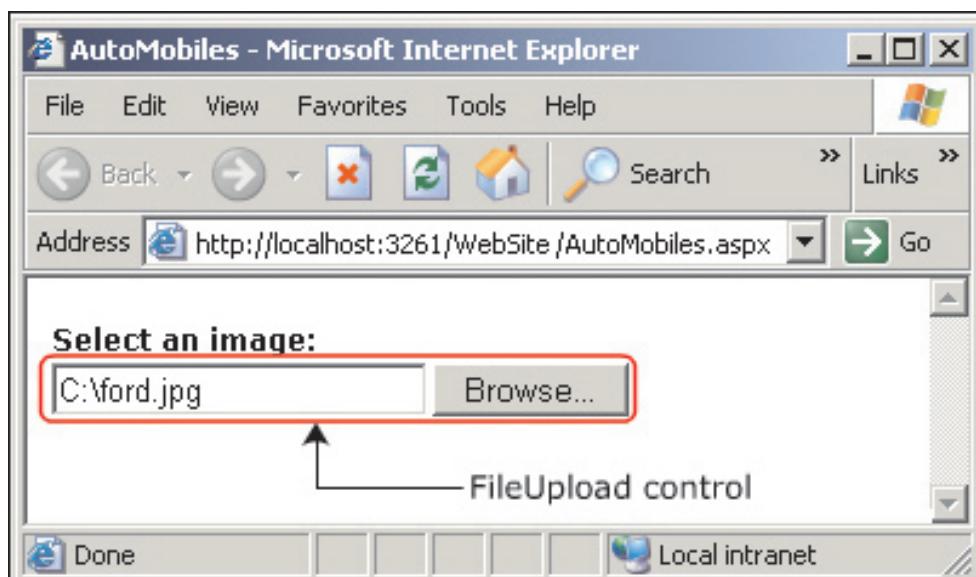


Figure 7.15: FileUpload Control

7.3.2 Properties and Methods of FileUpload Class

FileUpload class belongs to System.Web.UI.WebControls namespace.

Some of the important properties and methods of FileUpload class are as follows:

- HasFile Property
- FileName Property
- PostedFile Property
- SaveAs Method

Figure 7.16 depicts some of the properties and the method of the FileUpload class.

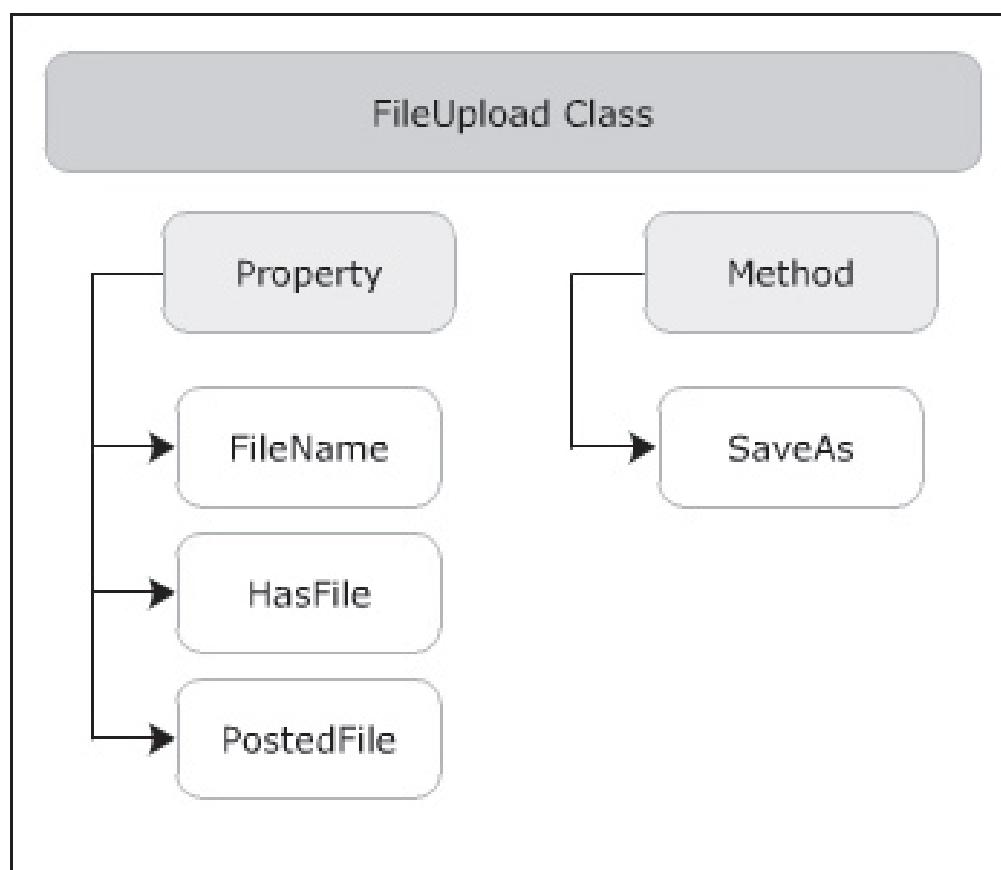


Figure 7.16: Properties and Method of FileUpload Class

These are described as follows:

→ **HasFile Property**

HasFile property retrieves a value indicating whether or not a file has been selected and appears in the input box.

The following code demonstrates how to use the HasFile property to check whether the specified file to be uploaded exists or not.

Code Snippet:

```
if (flupImage.HasFile)
{
    Response.Write("File found.");
}
```

In this code, the HasFile property checks whether the file which needs to be uploaded exists or not. If the file exists, then the message File found is displayed using Response.Write() method.

→ **FileName Property**

FileName property retrieves the name of a file selected for uploading.

The following demonstrates how to retrieve the name of the file.

Code Snippet:

```
Response.Write("File name is: " + flupImage.FileName);
```

In this code, the name of the selected file to be uploaded is retrieved using FileName property.

→ **PostedFile Property**

The PostedFile property is a reference to the file chosen for uploading. The ContentLength property of the file object returns the size of the file in bytes.

The following code demonstrates how to use the PostedFile property together with the Length property to retrieve the length of the file.

Code Snippet:

```
Response.Write("FileSize: " + flupImage.PostedFile.ContentLength);
```

In this code, ContentLength property of the PostedFile property is used to retrieve the length of the file in bytes.

→ SaveAs Method

SaveAs method is used to save the contents of a file that is uploaded. The uploaded file is saved to a specified path on the Web server.

The following code demonstrates how to use the `SaveAs ()` method to save the content of the uploaded file.

Code Snippet:

```
flupImage.SaveAs(@"c:\temp\");
```

In this code, the `SaveAs ()` method writes the uploaded file to the specified server directory.

7.3.3 View Control

The content within Web pages is sometimes too big to display completely on the screen and hence, have to be split into chunks. These chunks can then, be displayed one at a time. For example, when you are filling an online survey form, the questions may be segregated into groups and these groups are displayed one after another.

ASP.NET 2.0 provides the `View` control to manage such chunks of data. The `View` control can be referred to as a container of controls as this control can contain multiple other controls. The chunks of segregated data can be displayed one at a time using the `View` control.

Figure 7.17 demonstrates the concept.

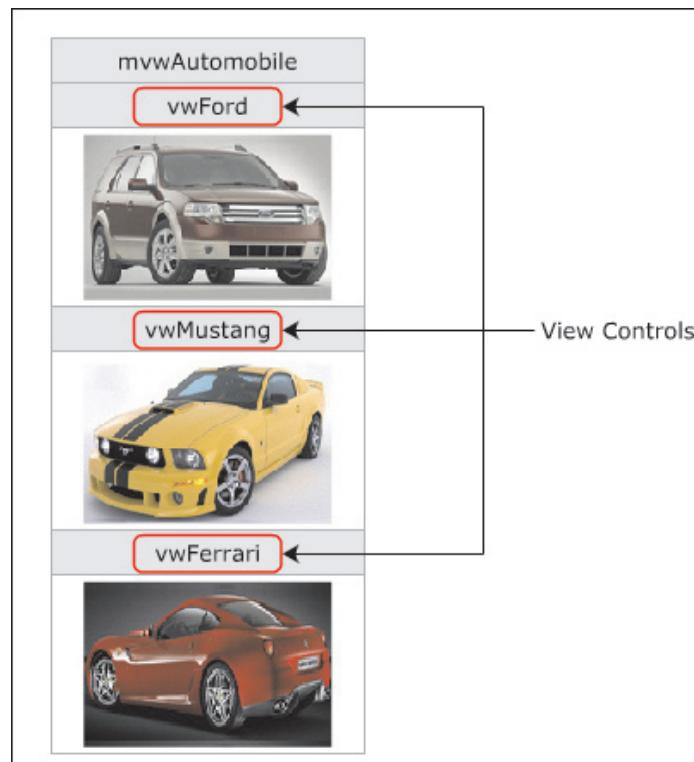


Figure 7.17: View Control

The following syntax is used to create View control at runtime.

Syntax:

```
<asp:View>  
...  
</asp:View>
```

7.3.4 MultiView Control

The View control can be referred to as a container of controls whereas the MultiView control can be referred to as a container of View controls. In the MultiView control, each View control forms the visible portion of the page to be displayed.

The MultiView control encloses one or more View controls. These views are indexed in their coded order of appearance.

Figure 7.18 demonstrates the concept.

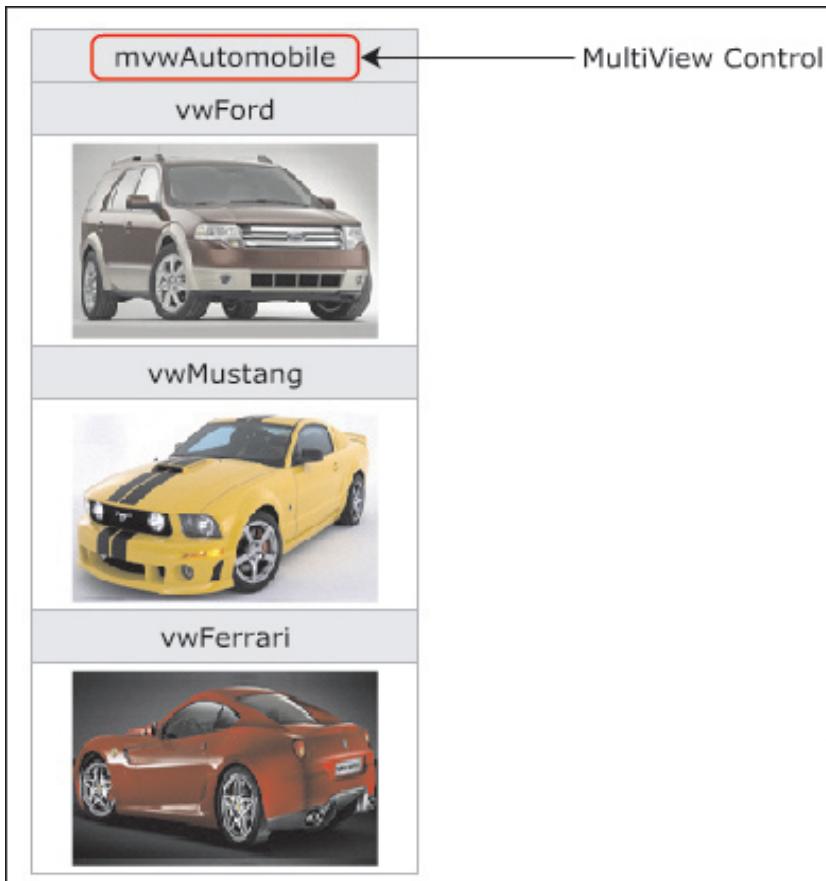


Figure 7.18: MultiView Control

The following is the syntax for creating a `Multiview` control at runtime.

Syntax:

```
<asp:multiview>
  ...
</asp:multiview>
```

7.3.5 Properties, Methods, and Events

`MultiView` class belongs to `System.Web.UI.WebControls`. `Multiview` class acts as a control that acts as a container for a group of View controls.

The properties, events, and methods of `Multiview` class are as follows:

- ➔ ActiveViewIndex Property
- ➔ Views Property
- ➔ SetActiveView Method
- ➔ ActiveViewChanged Event

Figure 7.19 depicts the properties, method, and event of `MultiView` Class.

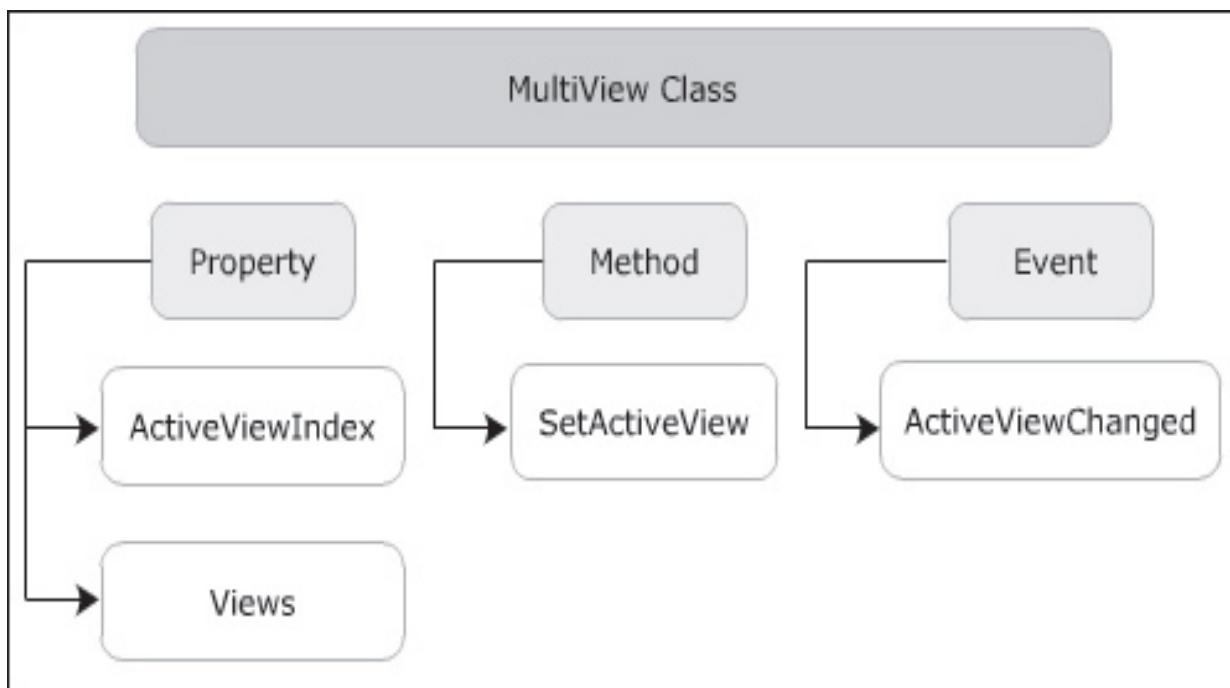


Figure 7.19: Property, Methods of MultiView Class

→ ActiveViewIndex Property

`ActiveViewIndex` property specifies or retrieves the index of the active `View` control.

The following code demonstrates how to set the `ActiveViewIndex` property.

Code Snippet:

```
mvwBookList.ActiveViewIndex = 0;
```

In this code, `ActiveViewIndex` is set to 0 to activate the first `View` control.

→ Views Property

`Views` property retrieves the collection of `View` controls.

The following code demonstrates how to add `View` controls of `ViewCollection` collection of the `MultiView` control.

Code Snippet:

```
mvwBookList.Views.Add(vwNovel);
mvwBookList.Views.Add(vwStory);
```

In this code, `Add()` method adds the `View` controls, `vwNovel` and `vwStory`, to a `MultiView` named `mvwBookList`.

→ SetActiveView Method

`SetActiveView` method sets the specified `View` control to the active view.

The following code demonstrates how to set a `View` control within a `MultiView` control as the active view.

Code Snippet:

```
mvwBookList.SetActiveView(vwStory);
```

In this code, `SetActiveView()` method activates the `View` control `vwStory`.

→ ActiveViewChanged Event

`ActiveViewChanged` event occurs when the active `View` control of a `MultiView` control changes.

The following code demonstrates the use of `ActiviewChanged` event.

Code Snippet:

```
void mvBookList_ActiveViewChanged(object sender, EventArgs e)
{
    Response.Write("Current activated view has been changed.");
}
```

In this code, the message Current activated view has been changed is displayed when the active view changes.

Knowledge Check 3

- Can you match the elements of FileUpload, View, and MultiView controls with their corresponding descriptions?

Description		Element	
(A)	Occurs when the current View control becomes the active view.	(1)	EnableTheming property
(B)	Saves the contents of a file that is uploaded.	(2)	Activate event
(C)	Specifies or retrieves the index of the active View control.	(3)	FileName property
(D)	Specifies or retrieves a value indicating whether themes apply to this control.	(4)	SaveAs method
(E)	Retrieves the name of a file selected for uploading.	(5)	ActiveViewIndex property

(A)	(A)-(2), (B)-(4), (C)-(5), (D)-(1), (E)-(3)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(4), (C)-(1), (D)-(5), (E)-(2)	(D)	(A)-(4), (B)-(3), (C)-(5), (D)-(1), (E)-(2)

- Which of the following statements about FileUpload, View and MultiView control are true?

(A)	The FileUpload control creates only a text box for locating files on the local folders.
(B)	FileName property retrieves the name of a file selected for download.
(C)	The View control can be referred to as a container of controls.
(D)	FileUpload class belongs to System.Web.UI.WebControls namespace.
(E)	MultiView control can be referred to as a container of View controls.

(A)	A, B, D	(C)	B, E
(B)	B, C	(D)	C, D, E

Module Summary

In this module, **Advanced Web Controls**, you learnt about:

→ **Navigation Web Controls**

Navigation Web controls allow easy navigation through large Web sites. Menu, TreeView, and SiteMapPath controls are the Navigation Web Controls provided by ASP.NET.

→ **ImageMap Control and HotSpot Classes**

Hot spots are clickable regions on an image which act as links. The ImageMap control is used to create images containing hot spots. The HotSpot class implements the functionality common to all hot spot shapes, namely circle, rectangle, and polygon.

→ **Advanced Web Server Controls**

FileUpload, View, and MultiView controls are some of the advanced Web server controls. FileUpload control provides the functionality of locating and uploading files. The View and MultiView controls allow dividing large Web pages into smaller logical chunks.

Module- 8

HTML Server Controls

Welcome to the module, **HTML Server Controls**.

Welcome to the module, HTML Server Controls. The module discusses the need for HTML server controls and highlights the differences between Web server controls and HTML server controls. Various HTML server controls like `HtmlForm` control, `HtmlImage` control, and `HtmlSelect` control are introduced in this module and the `System.Web.UI.HtmlControls` namespace is also discussed.

In this module, you will learn to:

- ➔ HTML Server Controls
- ➔ `System.Web.UI.HtmlControls` Namespace
- ➔ HTML Server Controls in ASP.NET

Web Development
`http://www`



8.1 HTML Server Controls

In this first lesson, **HTML Server Controls**, you will learn to:

- Describe the HTML Server Controls.
- Differentiate between Web server and HTML server controls.

8.1.1 Need for HTML Server Controls

By default, the HTML tags that are used to create HTML controls in an ASP.NET application are considered as plain text and not as server-side code. Hence, on the server, HTML controls are not recognized as controls. For them to be recognized on the server, HTML controls need to be converted into HTML server controls.

HTML server controls provide access to properties and attributes of the HTML tags. They also allow moving the logic affecting the tags away from the tags themselves. This results in neatly written codes.

HTML server controls in ASP.NET also allow the implementation of view state management and validation.

Figure 8.1 demonstrates the concept.

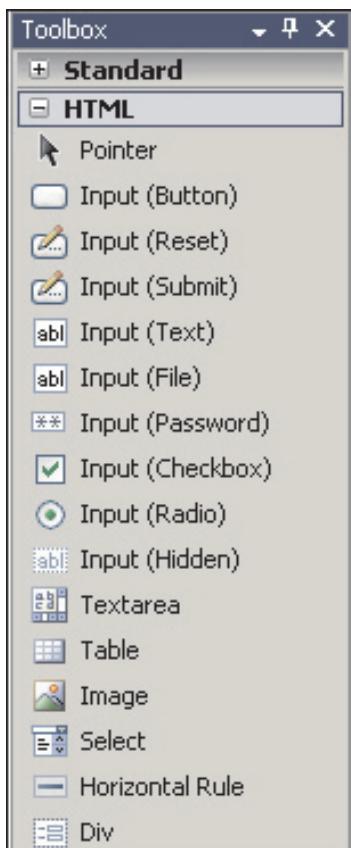


Figure 8.1: HTML Server Controls

8.1.2 Introduction to HTML Server Controls

HTML server controls in ASP.NET are a set of controls that resemble the corresponding HTML tags. These controls are declared on the Web page by adding the `runat="server"` attribute inside the `<form>` tag. The `runat="server"` attribute specifies that the HTML control is a server control. If the `runat` attribute is not specified within the HTML tag, by default, the control will be an HTML control.

Figure 8.2 demonstrates the concept.

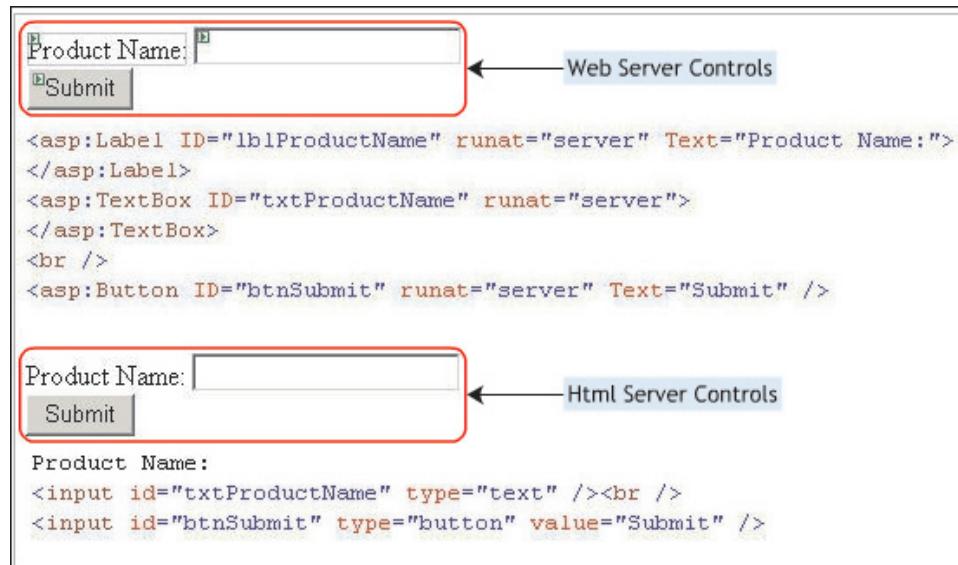


Figure 8.2: Web and Html Server Controls

Some of the basic HTML server controls are as follows:

- **HtmlButton**
- **HtmlImage**
- **HtmlForm**
- **HtmlTable**
- **HtmlInputText**
- **HtmlInputButton**
- **HtmlSelect**

8.1.3 Difference between Web Server and HTML Server Controls

HTML server controls have similar syntax and rendering methods like HTML controls whereas Web server controls are directly related to .NET Framework and have a wider range of special purpose controls.

Table 8.1 lists the differences between Web server controls and HTML server controls.

Web Server Controls	HTML Server Controls
ASP.NET Web server controls can be used only for server-side scripting.	HTML server controls can be used for client-side scripting as well as server-side scripting depending on the code.
ASP.NET Web server controls can be directly used for server-side scripting.	HTML server controls can be used for server-side scripting by adding <code>runat="server"</code> attribute in the HTML tag.
ASP.NET Web server controls can detect the client browser and render themselves accordingly.	HTML server controls are unable to identify the client browser.
ASP.NET Web server controls have a higher level of abstraction. A Web server control can be a result of many HTML tags that combine to create the control and its events.	HTML server controls have similar abstraction with the corresponding HTML tags and offer no abstraction with other controls.
Calendar, AdRotator, and ListBox are examples of Web server controls.	<code>HtmlImage</code> , <code>HtmlInputButton</code> , and <code>HtmlSelect</code> are examples of HTML server controls.

Table 8.1: Difference between Web and Html Server Controls

Knowledge Check 1

1. Which of the following statements about Web server and HTML server controls are true?

(A)	HTML server controls can be used for server-side scripting by adding <code>runat="server"</code> attribute in the HTML tag.
(B)	HTML controls in ASP.NET application are referred to as server-side code.
(C)	Web server controls are used for server-side scripting.
(D)	HTML tags are combined with one another to create HTML server controls.
(E)	<code>HtmlImage</code> and <code>AdRotator</code> are the Web server controls.

(A)	A, B, D	(C)	B, E
(B)	A, C	(D)	C, D, E

8.2 System.Web.UI.HtmlControls Namespace

In this second lesson, **System.Web.UI.HtmlControls Namespace**, you will learn to:

- Describe briefly the `System.Web.UI.HtmlControls` namespace.
- List the commonly used classes in the `System.Web.UI.HtmlControls` namespace.

8.2.1 System.Web.UI.HtmlControls Namespace

A Web application typically contains forms and GUIs with user elements such as labels and text boxes. Although this can be created using Web server controls, it is sometimes necessary to create these forms using HTML server controls because of the benefits offered by these controls.

`System.Web.UI.HtmlControls` namespace is a collection of classes that help in creating HTML server controls on a Web Forms page. The standard HTML tags are matched by the HTML server controls that run on the server. The HTML server controls run on the server and map to standard HTML tags that are supported by most browsers. These controls can thus, be programmatically controlled on the Web Forms page.

Figure 8.3 shows the class hierarchy of the `UI.HtmlControls` namespace.

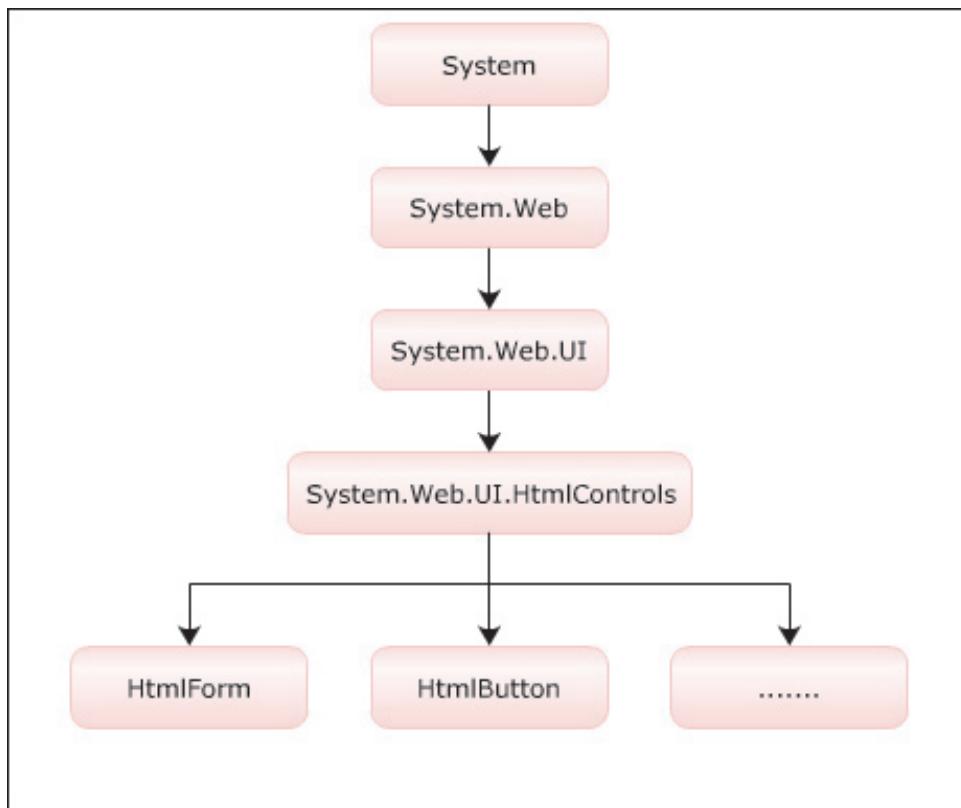


Figure 8.3: Class Hierarchy of `UI.HtmlControls`

8.2.2 Common Properties of System.Web.UI.HtmlControls Classes

The `System.Web.UI.HtmlControls` namespace consists of various classes such as `HtmlInputButton`, `HtmlImage`, `HtmlForm`, `HtmlAnchor`, and `HtmlTable`. These classes share some common properties, a few of which are listed in table 8.2:

Property	Description
Attributes	Retrieves a collection of attribute name and value pairs of all controls on the Web form.
ID	Specifies or retrieves an identifier that has been assigned to the control.
NamingContainer	Retrieves a reference to the naming container of the control, creating a unique namespace to help differentiate between server controls that have identical identifiers.
TagName	Retrieves the tag name of the control that contains <code>runat= "server"</code> attribute.
UniqueID	Retrieves the unique identifier that has been assigned to the control.

Table 8.2: Properties of `HtmlControls` Class

8.2.3 Classification of Controls

The controls in the `System.Web.UI.HtmlControls` namespace can be classified into categories such as container controls, input controls, and image controls based on their functionality. However, there is no such formal classification.

All controls in the `System.Web.UI.HtmlControls` namespace share some common properties, methods, and events. These properties, methods, and events are defined in the `HtmlControl` class of this namespace and can be inherited by all HTML server control classes.

Figure 8.4 shows the classification of HTML Server controls.

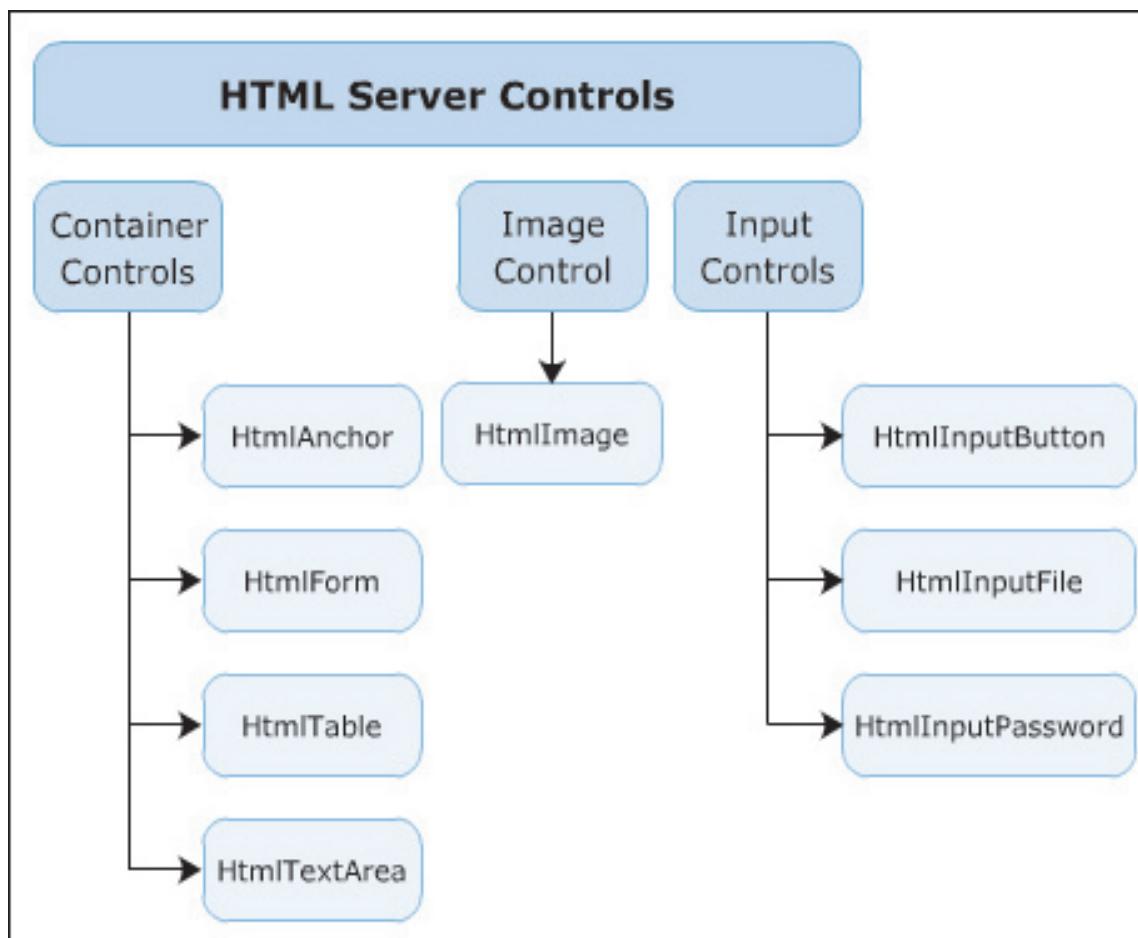


Figure 8.4: Classification of Html Server Controls

8.2.4 Commonly Used Classes of System.Web.UI.HtmlControls Namespace

Some of the commonly used classes of the `System.Web.UI.HtmlControls` namespace are listed in table 8.3 along with their description.

Class	Description
<code>HtmlButton</code>	Allows the developer to programmatically use the HTML <code><button></code> tag on the server.
<code>HtmlForm</code>	Enables the developer to programmatically use the HTML <code><form></code> tag on the server.
<code>HtmlImage</code>	Allows the developer to programmatically use the HTML <code></code> tag on the server.

Class	Description
HtmlInputButton	Allows the developer to programmatically use the HTML <code><input type= button></code> , <code><input type= submit></code> , and <code><input type= reset></code> tags on the server.
HtmlInputText	Allows the developer to programmatically use the HTML <code><input type= text></code> and <code><input type= password></code> tags on the server.
HtmlTable	Allows the developer to programmatically use the HTML <code><table></code> tag on the server.
HtmlSelect	Allows the developer to programmatically use the HTML <code><select></code> tag on the server.

Table 8.3: UI.HtmlControls Classes

Knowledge Check 2

1. Which of the following statements about `System.Web.UI.HtmlControls` namespace are true?

(A)	Classes of <code>System.Web.UI.HtmlControls</code> namespace help in creating HTML server controls on the Web Forms pages.
(B)	<code>HtmlTable</code> control allows the developer to programmatically use the <code><select></code> tag on the server.
(C)	<code>HtmlForm</code> class allows the developer to programmatically use the <code><form></code> tag on the server.
(D)	<code>HtmlInputButton</code> class allows the developer to programmatically use the HTML <code><input type= password></code> tag on the server.
(E)	<code>System.Web.UI.HtmlControls</code> namespace provides a collection of Web server control classes.

(A)	A, B, D	(C)	B, E
(B)	A, C	(D)	C, D, E

2. Can you match the classes of HTML server controls with their corresponding description?

Description		Class	
(A)	Maps to the HTML <select> tag on the server.	(1)	HtmlButton
(B)	Maps to the HTML <input type="text"> element on the server.	(2)	HtmlForm
(C)	Maps to the HTML <button> tag on the server.	(3)	HtmlImage
(D)	Maps to the HTML <form> tag on the server.	(4)	HtmlSelect
(E)	Maps to the HTML tag on the server.	(5)	HtmlInputText

(A)	(A)-(2), (B)-(4), (C)-(5), (D)-(1), (E)-(3)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(4), (C)-(1), (D)-(5), (E)-(2)	(D)	(A)-(4), (B)-(5), (C)-(1), (D)-(2), (E)-(3)

8.3 HTML Server Controls in ASP.NET

In this last lesson, **HTML Server Controls in ASP.NET**, you will learn to:

- Explain the `HtmlForm` control.
- List and describe the HTML server input controls.
- Describe the `HtmlImage` control.
- Describe the `HtmlSelect` control.

8.3.1. *HtmlForm* Control

Consider that you are developing an online shopping form in ASP.NET. The form will typically have text boxes to enter data, buttons to submit data or cancel the process, labels to display text, and so on. These controls are added to the Web Form by using the HTML `<form>` tag.

The `HtmlForm` control represents the HTML `<form>` element. The `HtmlForm` control can be referred to as a container of server controls. All controls posting back to the server have to be placed between the opening and closing tags of the `HtmlForm` control.

There can only be one `HtmlForm` control on a Web page.

Figure 8.5 demonstrates the concept.

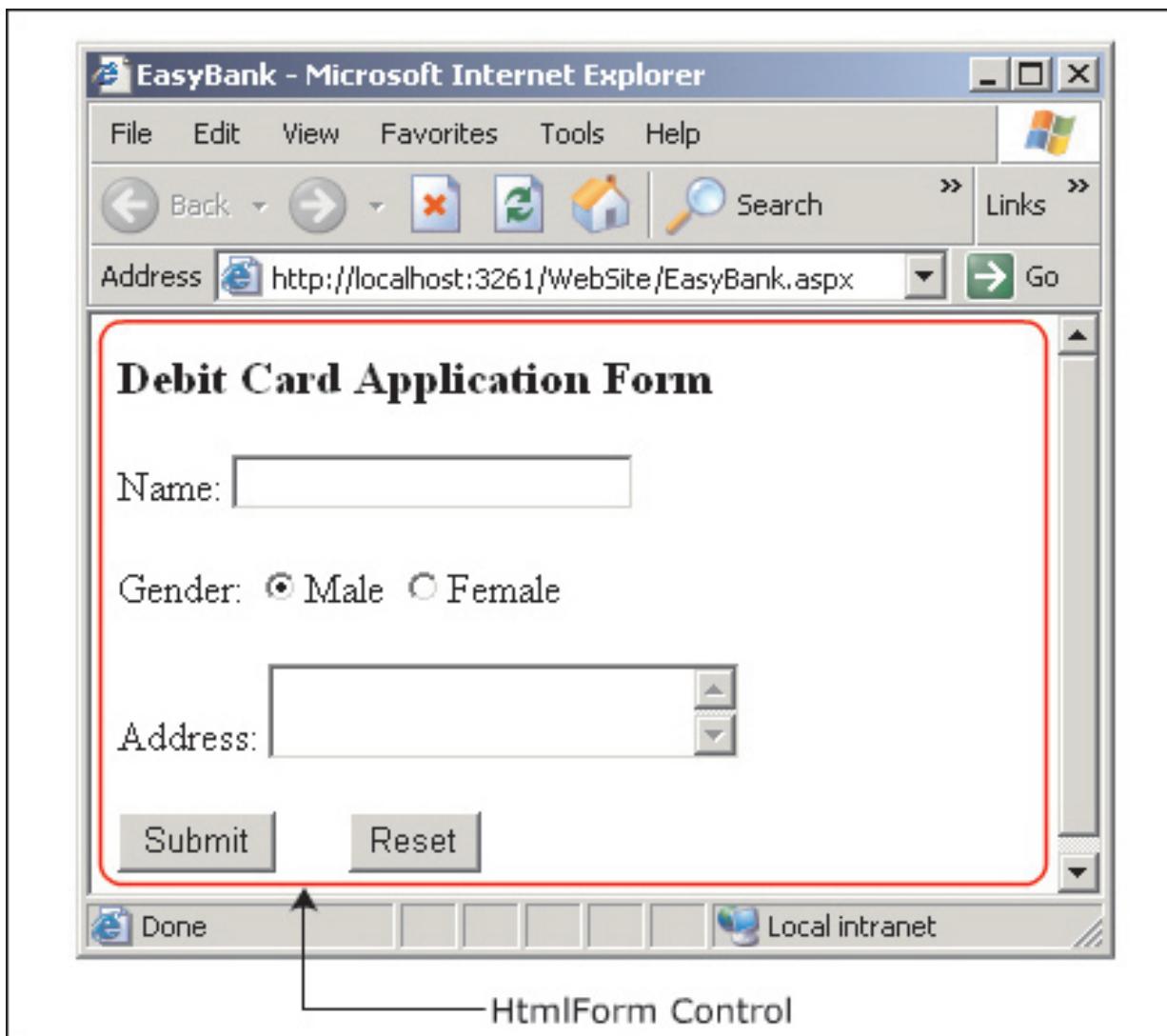


Figure 8.5:HtmlForm Control

The following code demonstrates how to create an `HtmlForm` control.

Code Snippet:

```
HtmlForm hfrmInvoiceDetails = new HtmlForm();
Page.Controls.Add(hfrmInvoiceDetails);
```

The code snippet creates an `HtmlForm` control named `hfrmInvoiceDetails`. The newly created form, `hfrmInvoiceDetails`, is added to the Web page. The `Add()` method of the `ControlCollection` class is used to add the control to the page.

8.3.2 Properties of HtmlForm Control

The `HtmlForm` control acts as a container and is used to present the various controls in a Web Forms page in a systematic order.

Some of the important properties of the `HtmlForm` control are as follows:

- ➔ Attributes
- ➔ DefaultButton
- ➔ DefaultFocus
- ➔ Method
- ➔ Name

These are described as follows:

➔ **Attributes**

The `Attributes` property retrieves a collection of attributes name and value pairs of all controls on the form.

The following code demonstrates the use of the `Attributes` property of the `HtmlForm` control.

Code Snippet:

```
// Returns the attribute name 'runat' and value 'server' of the element.
hfrmInvoiceDetails.Attributes["runat"] = "server";
```

In this code, the `Attributes` property retrieves the value `server` for the `runat` attribute.

➔ **DefaultButton**

The `DefaultButton` property specifies or retrieves the child control on the `HtmlForm` that causes postback when the ENTER key is pressed.

The following code demonstrates the use of the `DefaultButton` property of the `HtmlForm` control.

Code Snippet:

```
// Sets the button having ID hinbtnSubmit as the default button
hfrmInvoiceDetails.DefaultButton = "hinbtnSubmit";
```

In this code, the button named `hinbtnSubmit` is set as the default button on the `HtmlForm` control `hfrmInvoiceDetails`.

→ DefaultFocus

The `DefaultFocus` property specifies or retrieves the control on the form with the input focus when the `HtmlForm` control is loaded.

The following code demonstrates the use of the `DefaultFocus` property of the `HtmlForm` control.

Code Snippet:

```
// Set the default focus of a HtmlForm control to a TextBox control
hfrmInvoiceDetails.DefaultFocus = "hintxtFirstName";
```

In this code, `hintxtFirstName` is an object of `HtmlInputText` control. The `hintxtFirstName` is the identifier for the text box control. The `DefaultFocus` property of the `HtmlForm` control `hfrmInvoiceDetails` is set to the text box control `hintxtFirstName`.

→ Method

The `Method` property specifies the way to post data to the server. The possible values are “post” and “get”. The default value is “post”.

The following code demonstrates the use of `Method` property of the `HtmlForm` control.

Code Snippet:

```
// Set the Method property to POST for submitting form data information and
// which will then be sent to the server
hfrmInvoiceDetails.Method = "POST";
```

In this code, the `Method` property of the form `hfrmInvoiceDetails` is set to `Post`. This causes any submitted values to be displayed in the browser.

→ Name

The `Name` property specifies the name of the form and it has to be unique when multiple forms are attached in the same ASP.NET project.

The following code demonstrates the use of `Name` property of the `HtmlForm` control.

Code Snippet:

```
// To specify the identifier name for the HtmlForm
hfrmInvoiceDetails.Name = "frmInvoiceDetails";
```

In this code, `frmInvoiceDetails` is the identifier for the `HtmlForm` control `hfrmInvoiceDetails`.

Figure 8.6 shows an example demonstrating the properties of the `HtmlForm` control.

```

using System;
using System.Data;
using System.Web.UI;
using System.Web.UI.HtmlControls;

public partial class InvoiceDetails : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        HtmlForm hfrmInvoiceDetails = new HtmlForm();
        HtmlInputText hintxtFirstName = new HtmlInputText();
        hintxtFirstName.ID = "hintxtFirstName";
        hfrmInvoiceDetails.Attributes["runat"] = "server";
        hfrmInvoiceDetails.DefaultFocus = "hintxtFirstName";
        hfrmInvoiceDetails.Method = "POST";
        hfrmInvoiceDetails.Name = "frmInvoiceDetails";
        hfrmInvoiceDetails.Controls.Add(hintxtFirstName);
        Page.Controls.Add(hfrmInvoiceDetails);
    }
}

```

Figure 8.6: Example for Properties of `HtmlForm`

8.3.3 HTML Server Input Controls

HTML input controls like `Text`, `CheckBox`, `Image`, and `Button` are converted to HTML server input controls by adding the `runat="server"` and the `id` attributes in the HTML tag. The corresponding HTML server input controls created are `HtmlInputText`, `HtmlInputCheckBox`, `HtmlInputImage`, and `HtmlInputButton`.

→ **HtmlInputText**

`HtmlInputText` control creates a server-side code that maps to the `<input type = text>` and `<input type=password>` HTML elements. This control allows the developer to create a single line text box to accept user input.

Table 8.4 lists the most commonly used properties of `HtmlInputText` control.

Property	Description
<code>ID</code>	Specifies or retrieves an identifier that has been assigned to the control.
<code>MaxLength</code>	Specifies or retrieves the maximum number of characters that can be entered inside the text box.
<code>Name</code>	Specifies or retrieves the unique name for the control.
<code>Size</code>	Specifies or retrieves the width of the text box depending on the layout of the form.
<code>Visible</code>	Specifies or retrieves a value indicating whether or not the control is rendered on the user interface.

Table 8.4: Properties of `HtmlInputText` Control

The following code demonstrates use of the `HtmlInputText` control.

Code Snippet:

```
HtmlInputText hintxtFirstName = new HtmlInputText();

protected void Page_Load(object sender, EventArgs e)
{
    // Specify an identifier (ID) to textbox control
    hintxtFirstName.ID = "hintxtFirstName";

    // Specifies the maximum length (character limit) of TextBox control
    hintxtFirstName.MaxLength = 20;

    // Specifies the width of textbox control
    hintxtFirstName.Size = 25;

    // Specifies the value for the textbox control
    hintxtFirstName.Value = "John";
```

```
// Adds the textbox control to the HtmlForm
hfrmInvoiceDetails.Controls.Add(hintxtFirstName);
}
```

In this code, `hintxtFirstName` is the object of the `HtmlInputText` class. The `ID` property is used to assign an identifier `hintxtFirstName` to the control. The `MaxLength` property specifies that the control can accept only 20 characters. The `Size` property specifies the size of the control as 25. The `Value` property sets the value of the control to `John`. The `Add()` method of the `ControlCollection` class is used to add the control `hintxtFirstName` on the form `hfrmInvoiceDetails`.

→ **HtmlInputCheckBox**

`HtmlInputCheckBox` control allows the developer to access the HTML `<input type="checkbox">` element on the server. This control creates an event handler that executes a set of instructions each time the control raises an event. Table 8.5 lists the commonly used properties and events of the control.

Name	Description
Checked property	Specifies or retrieves a value that indicates whether or not the control is selected.
Name property	Specifies or retrieves a unique identifier name for the control.
Value property	Specifies or retrieves the value associated with the control.
Visible property	Specifies or retrieves a value indicating whether or not the control is rendered on the user interface.
ServerClick event	Occurs on the server when <code>HtmlInputCheckBox</code> control is clicked.

Table 8.5: Properties of `HtmlInputCheckBox` Control

The following code demonstrates use of the `HtmlInputCheckBox` control.

Code Snippet:

```
HtmlInputCheckBox hinchkProduct = new HtmlInputCheckBox();
protected void Page_Load(object sender, EventArgs e)
{
    // To specify the identifier name for the HtmlInputCheckBox
    hinchkProduct.Name = "hinchkProduct";
```

```

// To specify the value for the Checkbox
hinchkProduct.Value = "Furniture";

// To specify the Visible property of Checkbox to true
hinchkProduct.Visible = true;
// To set the checked state to true
hinchkProduct.Checked = true;

// To add checkbox to the form
hfrmInvoiceDetails.Controls.Add(hinchkProduct);
...
}

void hinchkProduct_ServerChange(object sender, EventArgs e)
{
    // Checks whether checkbox named 'hinchkProduct' is checked.
    if (hinchkProduct.Checked)
    {
        Response.Write("You have selected the product.");
    }
}

```

In this code, **hinchkProduct** is the object of `HtmlInputCheckBox` class. The `Page_Load` event occurs when the page is loaded for the first time. The `Name` property specifies `hinchkProduct` as the name of the control. The `Value` property assigns the value `Furniture` to the control. The `Visible` property is set to `True` to ensure that the control is seen on the Web page. The `Checked` property is set to `true`. The `Add()` method is used to add the control `hinchkProduct` on the form `hfrmInvoiceDetails`. The `ServerChange` event occurs when the control is clicked. If the control has been selected, then an appropriate message is displayed using `Response.Write()` method.

→ **HtmlInputImage**

`HtmlInputImage` control creates a button that displays an image on the Web Form.

Table 8.6 lists the commonly used properties and the events of the control.

Name	Description
Align property	Specifies or retrieves the alignment of the control with respect to other controls on the Web page.
Alt property	Specifies or retrieves the alternate text to be displayed by the browser when the image is not displayed on the Web page.
ID property	Specifies or retrieves the identifier assigned to the control.
Src property	Specifies or retrieves the path where the image file is saved.
ServerClick event	Occurs on the server when HtmlInputImage control is clicked.

Table 8.6: Properties of HtmlInputImage

The following code demonstrates use of the HtmlInputImage control.

Code Snippet:

```
HtmlInputImage hinimgProduct = new HtmlInputImage();

protected void Page_Load(object sender, EventArgs e)
{
    // Specify an identifier
    hinimgProduct.ID = "hinimgProduct";

    // Specify image description
    hinimgProduct.Alt = "This is a product";

    // To display image on the left side of the Web page
    hinimgProduct.Align = "Left";

    // Specify the file name for picture
    hinimgProduct.Src = "Chair.jpg";
```

```
// To add input image to the form
hfrmInvoiceDetails.Controls.Add(hinimgProduct);

...
}

void hinimgProduct_ServerClick(object sender, ImageClickEventArgs e)
{
    Response.Write("You clicked on the image.");
}
```

In this code, `hinimgProduct` is the object of `HtmlInputImage` class. The `Page_Load` event occurs when the page is loaded for the first time. The `ID` property assigns an identifier name `hinimgProduct` to the control. The `Alt` property specifies an image description noting that the image is of a product. The `Align` property specifies the alignment of the image to the `Left` side of the Web page. The `Src` property specifies the path for the image. If the image file is saved in the same folder where the ASP.NET application is saved, only the name of the image is specified. The `Add()` method is used to add the control `hinimgProduct` on the form `hfrmInvoiceDetails`.

→ **HtmlInputButton**

`HtmlInputButton` control creates a button on the Web page. Depending on the input type of the button, the form submits or resets the data or simply pushes the button. Table 8.7 lists the commonly used properties and events of the control.

Name	Description
Attributes property	Retrieves a collection of attribute name and value pairs for the control.
ValidationGroup property	Specifies or retrieves the group of controls for which the <code>HtmlInputButton</code> control causes validation at the time of post back.
Value property	Specifies or retrieves the value associated with the control.
ServerClick event	Occurs on the server when <code>HtmlInputButton</code> control is clicked.

Table 8.7: Properties of `HtmlInputButton` Control

The following code demonstrates use of the `HtmlInputButton` control.

Code Snippet:

```
HtmlInputButton hinbtnClickHere = new HtmlInputButton();  
  
protected void Page_Load(object sender, EventArgs e)  
{    // To specify an identifier  
    hinbtnClickHere.ID = "hinbtnClickHere";  
  
    // To specify the value for button  
    hinbtnClickHere.Value = "Click Here";  
  
    // To enable the button control by setting Disabled property to false  
    hinbtnClickHere.Disabled = false;  
  
    // To add button to the form  
    hfrmInvoiceDetails.Controls.Add(hinbtnClickHere);  
  
    ...  
}  
  
void hinbtnClickHere_ServerClick(object sender, EventArgs e)  
{  
    Response.Write("Welcome to Invoice Details page.");  
}
```

In this code, `hinbtnClickHere` is the object of the `HtmlInputButton` class. The `Page_Load` event occurs when the page is loaded for the first time. The `ID` property assigns an identifier name `hinbtnClickHere` to the control. The `Value` property sets the value for the control to `Click Here`. The `Disabled` property is set to `False` to enable the click event of the button. The `Add()` method is used to add the control `hinbtnClickHere` to the form `hfrmInvoiceDetails`. The `ServerClick` event displays a welcome message on the page using `Response.Write()` method after the button is clicked.

8.4 *HtmlImage Control*

Images play a very important role in creating a Web site. A site is made more attractive and informative by adding suitable images. For this reason, developers use `` tag to insert images on the site.

The `HtmlImage` control maps the `` tag on the server. This control is used to display an image on the Web page. This control is treated as plain text on the server and not as a control unless the `runat="server"` attribute is added in the `` tag.

Figure 8.7 demonstrates the concept

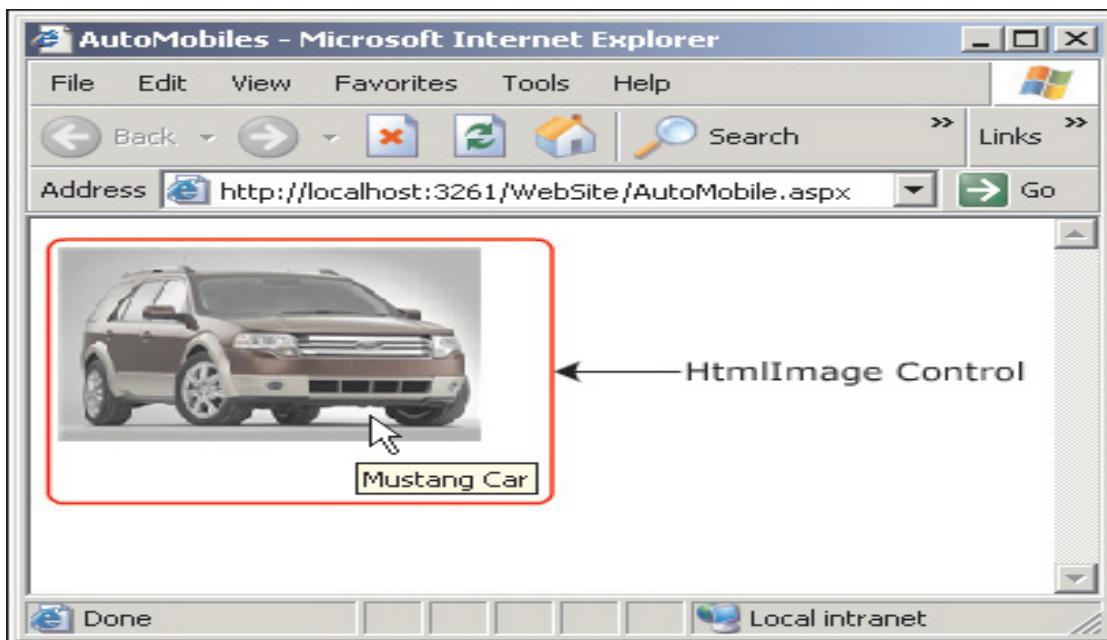


Figure 8.7: HtmlImage Control

The following snippet creates an `HtmlImage` control named `himgLogo`.

Code Snippet:

```
HtmlImage himgLogo = new HtmlImage();
```

8.4.1 Properties of `HtmlImage` Control

The `HtmlImage` control is available in the **Toolbox** of the Visual Studio 2005 IDE under **HTML**. The control is labeled as **Image**.

Some of the important properties of the `HtmlImage` class that represents this control are as follows:

- Align
- Alt
- ID
- Height
- SRC

→ Width

Figure 8.8 shows an example demonstrating the properties of the `HtmlImage` control.

```
using System;
using System.Data;
using System.Web.UI;
using System.Web.UI.HtmlControls;

public partial class ProductDetails : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        HtmlForm hfrmProductDetails = new HtmlForm();
        HtmlImage himgLogo = new HtmlImage();
        himgLogo.Align = "Right";
        himgLogo.Alt = "Company logo";
        himgLogo.Width = 100;
        himgLogo.Height = 175;
        himgLogo.Src = "Logo.jpg";
        hfrmProductDetails.Controls.Add(himgLogo);
        Page.Controls.Add(hfrmProductDetails);
    }
}
```

Figure 8.8: Example for Properties of `HtmlImage` Control

These are described as follows:

→ Align

The `Align` property specifies or retrieves the alignment of the image in relation to other controls on the Web page.

→ Alt

The `Alt` property specifies or retrieves an alternate text that is displayed by the browser when an image is not displayed or downloaded on the Web page.

The following code assigns the `Alt` property to the control.

Code Snippet:

```
// To specify the image description
himgLogo.Alt = "Company logo";
```

In this code, the `Alt` property specifies the alternate text `Company Logo` to the control `himgLogo`.

If the image is unable to be displayed on the browser, the text Company Logo will be displayed in its place.

→ **ID**

The `ID` property specifies or retrieves the identifier that is assigned to the server control.

→ **Height**

The `Height` property specifies or retrieves the height of the image.

The following code assigns the `Height` property to the control.

Code Snippet:

```
// To specify the height of the image  
himgLogo.Height = 175;
```

In this code, the `Height` property sets the height of the control `himglogo` to 175.

→ **SRC**

The `SRC` property specifies or retrieves a path where the image file is saved.

The following code assigns the `SRC` property to the control.

Code Snippet:

```
// To specify the file name for picture  
himgLogo.Src = "Logo.jpg";
```

In this code, the `SRC` property is used to specify the name of the image to be displayed on the Web page. The image is saved in the same folder where the ASP.NET file is saved.

→ **Width**

The `Width` property specifies or retrieves the width of the image.

The following code assigns the `Width` property to the control.

Code Snippet:

```
// To specify the width of the image  
himgLogo.Width = 100;
```

In this code, the `Width` property specifies the width of the control `himglogo` to 100.

8.4.2 HtmlSelect control

The `HtmlSelect` server control is a server-side control that maps to the `<select>` tag of the HTML element. The control allows a developer to create a drop-down list box. The `HtmlSelect` server control can be displayed as a list box if the `Size` property of the control is greater than 1.

Figure 8.9 shows the use of this control.

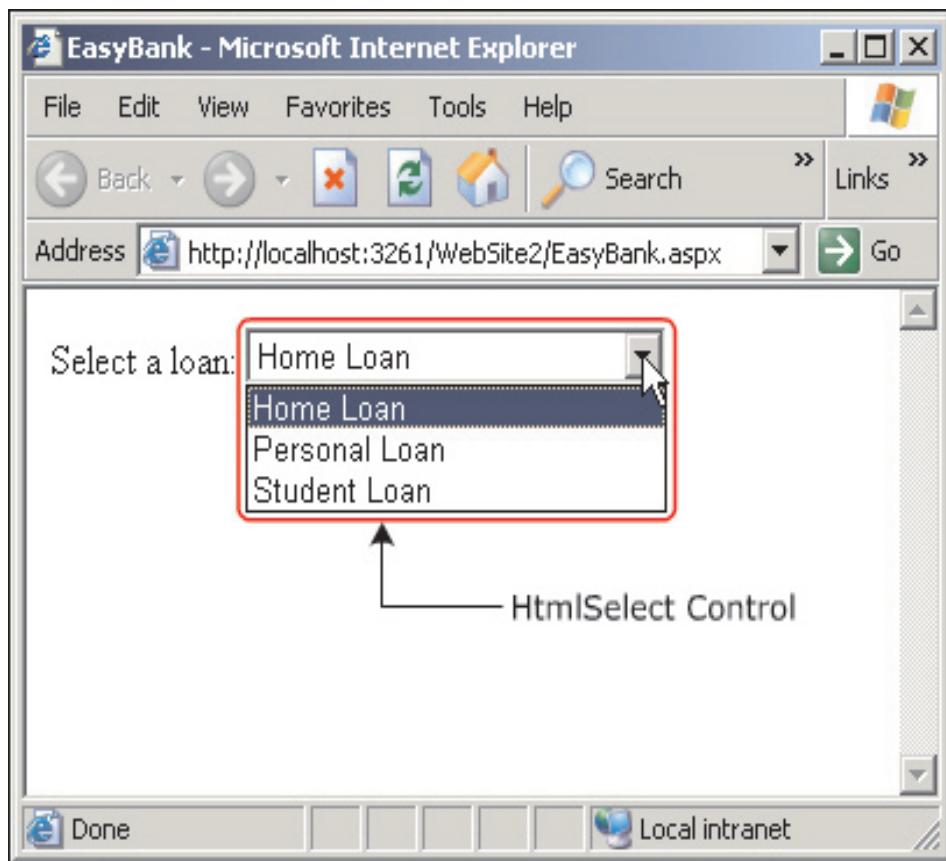


Figure 8.9: `HtmlSelect` Control

The following code creates an `HtmlSelect` control named `hslCategory`.

Code Snippet:

```
HtmlSelect hslCategory = new HtmlSelect();
```

8.4.3 Properties of HtmlSelect Control

The `HtmlSelect` server control is used to create a selection box. The items within the selection box are listed inside the `<option>` tag that is placed between the opening and the closing `<select>` tags.

Important properties of `HtmlSelect` class representing this control are as follows:

- ID
- Items
- Multiple
- SelectedIndex
- Value

Figure 8.10 shows an example demonstrating the properties of the `HtmlSelect` control.

```
using System;
using System.Data;
using System.Web.UI;
using System.Web.UI.HtmlControls;

public partial class ProductDetails : System.Web.UI.Page
{
    HtmlForm hfrmProductDetails = new HtmlForm();
    HtmlSelect hslCategory = new HtmlSelect();
    HtmlInputButton hinbtnGo = new HtmlInputButton();
    protected void Page_Load(object sender, EventArgs e)
    {
        hslCategory.ID = "hslCategory";
        hslCategory.Items.Add("Ordered");
        hslCategory.Items.Add("Unordered");
        hslCategory.Multiple = true;
        hinbtnGo.Value = "Go";
        hinbtnGo.ServerClick+=new EventHandler
            (hinbtnGo_ServerClick);
        hfrmProductDetails.Controls.Add(hslCategory);
        hfrmProductDetails.Controls.Add(hinbtnGo);
        Page.Controls.Add(hfrmProductDetails);
    }
    void hinbtnGo_ServerClick(object sender, EventArgs e)
    {
        Response.Write("Index Number : "
            + hslCategory.SelectedIndex);
    }
}
```

Figure 8.10: Example for properties of `HtmlSelect`

→ ID

The `ID` property specifies or retrieves the identifier that is assigned to the control.

The following snippet assigns `ID` property to the control.

Code Snippet:

```
// Specifies an identifier  
hslCategory.ID = "hslCategory";
```

In this code, the `ID` property is used to set an identifier `hslCategory` to the control `hslCategory`.

→ Items

The `Items` property retrieves a collection of items that are listed in the control.

The following snippet assigns `Items` property to the control.

Code Snippet:

```
// Add method of Items property adds items, namely 'Ordered' and 'UnOrdered'  
hslCategory.Items.Add("Ordered");  
hslCategory.Items.Add("Unordered");
```

In this code, the `Add()` method of the `Items` property is used to add items in the control `hslCategory`.

→ Multiple

The `Multiple` property specifies or retrieves a value which specifies whether multiple items in the list can be selected concurrently.

The following snippet demonstrates the use of `Multiple` property in the control.

Code Snippet:

```
// To enable multiple item selection  
hslCategory.Multiple = true;
```

In this code, the `Multiple` property is set to `true`, specifying that multiple items can be selected from the list.

→ SelectedIndex

The `SelectedIndex` specifies or retrieves the index number of the selected item on the list.

The following snippet assigns the `SelectedIndex` property to the control.

Code Snippet:

```
void hdnbtnGo_ServerClick(object sender, EventArgs e)
{
    Response.Write("You selected index number : " + hslCategory.SelectedIndex +
    "with value : " + hslCategory.Value);
}
```

In this code, the `SelectedIndex` property retrieves the index number of the selected item on the list.

→ Value

The `Value` property either retrieves the value of the item that is selected or sets the `SelectedIndex` property of the control to the index of the first item on the list.

Knowledge Check 3

- Can you match the properties of HTML server controls with their corresponding description?

Description		Property	
(A)	Specifies the way to post data to the server.	(1)	Multiple
(B)	Specifies or retrieves the control on the form with the input focus when the control is loaded.	(2)	Align
(C)	Specifies whether multiple items in the list can be selected.	(3)	Method
(D)	Retrieves a collection of items that are listed in the control.	(4)	Items
(E)	Specifies or retrieves the alignment of the image on the Web page.	(5)	DefaultFocus

(A)	(A)-(2), (B)-(4), (C)-(5), (D)-(1), (E)-(3)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(5), (C)-(1), (D)-(4), (E)-(2)	(D)	(A)-(4), (B)-(5), (C)-(1), (D)-(2), (E)-(3)

Module Summary

In this module, **HTML Server Controls**, you learnt about:

→ **HTML Server Controls**

The HTML server controls resemble HTML tags and are created by adding the `runat="server"` attribute in the HTML control tag. Unlike Web server controls, which can be used only for server-side scripting, HTML server controls can be used for client as well as server-side scripting depending on the code.

→ **System.Web.UI.HtmlControls Namespace**

`System.Web.UI.HtmlControls` namespace is a collection of classes used to create HTML Web server controls. `HtmlImage`, `HtmlSelect`, `HtmlInputText`, `HtmlForm` are some of the commonly used classes of this namespace and they represent the corresponding HTML server controls.

→ **HTML server controls in ASP.NET**

`HtmlSelect`, `HtmlInputText`, `HtmlInputCheckBox`, and `HtmlImage` are some of the commonly used HTML server controls. These controls allow performance of tasks such as displaying images, including check boxes, and displaying selectable lists on a Web page.

Module - 9

Validation Controls

Welcome to the module, **Validation Controls**.

Validation controls ensure the validity of input data. They perform checks on data to see if the data lies within specified limits, if the data is presented in the correct format and so on. ASP.NET provides various validation controls such as CompareValidator, RangeValidator, RequiredFieldValidator, and ValidationSummary. The classes representing these controls lie in the System.Web.UI.WebControls namespace.

In this module, you will learn to:

- Validation
- Validation Controls in ASP.NET

Web Development

http://www



9.1 Validation

In this first lesson, **Validation**, you will learn to:

- State the need for validation.
- List and describe the various validation controls in ASP.NET 2.0.

9.1.1 Overview of Validation

Validation is the process of determining the accuracy of the provided entity. In Web applications, input data needs to be validated to ensure processes run smoothly. Consider the process of blood transfusion. When we want to donate blood, we need to go through various tests to ensure that the donated blood is suitable for the recipient. Proper validation of the blood plays a major role in saving the patient's life.

Similarly, before accepting data from the user, it needs to be validated. Validation of the data determines its correctness in terms of attributes such as type, value, and format.

9.1.2 BaseValidator Class

The validation controls are a category of controls that provide features for Rapid Application Development (RAD). The RAD features of the validation controls assist you in automatic validation of the data input by the user.

The validation of values in Web server controls is done using validation controls provided by ASP.NET. You can include validation controls on a Web Form and assign them to the specific input Web server controls to validate the values entered by the user.

Validation controls are included in the `System.Web.UI.WebControls` namespace. The `BaseValidator` class serves as the base class for all the validation controls. Table 9.1 lists the commonly used properties and methods of the `BaseValidator` class.

Name	Description
<code>ControlToValidate</code> property	Specifies or retrieves the identifier of the input Web server control to be validated.
<code>Display</code> property	Specifies or retrieves the behavior of the error message to be displayed in a validation control. The behavior can be either None, Static, or Dynamic.
<code>ErrorMessage</code> property	Specifies or retrieves the text for the error message to be displayed when the validation is not performed.

Name	Description
IsValid property	Specifies or retrieves a value indicating whether the related input control passes through the validation process or not.
Validate() method	Validates the related input control and modifies the IsValid property.

Table 9.1: Properties of BaseValidator class

The following code demonstrates the use of the important properties and methods of the `BaseValidator` class.

Code Snippet:

```
BaseValidator valFirstName = new RequiredFieldValidator();
valFirstName.ControlToValidate = "txtFirstName";
valFirstName.Display = ValidatorDisplay.Dynamic;
valFirstName.ErrorMessage = "Enter first name.";
...
if (!valFirstName.IsValid)
{
    Response.Write("There is an error on the page.");
}
```

In the code, `valFirstName` validation control is an object of the `BaseValidator` class and is assigned as a `RequiredFieldValidator` control. This specifies that the user has to compulsorily enter a value for the `valFirstName` control. The `txtFirstName` is assigned to the validation control indicating that the input `Textbox` control is associated with the validation control. If any validation error occurs, the error message is displayed dynamically using the `Display` property. The `IsValid` property is used to confirm whether or not the input value in the `Textbox` control is valid. If the validation fails, an appropriate error message is displayed.

9.1.3 Validation Controls in ASP.NET 2.0

ASP.NET provides different validation controls for validation of values in Web server controls.

Table 9.2 lists the name, description, and example of the validation controls.

Control Name	Description	Example where it can be used
CompareValidator	Compares the value of one input control with another other input control or other constant value based on the comparison operator associated with it.	In a Confirm Password field to match the value entered by the user in the Password field.
CustomValidator	Evaluates the value of the input to check if it is valid according to the specified logic.	To check whether the value entered is a prime number or not.
RangeValidator	Checks whether the input value is within the specified range limit.	To assign grades to students on the basis of marks scored,
RegularExpressionValidator	Checks if the input value is in the specified format.	To check whether the zip code of a city is entered in the required format.
RequiredFieldValidator	Ensure that the input control is not left blank by the user.	To check whether values are entered in the mandatory fields of a form.
ValidationSummary	Allows reviewing of all error messages received from the validation controls.	To display a list of error messages that occurred during validation of an employee registration.

Table 9.2: Validation Controls and Description

9.1.4 New Features of Validation in ASP.NET 2.0

ASP.NET provides the `ValidationSummary` control to track errors when any validation error occurs on the Web page. The `ValidationGroup` property is new in ASP.NET. This property of a validation control can be set to associate the control with a particular group of controls. A `ValidationSummary` control can then be used to display validation errors occurring in the various controls of that group. This is done by associating the `ValidationSummary` control with the group's `ValidationGroup` property.

Figure 9.1 demonstrates the concept.

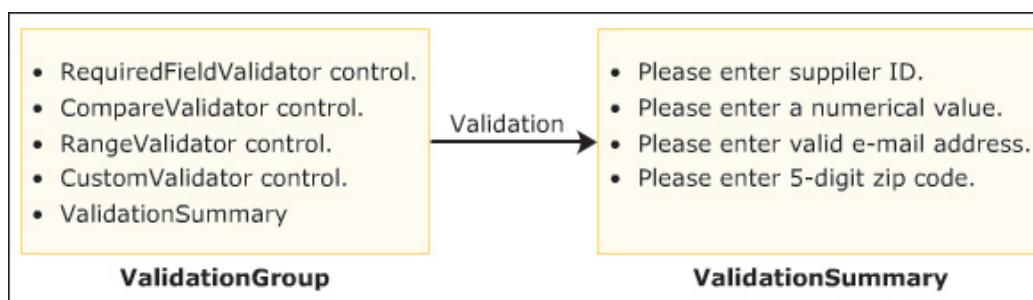


Figure 9.1: Validation Classes

Knowledge Check 1

- Which of the following statements about validation and validation controls are false?

(A)	Validation of the input data plays a role in ensuring smooth functioning of the Web application.
(B)	The <code>BaseValidate</code> class serves as the base class for the properties and methods common for all the validation controls
(C)	The <code>System.Web.UI.WebControls</code> namespace contains the validation controls.
(D)	The <code>ErrorMessage</code> property of the <code>BaseValidator</code> class allows viewing the error messages received from all the validation controls.
(E)	The <code>RequiredFieldValidator</code> control checks for ensuring that the input control is not left blank by the user.

(A)	A, B	(C)	B, D
(B)	A, C	(D)	C, D, E

2. Can you match the different validation controls in ASP.NET with their corresponding examples?

	Example	Validation Control	
(A)	Ensures that values for mandatory fields of a registration form are entered.	(1)	CompareValidator
(B)	Checks if the value entered in the confirm password field matches the value entered in the password field.	(2)	CustomValidator
(C)	Checks whether the value entered is a prime number.	(3)	RegularExpressionValidator
(D)	Displays error messages when multiple validations fail in a loan application form.	(4)	RequiredFieldValidator
(E)	Checks whether the zip code is entered in the required format.	(5)	ValidationSummary

(A)	(A)-(2), (B)-(4), (C)-(5), (D)-(1), (E)-(3)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(5), (C)-(1), (D)-(4), (E)-(2)	(D)	(A)-(4), (B)-(1), (C)-(2), (D)-(5), (E)-(3)

9.2 Validation Controls in ASP.NET

In this second lesson, **Validation Controls in ASP.NET**, you will learn to:

- Explain the `RequiredFieldValidator` control and its use.
- Describe the `CompareValidator` control.
- Describe the `RangeValidator` control.
- Describe the `RegularExpressionValidator` control.
- Describe the `CustomValidator` control.
- Explain the `ValidationSummary` control.
- State the use of `Page.IsValid` property.

9.2.1 RequiredFieldValidator Control

The `RequiredFieldValidator` control ensures that the user enters data in the input control.

If the user does not enter data in the input control which is associated with a `RequiredFieldValidator` control, the processing of the page halts until data is entered into the control.

The `RequiredFieldValidator` control is specially used when the user should be forced to enter data in the input control.

For example, in a registration form, the fields marked with '*' (asterisk) are mandatory and require to be filled. The `RequiredFieldValidator` control can be used to specify any input field as a mandatory field. If you do not enter any data in a mandatory field, an error message is displayed asking the user to enter data in that field. You can specify the error message by assigning an appropriate text message to the `Text` property of `RequiredFieldValidator` control.

Figure 9.2 demonstrates the concept.

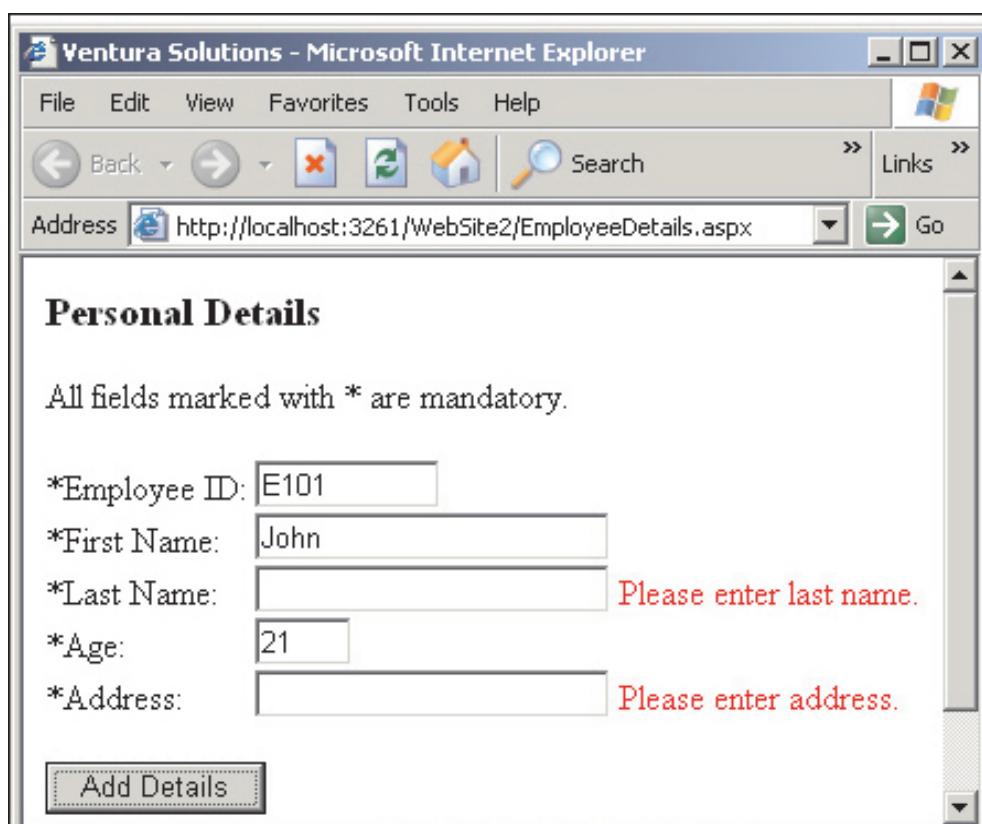


Figure 9.2: RequiredFieldValidator Control

9.2.2 Properties of the `RequiredFieldValidator` Class

The `RequiredFieldValidator` class represents the `RequiredFieldValidator` control.

Table 9.3 lists some of the important properties of the `RequiredFieldValidator` class.

Property	Description
InitialValue	Specifies or retrieves the initial value of the input control. The default initial value is "".
SetFocusOnError	Specifies or retrieves a value indicating whether focus is set automatically to the control indicated by the <code>ControlToValidate</code> property when the validation fails.
Text	Specifies or retrieves the text to be displayed on the validation control when the validation fails.

Table 9.3: Properties of RequiredFieldValidator Class

The following code demonstrates the use of the important properties and methods of the `RequiredFieldValidator` class.

Code Snippet:

```
//An input TextBox control txtAccountNumber is created
...
RequiredFieldValidator valAccountNumber = new RequiredFieldValidator();
valAccountNumber.ID = "valAccountNumber";
valAccountNumber.ControlToValidate = "txtAccountNumber";
// Specifies the initial value of associated input control.
valAccountNumber.InitialValue = "";
// Specifies the value that indicates whether focus is set to the control when validation fails.
valAccountNumber.SetFocusOnError = true;
// Specifies the text to be displayed when validation fails.
valAccountNumber.Text = "Enter valid account number.";
// To add validation control to the form.
frmBank.Controls.Add(valAccountNumber);
...
```

In the code, it is assumed that a `TextBox` control `txtAccountNumber` is created and added to the form. A `RequiredFieldValidator` control named `valAccountNumber` is created. The unique identifier `valAccountNumber` is set to the `RequiredFieldValidator` control

using the `ID` property. The input `Textbox` control is assigned to the `RequiredFieldValidator` control using the `ControlToValidate` property, which indicates that the user must compulsorily enter a value in the `TextBox` control, otherwise the validation will fail. The initial value of the input `TextBox` control, `txtAccountNumber`, is set to a blank space using the `InitialValue` property. The `SetFocusOnError` property is set to `true`, which indicates that the focus is set to the control when the validation fails. The `Text` property of the `RequiredFieldValidator` control displays the specified error message when the validation fails. The validation control `valAccountNumber` is added to the form using the `Add()` method.

9.2.3 CompareValidator Control

The `CompareValidator` control compares the value of one input control with another input control or a constant value. You can specify the data type of the input values to be compared using the `BaseCompareValidator.Type` property. Values of different data types can be compared using the `CompareValidator` control. The different types of data that can be compared using the `CompareValidator` control are `String`, `Integer`, `Double`, `Date`, and `Currency`.

The main drawback of the `CompareValidator` control is that it cannot validate the values of the controls located in two different Web pages.

For example, when you register for a user account, the `CompareValidator` control evaluates value entered in the **Password** field with the value entered in the **Confirm Password** field. Figure 9.3 demonstrates the concept.

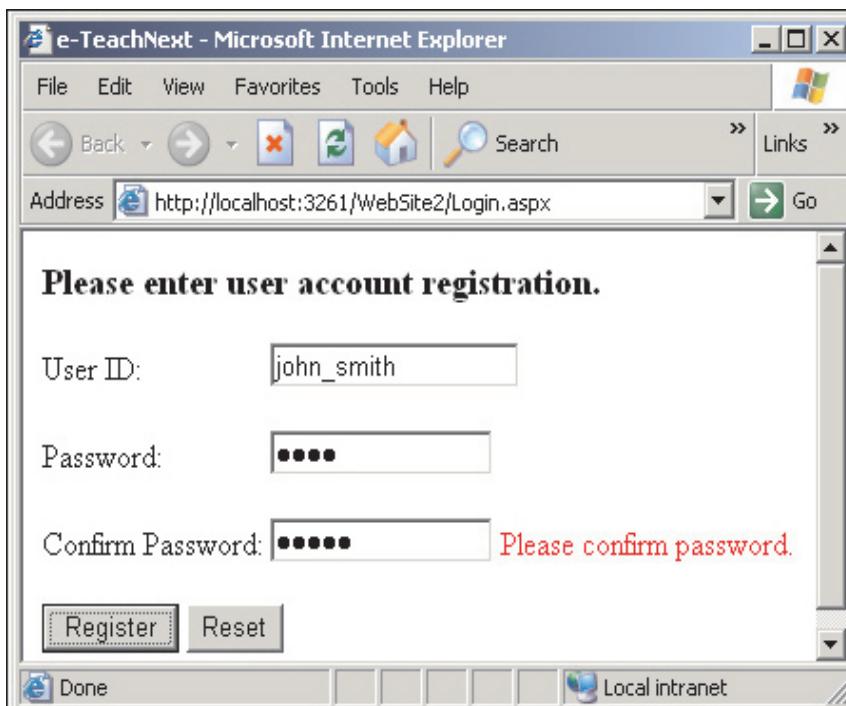


Figure 9.3: Example of CompareField Validator Control

Similarly, when you apply for a driver's license, the `CompareValidator` control can compare your age to the constant representing the required minimum age.

9.2.4 Property, Methods and Events of the CompareValidator Class

The CompareValidator class represents the CompareValidator control.

Table 9.4 lists some of the important properties and events of the CompareValidator class.

Name	Description
Operator property	Specifies or retrieves the operator to perform the comparison operation.
Text property	Specifies or retrieves the text to be displayed on the validation control when the validation fails.
ValueToCompare property	Specifies or retrieves the constant value to be compared with the value entered by the user in the user control.

Table 9.4: Properties of CompareValidator Class

The following code demonstrates the use of the important properties and methods of the CompareValidator class.

Code Snippet:

```
TextBox txtConfirmPass = new TextBox();
txtConfirmPass.ID = "txtConfirmPass";
frmLogin.Controls.Add(txtConfirmPass);
...
CompareValidator valConfirmPass = new CompareValidator();
valConfirmPass.ControlToValidate = "txtConfirmPass";
valConfirmPass.Operator = ValidationCompareOperator.Equal;
valConfirmPass.Text = "Enter the valid password.";
valConfirmPass.ValueToCompare = "abcd";
frmLogin.Controls.Add(valConfirmPass);
...
```

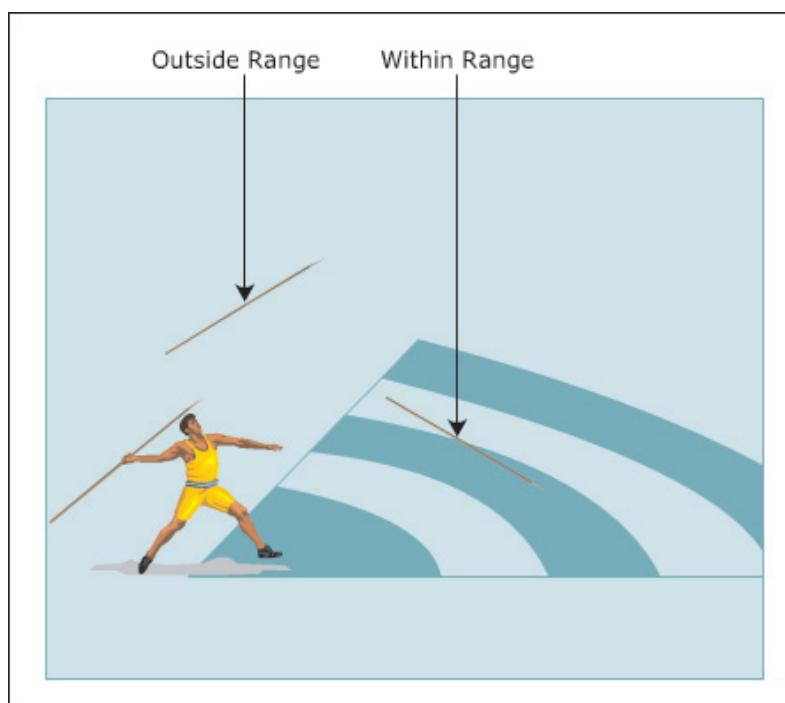
In the code, a TextBox control named txtConfirmPass is created to accept the input from the user. The control is given a unique identifier txtConfirmPass using the ID property and it is added to the form using the Add() method. A CompareValidator control named valConfirmPass is created and is assigned to the input TextBox control txtConfirmPass. The comparison operator for checking the equality of the values is assigned using the Operator property. The message to be displayed when the validation fails is specified using the Text property. The ValueToCompare property is assigned the value abcd. This value is checked against the value entered by the user in the TextBox control.

9.2.5 RangeValidator Control

Consider an online job application form in which, among other fields, is the field for accepting the applicant's age. To meet the eligibility criteria for the job, the applicant should be an adult below retirement age. That is, the age of the applicant should be between 18 and 60 years. To verify the applicant's eligibility in terms of his/her age, the `RangeValidator` control can be used.

The `RangeValidator` control provided by ASP.NET checks whether the value provided by the user is within a specified range. This control can validate numeric, character, and date values.

Figure 9.4 demonstrates the concept.



9.2.6 RangeValidator Class

The `RangeValidator` control checks whether or not entered values lie in specified ranges. The `RangeValidator` class represents the `RangeValidator` control on the ASP.NET Web page.

Following are the important properties of the `RangeValidator` class:

- `ErrorMessage`
- `IsValid`
- `MaximumValue`
- `MinimumValue`

→ Text

→ Type

These properties are described as follows:

→ **ErrorMessage**

The `ErrorMessage` property of the `RangeValidator` class specifies or retrieves the error message text to be displayed on the validation summary control when validation fails. It also displays the error message on the `RangeValidator` control when validation fails if the `Text` property for the control is not specified.

The following code demonstrates the use of the `ErrorMessage` property.

Code Snippet:

```
valAge.ErrorMessage = "Enter the appropriate age between 18 and 60.";
```

In this code, if the validation fails, the error message assigned to the `ErrorMessage` property of the `RangeValidator` validation control `valAge` is displayed.

→ **IsValid**

The `IsValid` property of the `RangeValidator` class specifies or retrieves a value indicating whether or not the input passes the validation criteria.

The following code demonstrates the use of the `IsValid` property.

Code Snippet:

```
if (!valPrice.IsValid)
{
    Response.Write("There is an error on the page.");
}
```

In this code, the message is displayed when input control related to the `RangeValidator` control, `valPrice`, fails the validation criteria.

→ **MaximumValue**

The `MaximumValue` property of the `RangeValidator` class specifies the maximum valid value for the input.

The following code demonstrates the use of the `MaximumValue` property.

Code Snippet:

```
valMarks.MaximumValue = "75";
```

In this code, the maximum valid value of the `RangeValidator` validation control, `valMarks`, is set as 75.

→ **MinimumValue**

The `MinimumValue` property of the `RangeValidator` class specifies the minimum valid value for the input.

The following code demonstrates the use of the `MinimumValue` property.

Code Snippet:

```
valAccountNo.MinimumValue = "1";
```

In this code, the minimum valid value of the `RangeValidator` validation control, `valAccountNo`, is set as 1.

→ **Text**

The `Text` property of the `RangeValidator` class specifies or retrieves the text to be displayed on the `RangeValidator` control when validation fails.

The following code demonstrates the use of the `Text` property.

Code Snippet:

```
valProductNo.Text = "Enter the product number.";
```

In this code, the text specified with the `Text` property of the `RangeValidator` validation control, `valProductNo`, is displayed when the validation fails.

→ **Type**

The `Type` property of the `RangeValidator` class specifies the data type of the value to be validated. The data type can be `Currency`, `Date`, `Double`, `Integer`, or `String`.

The following code demonstrates the use of the `Type` property.

Code Snippet:

```
valNumberOfItems.Type = ValidationDataType.Integer;
```

In this code, the `Type` property of the `RangeValidator` validation control, `valNumberOfItems`, specifies the data type of the checked value should be integer.

Figure 9.5 shows an example demonstrating the RangeValidator control.

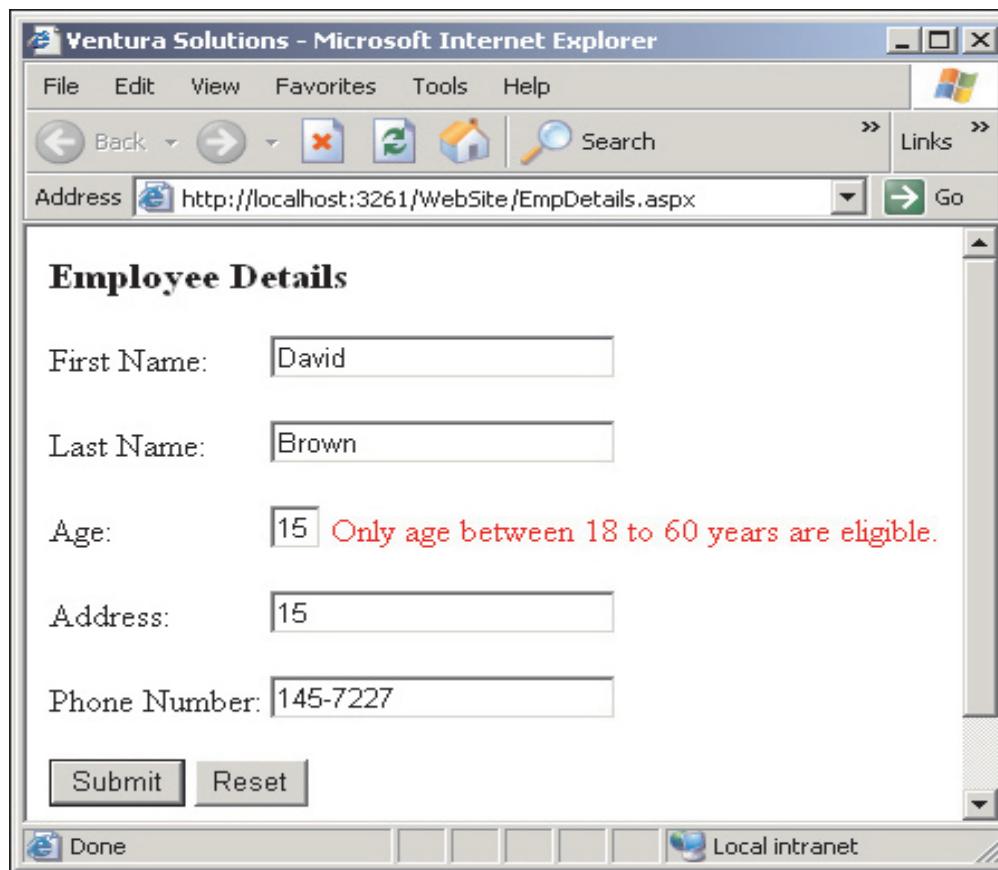


Figure 9.5: Example of RangeValidator Control

9.2.7 RegularExpressionValidator Control

Consider a Web application in which user's data such as credit card number has to be validated. Credit cards such as MasterCard have a standard prefix such as 51xxxx-55xxxx. To validate such a pattern, as a developer, you would have had to write lengthy logic. ASP.NET simplifies this task of validating patterns by providing the `RegularExpressionValidator` control.

The `RegularExpressionValidator` control checks for the validation of the value in the input control with a pattern of an expression. For example, the expression can have the pattern of a phone number, e-mail address, zip code and so on. The `RegularExpressionValidator` control can be used to check whether or not a given e-mail address is entered in the correct format or the phone number has the required number of digits.

9.2.8 Properties of the RegularExpressionValidator Class

The `RegularExpressionValidator` class represents the `RegularExpressionValidator` control on the ASP.NET Web page.

Some important properties of the `RegularExpressionValidator` class are as follows:

→ **ControlToValidate**

→ **Display**

→ **ErrorMessage**

→ **IsValid**

→ **ValidationExpression**

Figure 9.6 demonstrates the concept.

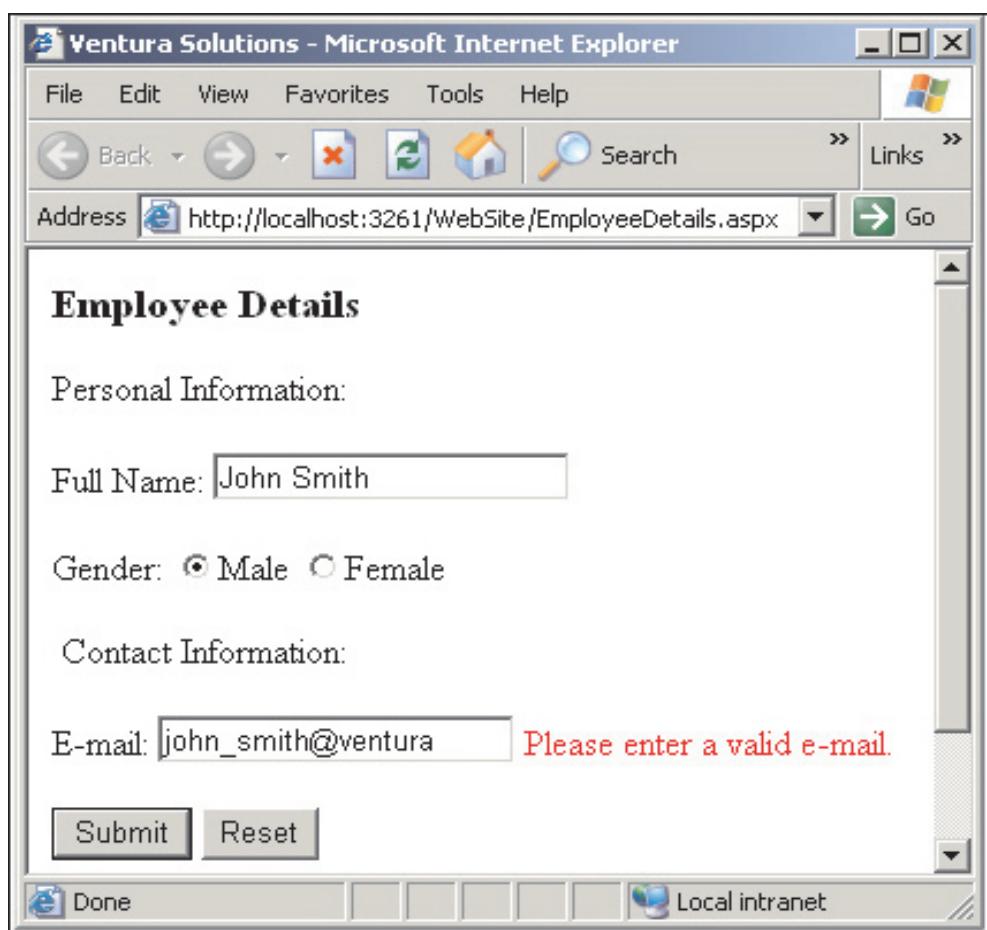


Figure 9.6: Example of RegularExpressionValidator Control

Properties are described as follows:

→ **ControlToValidate**

The **ControlToValidate** property of the **RegularExpressionValidator** class specifies or retrieves the **ID** of the input Web server control to be validated.

The following code demonstrates the use of the **ControlToValidate** property.

Code Snippet:

```
valPostalCode.ControlToValidate = "txtPostalCode";
```

In this code, the input TextBox control, txtPostalCode, is linked to the RegularExpressionValidator control named valPostalCode using the ControlToValidate property.

→ Display

The Display property of the RegularExpressionValidator class specifies or retrieves the behavior of the error message to be displayed in the validation control. The behavior can be either None, Static, or Dynamic.

The following code demonstrates the use of the ControlToValidate property.

Code Snippet:

```
valEmail.Display = ValidatorDisplay.Static;
```

In this code, the error message is displayed statically in the RegularExpressionValidator control, valEmail, using the Static and Display properties.

→ ErrorMessage

The ErrorMessage property of the RegularExpressionValidator class specifies or retrieves the error message text to be displayed when the validation fails.

The following code demonstrates the use of the ErrorMessage property.

Code Snippet:

```
valAuthorID.ErrorMessage = "Enter valid author ID.";
```

In this code, if the validation fails, the error message assigned to the ErrorMessage property of the RegularExpressionValidator validation control, valAuthorID, is displayed.

→ IsValid

The IsValid property of the RegularExpressionValidator class specifies or retrieves a value indicating whether or not the input passes the validation criteria.

The following code demonstrates the use of the IsValid property.

Code Snippet:

```
if (!valZipCode.IsValid)
{
    Response.Write("There is an error on the page.");
}
```

In this code, a message is displayed when input control associated with the RegularExpressionValidator control, valZipCode, does not pass the validation criteria.

→ ValidationExpression

The ValidationExpression property of the RegularExpressionValidator class specifies or retrieves the pattern of an expression against which the input has to be validated.

The following code demonstrates the use of the ValidationExpression property.

Code Snippet:

```
valPhoneNumber.ValidationExpression = "[0-9]{3} [0-9]{3}-[0-9]{4}";
```

In this code, the format of the data to be validated is assigned to the RegularExpressionValidator control, valPhoneNumber, using the ValidationExpression property.

9.2.9 CustomValidator control

The CustomValidator control checks the value of an input control to determine whether the value entered is valid or not according to the specified logic. The CustomValidator control returns the value as true when the value in the input control matches the criteria of the specified logic.

For example, you can use the CustomValidator control to check for even, odd, or prime numbers, and also to check whether or not the entered string value is a palindrome.

Figure 9.7 demonstrates the concept.

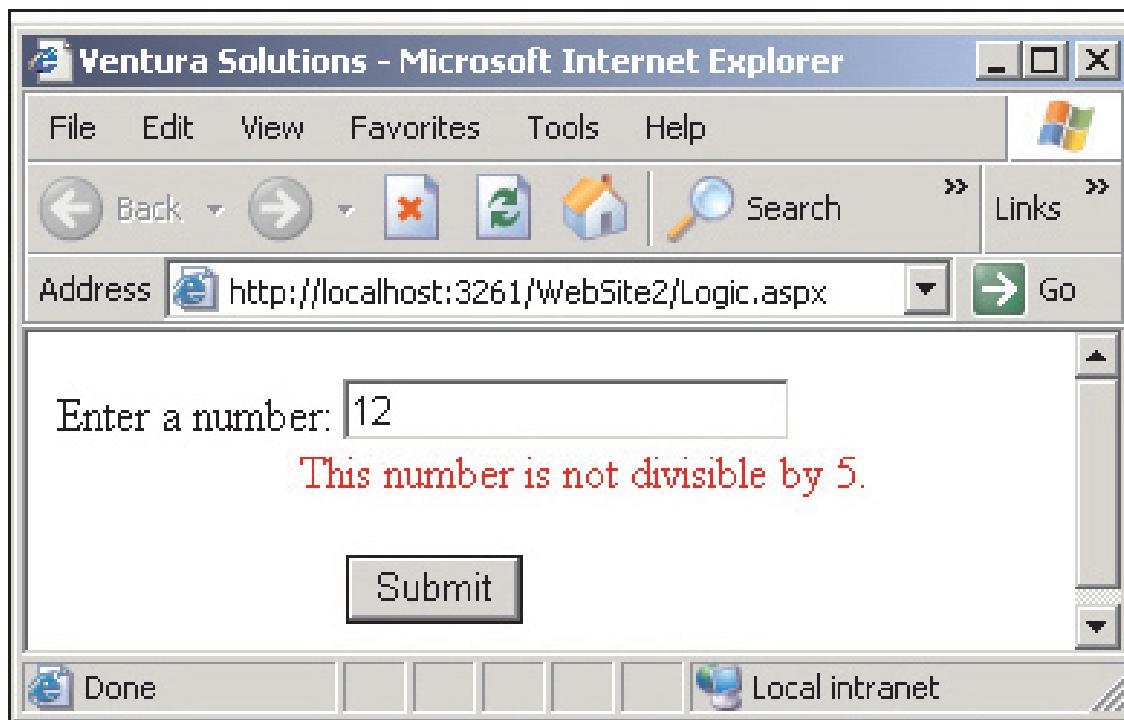


Figure 9.7: Example of Custom Validator Control

9.2.10 Properties of the CustomValidator Control

The `CustomValidator` control checks for the validity of the value in the input control by evaluating the specified logic.

The important properties of the `CustomValidator` class, which represents the `CustomValidator` control, are as follows:

→ **ClientValidationFunction**

The `ClientValidationFunction` property of the `CustomValidator` class specifies the client-side function name that is used to check for the validation.

The following code demonstrates the use of the `ClientValidationFunction` property.

Code Snippet:

```
valPinCode.ClientValidationFunction = "ShowMessage";
```

In this code, `ClientValidationFunction` property of the `CustomValidator` control is assigned to the client-enabled script function named `ShowMessage`.

→ **ControlToValidate**

The `ControlToValidate` property of the `CustomValidator` class specifies or retrieves the identifier of the input Web server control to be validated.]

The following code demonstrates the use of the `ControlToValidate` property.

Code Snippet:

```
valProductID.ControlToValidate = "txtProductID";
```

In this code, the input `TextBox` control, `txtProductID`, is linked to the `RegularExpressionValidator` control, `valPostalCode`, using the `ControlToValidate` property.

→ **ErrorMessage**

The `ErrorMessage` property of the `CustomValidator` class specifies or retrieves the text for the error message to be displayed when the validation is not performed.

The following code demonstrates the use of the `ErrorMessage` property.

Code Snippet:

```
valQuantity.ErrorMessage = "Enter the valid quantity.";
```

In this code, if the validation fails, the error message assigned to the `ErrorMessage` property of the `CustomValidator` validation control, `valQuantity`, is displayed.

→ ValidateEmptyText

The `ValidateEmptyText` property of the `CustomValidator` class specifies or retrieves a Boolean value that indicates whether or not blank text should be validated. The `CustomValidator` class specifies or retrieves the value as `true` if the blank text is to be validated; otherwise it specifies or retrieves the value as `false`.

The following code demonstrates the use of the `ValidateEmptyText` property.

Code Snippet:

```
valLastName.ValidateEmptyText = true;
```

In this code, the `ValidateEmptyText` property of the `CustomValidator` control, `valLastName`, is set to `true`, indicating that empty text should be validated.

Figure 9.8 shows an example demonstrating the properties of the `CustomValidator` control.

```
protected void Page_Load(object sender, EventArgs e)
{
    TextBox txtNumber = new TextBox();
    txtNumber.ID = "txtNumber";
    CustomValidator valDivisibleByFive =
        new CustomValidator();
    valDivisibleByFive.ControlToValidate = "txtNumber";
    valDivisibleByFive.ErrorMessage =
        "Number must be divisible by 5";
    valDivisibleByFive.ClientValidationFunction =
        "DivisibleByFive";
    valDivisibleByFive.ValidateEmptyText = true;
    frmLogic.Controls.Add(txtNumber);
    frmLogic.Controls.Add(valDivisibleByFive);
}
```

Figure 9.8: Example for Properties of `CustomValidator` Class

9.2.11 ValidationSummary Control

ValidationSummary control allows reviewing error messages from all the validation controls on the Web page. The ValidationSummary control summarizes and displays all the validation error messages on a single page.

For example, during the employee's registration, ValidationSummary can display a list of error messages such as when a control is left blank or left unselected by the user.

You must set the Text property of the other validation controls to properly work with the ValidationSummary control. The ValidationSummary control displays the error message based on the provided Text property. If you do not set the Text property of the other validation controls, the ValidationSummary control does not display proper error messages when any validation error occurs.

Figure 9.9 demonstrates the use of this control.

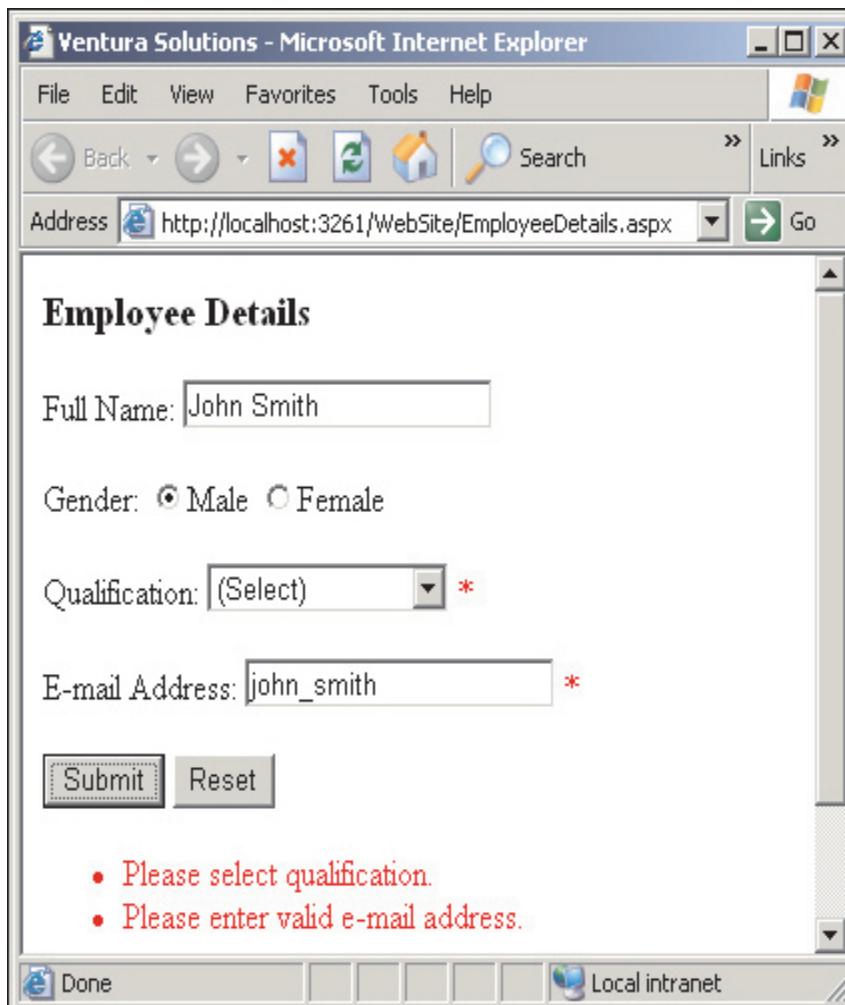


Figure 9.9: Example of Validation Summary Control

9.2.12 Properties of the ValidationSummary control

The ValidationSummary control is used to display a list of error messages for errors occurring in other validation controls. The ValidationSummary class represents the ValidationSummary control on the ASP.NET Web page.

The important properties of the ValidationSummary class are as follows:

- ➔ **DisplayMode**
- ➔ **EnableClientScript**
- ➔ **HeaderText**
- ➔ **ShowMessageBox**
- ➔ **ShowSummary**
- ➔ **DisplayMode**

The DisplayMode property of the ValidationSummary class specifies or retrieves the appearance mode of the validation summary. The validation summary can be displayed in list, bulleted list, or paragraph modes.

The following code demonstrates the use of the DisplayMode property.

Code Snippet:

```
valsCustomerDetails.DisplayMode = ValidationSummaryDisplayMode.  
BulletList;
```

In this code, the error messages in the ValidationSummary control, valsCustomerDetails are displayed in the form of a bulleted list using the DisplayMode property.

- ➔ **EnableClientScript**

The EnableClientScript property of the ValidationSummary class specifies or retrieves a value which is used to update the ValidationSummary control using client-side script.

The following code demonstrates the use of the EnableClientScript property.

Code Snippet:

```
valsEmployeeDetails.EnableClientScript = false;
```

In this code, the EnableClientScript property of the ValidationSummary control, valsEmployeeDetails, is set to false to prevent access to client-scripting.

→ HeaderText

The `HeaderText` property of the `ValidationSummary` class specifies or retrieves the text that is to be displayed as the heading for the validation summary.

The following code demonstrates the use of the `HeaderText` property.

Code Snippet:

```
valsProductDetails.HeaderText = "List of errors";
```

In this code, the header text for the `ValidationSummary` control, `valsProductDetails`, is assigned using the `HeaderText` property.

→ ShowMessageBox

The `ShowMessageBox` property of the `ValidationSummary` class specifies or retrieves a Boolean value indicating whether the validation summary is displayed in a message box or not. When the `ShowMessageBox` property is set to `true`, it indicates that the validation summary is displayed in a message box. The value of the `ShowMessageBox` property is set to `false` by default.

The following code demonstrates the use of the `ShowMessageBox` property.

Code Snippet:

```
valsBankInfo.ShowMessageBox = false;
```

In this code, the `ShowMessageBox` property of the `ValidationSummary` control, `valsBankInfo`, is set to `false`, which indicates that the `ValidationSummary` control will not be displayed in a message box.

→ ShowSummary

The `ShowSummary` property of the `ValidationSummary` class specifies or retrieves a value that indicates whether the validation summary is displayed in inline mode or not.

The following code demonstrates the use of the `ShowSummary` property.

Code Snippet:

```
valsShopping.ShowSummary = true;
```

In this code, the `ShowSummary` property of the `ValidationSummary` control, `valsShopping`, is set to `true`, which indicates that the validation summary is displayed in the inline mode.

Figure 9.10 shows an example demonstrating the properties of the ValidationSummary class.

```
protected void Page_Load(object sender, EventArgs e)
{
    ValidationSummary valsShopping = new ValidationSummary();
    valsShopping.DisplayMode =
        ValidationSummaryDisplayMode.BulletList;
    valsShopping.EnableClientScript = false;
    valsShopping.HeaderText = "List of Errors";
    valsShopping.ShowMessageBox = false;
    valsShopping.ShowSummary = true;

    frmShopping.Controls.Add(valsShopping);
}
```

Figure 9.10: Example for Properties of ValidationSummary Class

9.2.13 Page.IsValid Property

The `Page.IsValid` property is included in the `Page` class of the `System.Web.UI` namespace. The `Page.IsValid` property retrieves a value which indicates whether or not the validation of the page has been successful. The `Page.IsValid` property retrieves the value as `true` when all the validation controls within the page are successfully validated.

Based on the value returned by the `Page.IsValid` property, you can decide whether or not to proceed with post backs to the server. If the value returned is `true`, you can proceed with the post back. However, if the value returned is `false`, it means all validation controls have not been successfully validated and hence, post back to the server can be halted using programming logic. This provides additional security for the Web site.

Figure 9.11 demonstrates the concept.

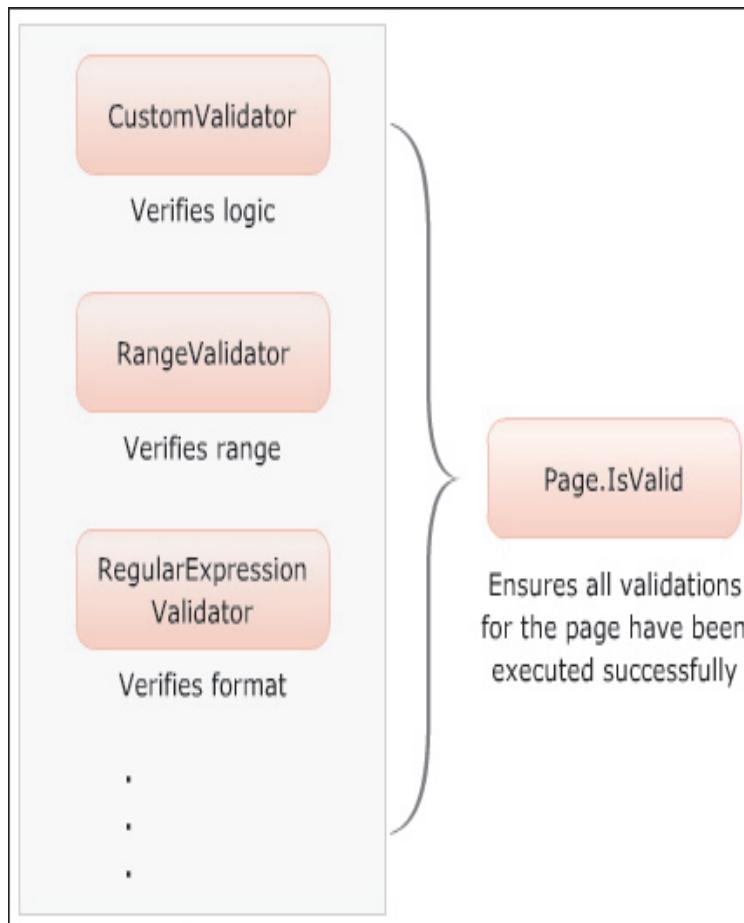


Figure 9.11: Page.IsValid Property

Knowledge Check 2

1. Which of the following statements about validation controls are false?

(A)	The CompareValidator control compares the value of one input control with the other input control based on the associated comparison operator.
(B)	The Operator property of the CompareValidator control specifies or retrieves the operator to perform the arithmetic operation.
(C)	The MaximumValue property of the RangeValidator class specifies the maximum range value to be validated with the value in the input control.
(D)	The ValidationExpression property of the RegularExpressionValidator class specifies or retrieves a value indicating whether the related input control passes the validation process.
(E)	The Page.IsValid property retrieves a value which indicates whether or not the validation of the page has been successful.

(A)	A, B, D	(C)	B, D
(B)	A, C	(D)	C, D, E

2. You want to create a Web page to register login information. This page should accept the user name, which cannot be left blank, a password, and a copy of the password, which should be the same as the password. Which one of the following codes will help you achieve this?

(A)	<pre> TextBox txtUserName = new TextBox () ; txtUserName.ID = "txtUserName" ; TextBox txtPassword = new TextBox () ; txtPassword.ID = "txtPassword" ; TextBox txtConfirmPassword = new TextBox () ; txtConfirmPassword.ID = "txtConfirmPassword" ; ... RequiredFieldValidator valUserName = new RequiredFieldValidator; CompareValidator valPassword = new CompareValidator () ; valUserName.ControlToValidate = "txtUserName" ; valUserName.ErrorMessage = "Enter the user name." ; valPassword.ControlToValidate = "txtConfirmPassword" ; valPassword.ControlToCompare = "txtPassword" ; valPassword.ErrorMessage = "Enter the correct password." ; </pre>
-----	--

(B)	<pre>TextBox txtUserName = new TextBox(); txtUserName.ID = "txtUserName"; TextBox txtPassword = new TextBox(); txtPassword.ID = "txtPassword"; TextBox txtConfirmPassword = new TextBox(); txtConfirmPassword.ID = "txtConfirmPassword"; ... RequiredFieldValidator valUserName = new RequiredFieldValidator(); CompareValidator valPassword = new CompareValidator(); valUserName.ControlToValidate = "txtUserName"; valUserName.ErrorMessage = "Enter the user name."; valPassword.ControlToValidate = "txtConfirmPassword"; valPassword.ControlToCompare = "txtPassword"; valPassword.ErrorMessage = "Enter the correct password.";</pre>
(C)	<pre>TextBox txtUserName = new TextBox(); txtUserName.ID = "txtUserName"; TextBox txtPassword = new TextBox(); txtPassword.ID = "txtPassword"; TextBox txtConfirmPassword = new TextBox(); txtConfirmPassword.ID = "txtConfirmPassword"; ... RequiredFieldValidator valUserName = new RequiredFieldValidator(); CompareValidator valPassword = new CompareValidator(); valUserName.controlToValidate = "txtUserName"; valUserName.ErrorMessage = "Enter the user name."; valPassword.controlToValidate = "txtConfirmPassword"; valPassword.controlToCompare = "txtPassword"; valPassword.ErrorMessage = "Enter the correct password.";</pre>

...

```
TextBox txtUserName = new TextBox();
txtUserName.ID = "txtUserName";
TextBox txtPassword = new TextBox();
txtPassword.ID = "txtPassword";
TextBox txtConfirmPassword = new TextBox();
txtConfirmPassword.ID = "txtConfirmPassword";
...
```

(D) RequiredFieldValidator valUserName = new
RequiredFieldValidator();
CompareValidator valPassword = new CompareValidator();
valUserName.ControlToValidate = txtUserName;
valUserName.ErrorMessage = "Enter the user name.";
valPassword.ControlToValidate = txtConfirmPassword;
valPassword.ControlToCompare = txtPassword;
valPassword.ErrorMessage = "Enter the correct password.";

...

(A)	B	(C)	D
(B)	A	(D)	C

Module Summary

In this module, **Validation Controls**, you learnt about:

→ **Validation**

The validation controls ensure that data provided lies within the specified restrictions. This helps in smooth functioning of the ASP.NET Web application. ASP.NET provides various validation controls, some of which are `CompareValidator`, `CustomValidator`, and `RangeValidator`.

→ **Validation Controls in ASP.NET**

ASP.NET provides various validation controls such as `CompareValidator`, `RangeValidator`, `RequiredFieldValidator`, and `ValidationSummary`. The classes representing these controls lie in the `System.Web.UI.WebControls` namespace. The validation controls are used with their different properties, methods and events for controlling their behavior.

Module - 10

Application, Session, and Cookies

Welcome to the module, **Application, Session, and Cookies**.

The module explores Application and Session objects and also how to use cookies. The Application object, stores data shared across the application. A session is the time period between the start and end of the user's interaction with an application. Cookies are files used by a Web application to store user-related information.

In this module, you will learn to:

- ➔ Application object
- ➔ Cookies
- ➔ Session state
- ➔ Global.asax

Web Development

http://www



10.1 Application Object

In this first lesson, **Application Object**, you will learn to:

- Define Application objects and describe the `HttpApplicationState` class.
- Describe the use of `Lock` and `UnLock` methods of `HttpApplicationState` class.
- Explain the use of `Add`, `Clear`, `Remove`, `RemoveAll`, `RemoveAt` methods.

10.1.1 Application Object

Application objects are used to reference instances of `HttpApplicationState` class. An application state stores global information used across multiple sessions and requests, unlike a session state, which only stores information specific to a single user session.

The Application object generally holds information that will be used by multiple pages of the application. This object allows information to be accessed from any page. Any change made to the information is thus, automatically reflected in all the affected pages.

Figure 10.1 demonstrates the concept.

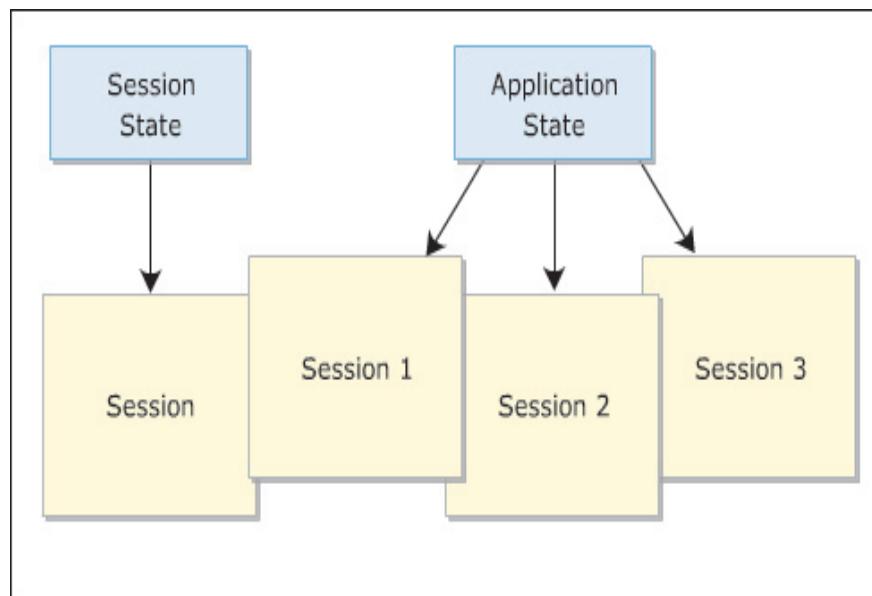


Figure 10.1: Session and Application State

10.1.2 *HttpApplicationState Class*

Application object in ASP.NET 2.0 is represented by `HttpApplicationState` class. `HttpApplicationState` class belongs to the `System.Web` namespace. This class allows sharing global information across various sessions and requests within an ASP.NET application.

The first time a client requests a URL resource from within an ASP.NET application virtual directory, an instance of the `HttpApplicationState` class is created. A reference to this instance is exposed using the `Application` object. A separate instance of the `HttpApplicationState` class is created for each ASP.NET application on the server.

10.1.3 *Properties of HttpApplicationState Class*

object exposes the properties of `HttpApplicationState` class. Some important properties of the `HttpApplicationState` class are as follows:

- ➔ AllKeys
- ➔ Contents
- ➔ Count
- ➔ Item
- ➔ StaticObjects

Figure 10.2 shows some of the properties of the `HttpApplicationState` control.

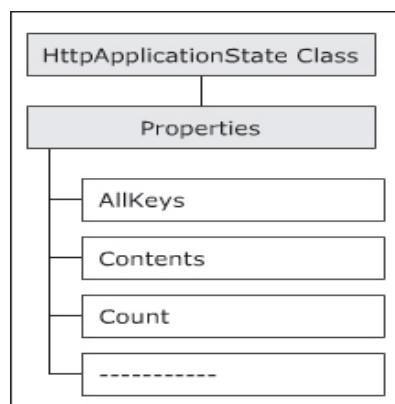


Figure 10.2: Properties of `HttpApplicationState` Class

These are described as follows:

➔ **AllKeys**

The `AllKeys` property retrieves the access keys from the application state collection.

The following code demonstrates the use of the `AllKeys` property.

Code Snippet:

```
String [] strKeys = new String [ Application.Count ];
strKeys = Application.AllKeys;
```

→ **Contents**

The `Contents` property retrieves the reference to `HttpApplicationState` object.

In this code, a string array `strKeys` is filled with all the object names in the application state collection.

The following code demonstrates the use of the `Contents` property.

Code Snippet:

```
protected void Page_Load(object sender, EventArgs e)
{
    Application.Contents.RemoveAll();
    Response.Write("Removed all items from current Application");
}
```

In this code, `RemoveAll()` method removes all the `Application` objects.

→ **Count**

The `Count` property counts and retrieves the number of objects. By default, the value is 0.

The following code demonstrates the use of the `Count` property.

Code Snippet:

```
Response.Write("Application collection contains " + Application.Count +
variables);
```

In this code, the `Count` property retrieves the number of objects in the application state collection and this number is displayed using the `Response.Write()` method.

→ **Item**

The `Item` property provides access to individual objects in an `HttpApplicationState` collection.

The following code demonstrates the use of the `Item` property. The syntax for using the `Item` property does not require the `Item` keyword. The `Application` variable specified in square brackets represents the item.

Code Snippet:

```
Application["UserName"] = "John";
Response.Write("UserName is :" + Application[0]);
```

In this code, the `Application` variable `UserName` is assigned the value `John`. The index value is `0` and this value is used to provide access to the first `Application` object.

→ **StaticObjects**

The `StaticObjects` property retrieves all objects declared using the `<object>` tag where scope is defined as “`Application`”.

The following code demonstrates the use of the `StaticObjects` property.

Code Snippet:

```
if (Application.StaticObjects.Count > 0)
{
    Response.Write("There are one or more objects in the current application.");
}
```

In this code, the `if` condition checks whether or not the number of `<object>` tags used in a `Global.asax` file is greater than `0`. This is done by using the `Count` property of the `StaticObjects` property. If the number of `<object>` tags is greater than `0`, the `Response.Write()` method displays the message `There are one or more objects in the current application.`.

10.1.4 Lock Method

Consider the railway reservation system where reservations for a ticket can be made across any reservation counter on the network. If several people are simultaneously booking tickets for the same date on the same train, there could be confusion regarding the actual availability status of the desired tickets. To avoid this, if a counter is issuing tickets for a specific date on a specific train, these tickets should not be available for issuing at any other counter. Once the first counter has finished the processing, the tickets can be made available for issuing at the next counter.

Similarly, when multiple users are accessing an ASP.NET application, they should not be allowed to simultaneously make changes to the same data. This can be achieved by locking the data variables. The `Lock()` method, a member of `HttpApplicationState` class, prevents other users from altering the variables stored in the `Application` object.

Figure 10.3 demonstrates the concept.

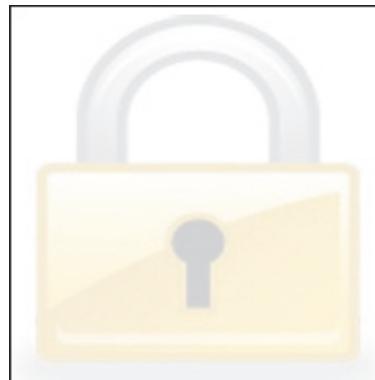


Figure 10.3: Lock() Method

The syntax for the Lock method is:

Syntax:

```
public void Lock ()
```

10.1.5 *UnLock* Method

The UnLock method, a member of the `HttpApplicationState` class, is used to unlock the locked variables stored in the `Application` object. Once variables have been unlocked, other users are able to modify them.

When you call `Lock()` on an `Application` object, it causes ASP.NET to block attempts by code running on other worker threads to access anything in application state. These threads are unblocked only when the thread which invoked `Lock()` calls the corresponding `UnLock()` method on the `Application` object.

If you do not explicitly call the `UnLock()` method, the .NET Framework automatically removes the lock either when the request completes, or when the request times out, or when an unhandled exception occurs causing the request to fail.

Figure 10.4 demonstrates the concept.

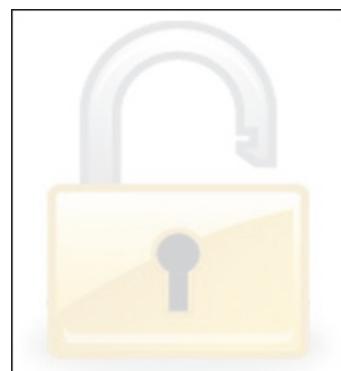


Figure 10.4: UnLock() Method

The syntax for the `UnLock` method is:

Syntax:

```
public void UnLock ()
```

The following code demonstrates the use of the `Lock()` and `UnLock()` methods.

Code Snippet:

```
Application.Lock ();
Application["VisitorsCount"] = (int)Application["VisitorsCount"] + 1;
Application.UnLock();
```

In this code, `Lock()` method prevents other sessions from changing the value of the application variable `VisitorsCount`. When the current session starts, the value of the variable, `VisitorsCount`, is incremented by 1. The `UnLock()` method unlocks the variable.

10.1.6 Methods of `HttpApplicationState` Class

The removal methods of `HttpApplicationState` class are used to insert and delete objects in the application state collection. The removal methods can delete all or only a particular object from the collection.

Some of the important removal methods of `HttpApplicationState` class are as follows:

- ➔ Add
- ➔ Clear
- ➔ Remove
- ➔ RemoveAll and RemoveAt

Figure 10.5 demonstrates the concept.

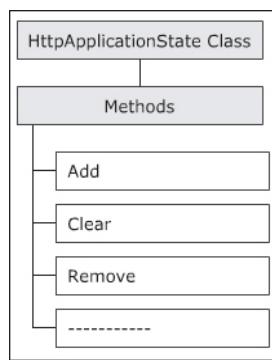


Figure 10.5: `HttpApplicationState` Class Methods

→ **Add**

The `Add` method adds a new object to the application state collection.

The syntax of the `Add` method is:

Syntax:

```
public void Add (string name, Object value)
```

where,

name: Is the name of the object that is to be added to the collection.

value: Is the value of the object.

The following code demonstrates the use of the `Add()` method.

Code Snippet:

```
Application.Add("ProductID", "E101");
```

In this code, the `Add()` method adds a variable named `ProductID` having value `E101` to application state collection.

→ **Clear**

The `Clear` method removes all the objects of the application state collection.

The syntax of the `Clear` method is:

Syntax:

```
public void Clear ()
```

The following code demonstrates the use of the `Clear()` method.

Code Snippet:

```
Application.Clear();
```

In this code, the `Clear()` method clears the variables of the application state collection.

→ **Remove**

The `Remove` method deletes the specified object of the application state collection.

The syntax of the `Remove` method is:

```
public void Remove (string name)
```

where,

name: Is the name of the object that is to be removed from the collection.

Code Snippet:

```
Application.Remove("ProductID");
```

In this code, the Remove () method removes the application variable ProductID.

→ RemoveAll

The RemoveAll method deletes all objects of the application state collection.

The syntax of the RemoveAll method is:

Syntax:

```
public void RemoveAll ()
```

The following code demonstrates the use of the RemoveAll () method.

Code Snippet:

```
Application.RemoveAll();
```

In this code, RemoveAll () method removes all the Application objects.

→ RemoveAt

The RemoveAt method deletes the object at the specified index position.

The syntax of the RemoveAt method is:

```
public void RemoveAt (int index)
```

where,

index: Is the position from which the item is removed from the collection.

Code Snippet:

```
Application.RemoveAt(0);
```

In this code, the RemoveAt () method removes the application variable at index 0.

Knowledge Check 1

1. Which of the following statements about Application objects are false?

(A)	The Remove () method deletes all the objects of the application state collection.
(B)	The UnLock property unlocks the locked variable.
(C)	The Lock () method prevents other sessions from changing the value of the application variable.
(D)	The Application object generally holds information that will be used by multiple pages of the application.
(E)	The Application object is represented by HttpApplication class.

(A)	A, B, E	(C)	B, D
(B)	A, C	(D)	C, D, E

2. Can you match the properties and methods of HttpSessionState class with their corresponding description?

Description		Property and Method	
(A)	Deletes the object at the specified index location.	(1)	AllKeys
(B)	Clears all objects of the application state collection.	(2)	RemoveAt
(C)	Allows access to individual objects.	(3)	Lock
(D)	Retrieves the access keys from the application state collection	(4)	Clear
(E)	Prevents other users from modifying the variables.	(5)	Item

(A)	(A)-(2), (B)-(4), (C)-(5), (D)-(1), (E)-(3)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(5), (C)-(1), (D)-(4), (E)-(2)	(D)	(A)-(4), (B)-(1), (C)-(2), (D)-(5), (E)-(3)

10.2 Cookies

In this second lesson, **Cookies**, you will learn to:

- Define session cookies and describe how to create and read them.
- Define persistent cookies and describe how to use them.

10.2.1 Creating and Reading **Session Cookies**

Consider a simple e-mail client such as Yahoo Mail. To log on to your mail account in Yahoo, you need to enter the user name and the password. Once you log on, a session is started.

A session can be defined as the time period between the start and the end of the user's interaction with an application. The session information such as login and logout times and pages visited is stored in temporary files called session cookies.

A cookie is a temporary or permanent file used by a Web application to store user-related information. Session cookies have a certain time limit. Once the time limit is reached, the user needs to re-login.

Figure 10.6 demonstrates the concept.

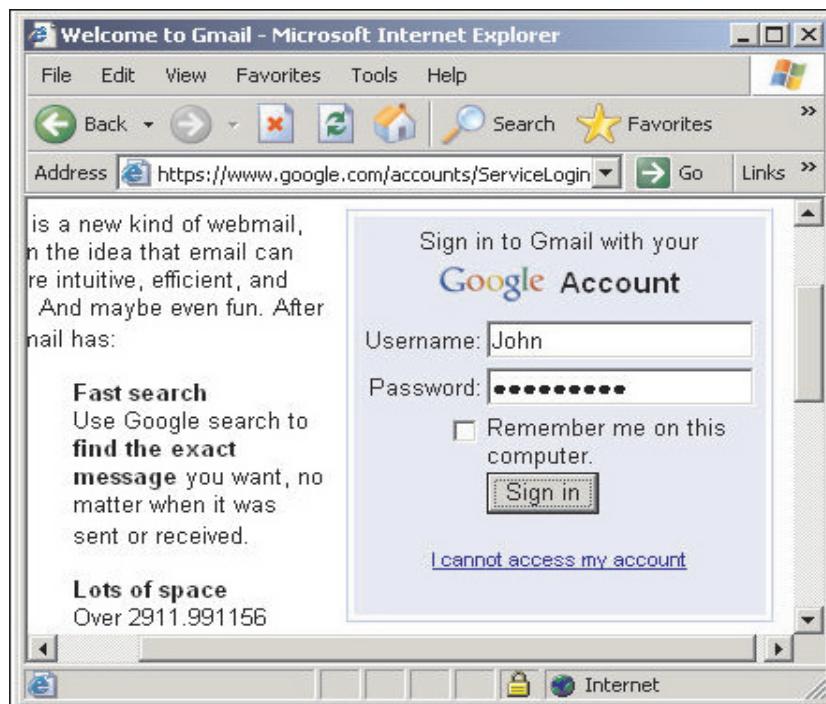


Figure 10.6: Example of Session Cookies

10.2.2 Creating and Reading Session Cookies

A session cookie is also referred to as a transient cookie. Session cookies are stored temporarily in the memory. Once the browser is closed, the session cookie cannot be retained. The next time when the same user visits the same site, he/she will be treated as a new visitor.

Instead of collecting information from the user's computer, session cookies are generally used by those Web applications in which users need to be identified as they move from page to page. For example, if you are a member of an online book library, once you have logged in, you can browse through any number of books. The session cookie identifies you as a valid subscriber as you move from one Web page to another across the application.

Figure 10.7 shows an example of using session cookies.

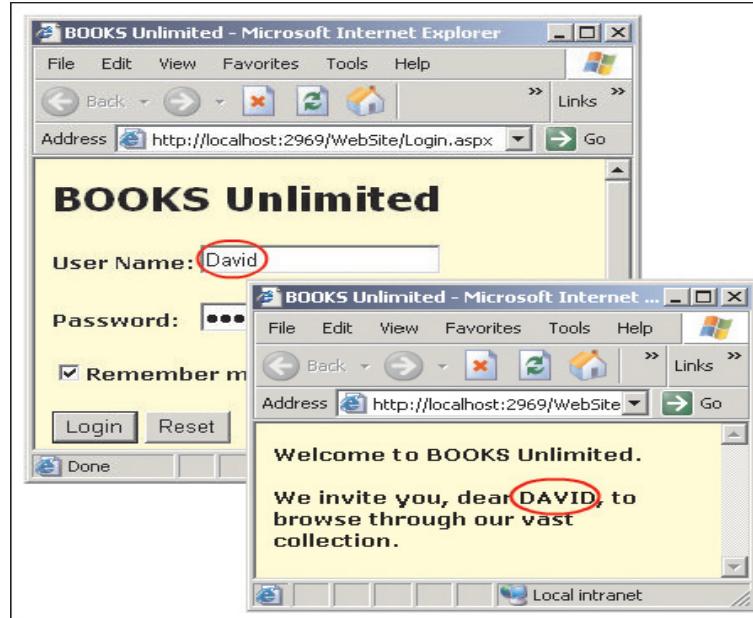


Figure 10.7: Example of Session Cookies

The following code demonstrates how to create and read a session cookie.

Code Snippet:

```
protected void btnAdd_Click(object sender, EventArgs e)
{
    Response.Cookies["Login"]["UserName"] = txtUserName.Text;
}

protected void btnView_Click(object sender, EventArgs e)
{
    if (Request.Cookies["Login"] == null)
    {
        Response.Write("No cookie.");
    }
    else
    {
        Response.Write("Cookie information: " + Request.Cookies["Login"]["UserName"]);
    }
}
```

In this code, `txtUserName` is an object of `TextBox` class. A cookie named `Login` and subkey named `UserName` are set to store the user name entered in the text box. When the `Add` button is clicked, the `btnAdd_Click` event is triggered and the value is added to the cookies collection. When the `View` button is clicked, `bntView_Click` event is triggered. If the cookie does not have any information, then the message `No cookie` is displayed on the Web page. Otherwise, the information stored in the cookie is displayed on the Web page.

10.2.3 Persistent Cookies

Consider a login page of any popular e-mail services provider such as Hotmail or Yahoo Mail. Once you enter your user name and password, you are prompted to indicate if you want your login details to be remembered on the computer. If you say yes, then these details are stored in a cookie. The next time you start to log on, these details automatically appear in their respective places.

Cookies storing such information that is remembered across multiple sessions are referred to as persistent cookies.

Persistent cookies are also known as permanent cookies or stored cookies. A persistent cookie can be used to track a user's browsing habits. Persistent cookies have an expiry date and are stored on to user's hard disk until the expiry date or until they are manually deleted.

Figure 10.8 demonstrates the concept.

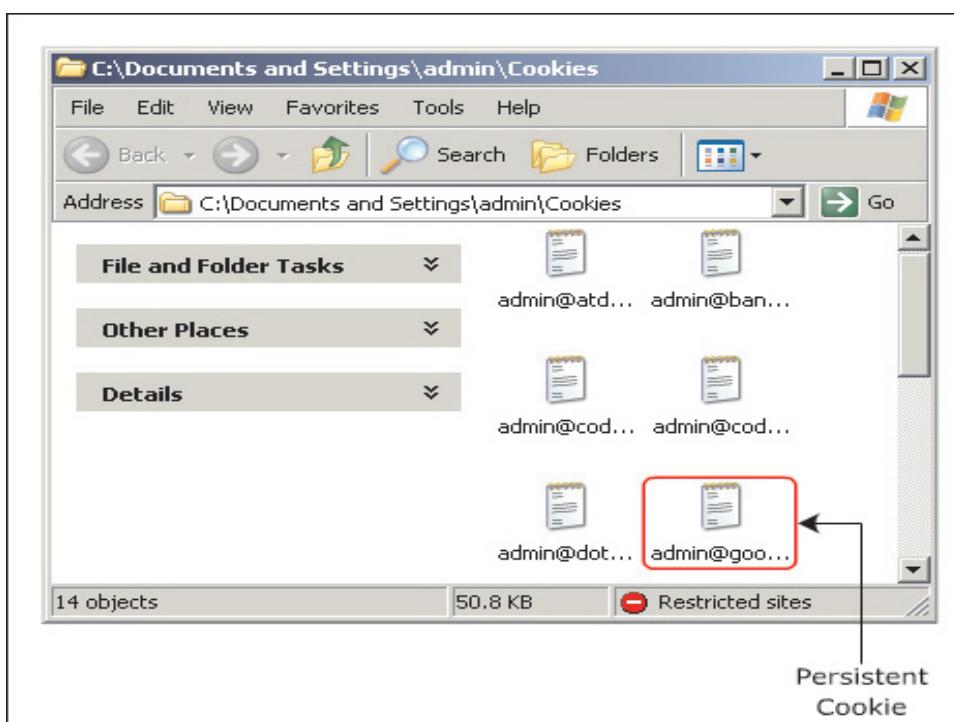


Figure 10.8: Persistent Cookies Example

The following code demonstrates how to create a persistent cookie.

Code Snippet:

```
HttpCookie userInfoCookie = new HttpCookie("UserInfo");
userInfoCookie.Values["UserName"] = "John";
userInfoCookie.Values["LastVisit"] = DateTime.Now.ToString();
userInfoCookie.Expires = DateTime.MaxValue;
Response.Cookies.Add(userInfoCookie);
```

In this code, an object of `HttpCookie` class represents a cookie named `UserInfo`. `UserName` is the subkey of `UserInfo` cookie. The `UserName` subkey's value is set to `John`. `LastVisit`, the second subkey of `UserInfo` cookie, is assigned the current date and time using `Now` property of `DateTime` class. The `Expires` property is assigned the maximum date using the `DateTime.MaxValue` field.

Knowledge Check 2

1. Which of the following statements about session and persistent cookies are false?

(A)	A session cookie is also referred to as a stored cookie.
(B)	A session cookie is retained after the browser is closed.
(C)	Persistent cookies are referred to as permanent cookies.
(D)	Persistent cookies have an expiry date.
(E)	A cookie is a temporary or permanent file used by a Web application to store user-related information.

(A)	A, B, E	(C)	A, B
(B)	A, C	(D)	C, D, E

2. Which one of the following codes helps you to add a cookie containing login information such as user name and password along with the expiry date 31st December, 2007?

(A)	<pre>HttpCookie loginCookie = new HttpCookie("Login"); loginCookie.Values["UserName"] = txtUserName.Text; loginCookie.Values["Password"] = txtPassword.Text; DateTime dtExpiry = new DateTime(2007, 12, 31); loginCookie.Expires = dtExpiry; Request.Cookies.Add(loginCookie);</pre>
-----	--

(B)	<pre>HttpCookie loginCookie = new HttpCookie("Login"); loginCookie.Values["UserName"] = txtUserName.Text; loginCookie.Values["Password"] = txtPassword.Text; DateTime dtExpiry = new DateTime(2007, 12, 31); loginCookie.Expires = dtExpiry; Request.Cookies.Add(loginCookie);</pre>
(C)	<pre>HttpCookie loginCookie = new HttpCookie("Login"); loginCookie.Values["UserName"] = txtUserName.Text; loginCookie.Values["Password"] = txtPassword.Text; DateTime dtExpiry = new DateTime(2007, 12, 31); loginCookie.Expires = dtExpiry; Request.Cookies.Add(loginCookie);</pre>
(D)	<pre>HttpCookie loginCookie = new HttpCookie("Login"); loginCookie.Values["UserName"] = txtUserName.Text; loginCookie.Values["Password"] = txtPassword.Text; DateTime dtExpiry = new DateTime(2007, 12, 31); loginCookie.Expires = "dtExpiry"; Request.Cookies.Add(loginCookie);</pre>

(A)	B	(C)	C
(B)	A	(D)	D

10.3 Session State

In this third lesson, **Session State**, you will learn to:

- Define **session** variables and describe their purpose and how to use them.
- Explain the session events.
- Describe session identifiers.

10.3.1 Session Variables

Session variables are used to store information about a single user session. This information is available to all pages in the application.

Generally, the information stored in session variables is the user name, password, user preferences, and so on. Session variables are cleared as soon as the user's session at the site comes to an end.

Session variables are stored in the `SessionStateItemCollection` class present in the `System.Web.SessionState` namespace. These variables are exposed through the `System.Web.HttpContext.Session` property.

Figure 10.9 demonstrates the concept.

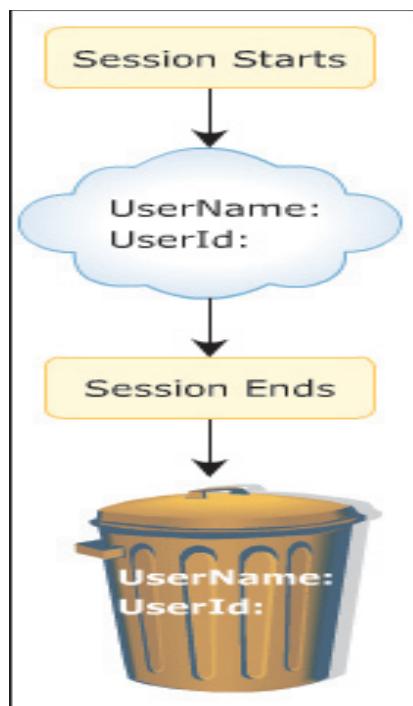


Figure 10.9: Session Concept

The following code demonstrates how to set session variables.

Code Snippet:

```

Session["UserID"] = "101";
if(Session["UserID"] == null)
{
    Response.Write("Page has expired.");
}
  
```

In this code, `UserID` is the session variable which is assigned the value `101`. If the session variable is null, `Page has expired` message is displayed on the Web Page.

10.3.2 Session-State Events

In a Web application, it is necessary to keep a track of and manage the user session. ASP.NET 2.0 provides two events to manage the user sessions.

They are as follows:

- Session _ OnStart Event
- Session _ OnEnd Event

Figure 10.10 demonstrates the depicts these events..

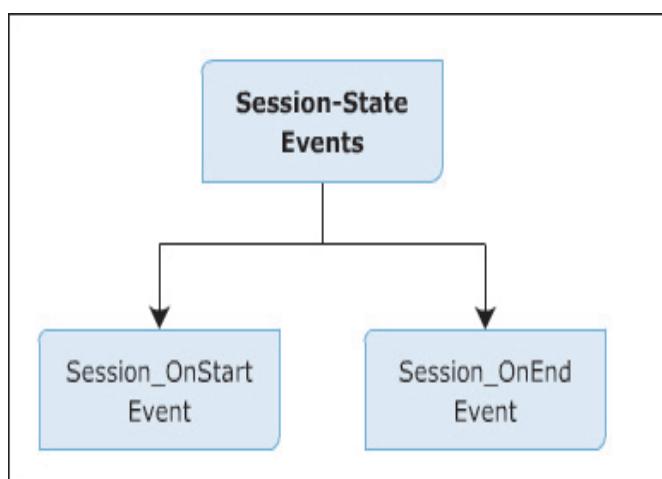


Figure 10.10: Session State Events

→ Session _ OnStart Event

`Session_OnStart` event is handled by adding a subroutine named `Session_OnStart` to the `Global.asax` file. This event is fired at the start of a new session. A new session starts if the request references a session having a `SessionID` property that has already expired.

The following code demonstrates the use of `Session_OnStart` event.

Code Snippet:

```

public void Session_OnStart()
{
    Application["UsersCount"] = (int)Application["UsersCount"] + 1;
    Response.Write("Users: " + Application["UsersCount"]);
}
  
```

In this code, the `Session_OnStart` event is triggered at the start of the session. This event is used to increment the `UsersCount` variable by 1. The number of users, as indicated by the `UsersCount` variable is displayed on the Web Page using the `Response.Write()` method.

→ Session _ OnEnd Event

Session _OnEnd event is handled by adding a subroutine named Session _OnEnd to the Global.asax file. This event is fired when a session times out. Session _OnEnd event can be used for executing clean-up code or for dumping data into the database.

The following code demonstrates the use of Session _OnEnd event.

Code Snippet:

```
public void Session_OnEnd()
{
    Response.Write("The current session has expired.");
}
```

In this code, the Session _OnEnd event is triggered at the end of the session and the message The current session has expired is displayed on the Web Page using the Response .Write () method.

10.3.3 Session Identifiers

Sessions are identified by a session identifier, which is unique. The session identifier can be read using the SessionID property. When a session state is enabled, each request for a page in the application is examined for the SessionID value. If the SessionID value is not supplied, a new session is started. The SessionID is then sent for that session to the browser along with the response.

By default, SessionID values are stored in a cookie. As long as requests continue to be made with the same SessionID value, the session is considered as active. If a request is made with an expired SessionID value, a new session is started.

Figure 10.11 demonstrates the concept.

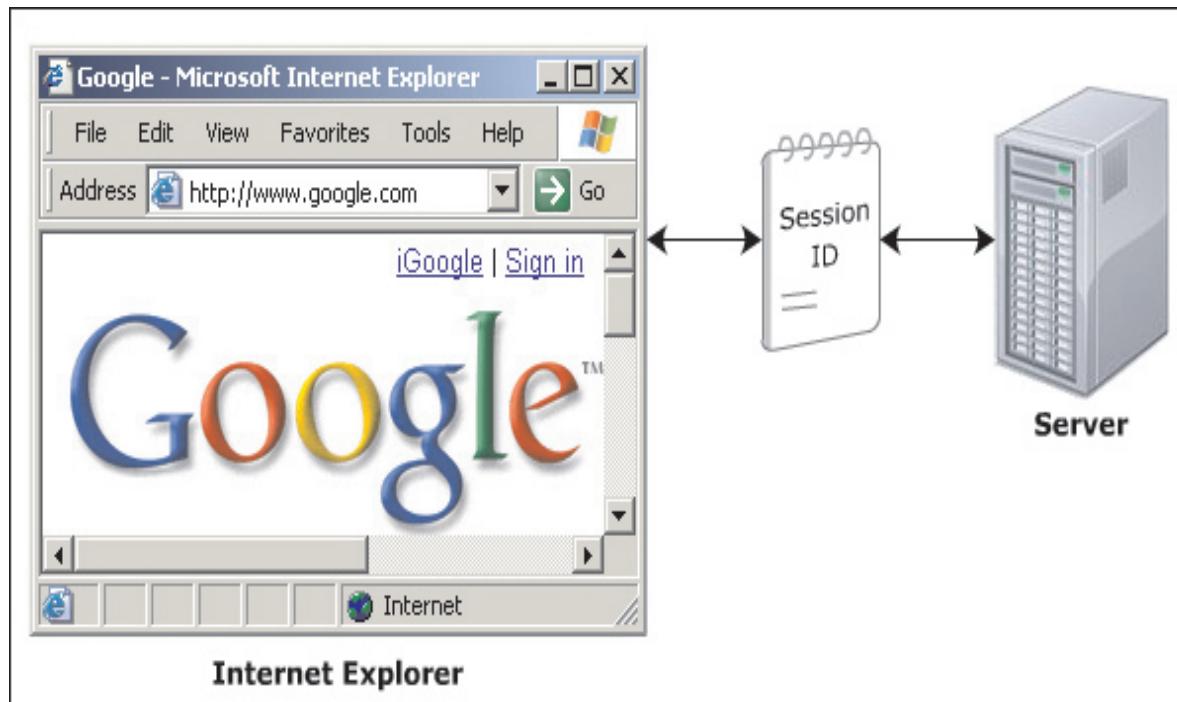


Figure 10.11: Session Identifiers Example

The following code demonstrates the use of session identifiers.

Code Snippet:

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("Session ID: " + Session.SessionID);
}
```

In this code, `SessionID` property retrieves the session's ID. `Response.Write()` method displays this session ID on the Web page.

Knowledge Check 3

1. Which of the following statements about the session variables, session events, and session identifiers are false?

(A)	Session variables are stored in the <code>SessionStateCollection</code> class.
(B)	<code>SessionID</code> values are stored in a cookie.
(C)	Session variables are used to store information about multiple user sessions.
(D)	<code>Session_OnStart</code> event is handled by adding a subroutine named <code>Session OnStart</code> .
(E)	A new session is started when the <code>SessionID</code> value is supplied.

(A)	A, C, E	(C)	A, B
(B)	A, C	(D)	C, D, E

10.4 Global.asax

In this last lesson, `Global.asax`, you will learn to:

- Explain the `Global.asax` file.
- Describe how to use the `Global.asax` file.

10.4.1 Overview of `Global.asax` File

`Global.asax` file is referred to as an ASP.NET application file. This file contains the code for responding to application or module-level events in one central location. The `Global.asax` file is an optional file and created only if the application or session events need to be handled.

The `Global.asax` file is located in the root application directory. The `.asax` extension of `Global.asax` file denotes that it is an application file rather than an ASP.NET file, which would have the extension `.aspx`.

Figure 10.12 shows the file in the Solution Explorer.

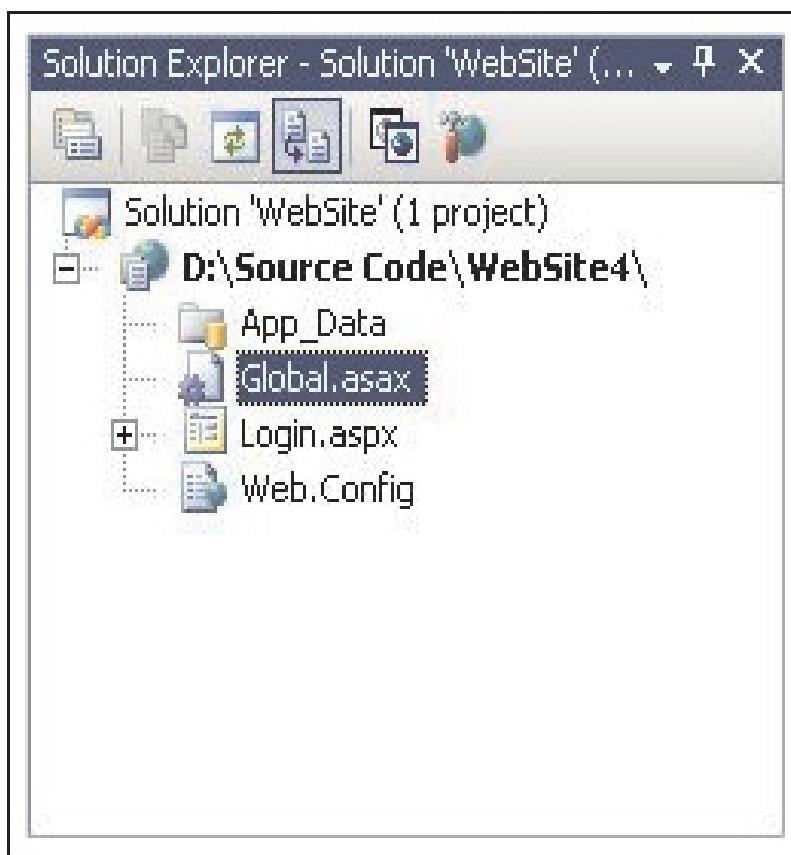


Figure 10.12: Global.asax in Solution Explorer

10.4.2 Using Global.asax File

When a new ASP.NET Web application is created with Visual Studio 2005 IDE, a `Global.asax` file is automatically added to the project. Every ASP.NET Web application can have one, and only one, `Global.asax` file. This file cannot be requested and used by the user.

There are two events associated with the `Global.asax` file, namely `Application_BeginRequest` event and `Application_EndRequest` event. The code that is to be executed in the beginning of the each page can be placed inside the `Application_BeginRequest` event handler.

Figure 10.13 demonstrates the concept.

```
<%@ Application Language="C#" %>
<script runat="server">
void Application_Start(object sender, EventArgs e)
{
    // Code that runs on application startup
}
void Application_End(object sender, EventArgs e)
{
    // Code that runs on application shutdown
}
void Application_Error(object sender, EventArgs e)
{
    // Code that runs when an unhandled error occurs
}
void Session_Start(object sender, EventArgs e)
{
    // Code that runs when a new session is started
}
void Session_End(object sender, EventArgs e)
{
    // Code that runs when a session ends.
    // Note: The Session_End event is raised only when
    // the sessionstate mode is set to InProc in the
    // Web.config file. If session mode is set to
    // StateServer or SQLServer, the event is not raised.
}
</script>
```

Figure 10.13: Skeleton Structure of Global.asax

Knowledge Check 4

1. Which of the following statements about Global.asax file are false?

(A)	The Global.asax file is referred to as an ASP.NET application file.
(B)	The Global.asax file is a must for an application to execute successfully.
(C)	The .asax extension of Global.asax file denotes that it is an ASP.NET file.
(D)	The Global.asax file cannot be requested and used by the user.
(E)	The Application_EndRequest event is the only event associated with Global.asax file.

(A)	A, B, E	(C)	A, B
(B)	A, C	(D)	B, C, E

Module Summary

In this module, **Application, Session, and Cookies**, you learnt about:

→ Application Object

The Application object stores data that is shared across the application. Application object in ASP.NET 2.0 is represented by `HttpApplicationState` class present in the `System.Web` namespace. The `Lock()` and `UnLock()` methods are used to restrict and allow modification of variables stored in Application object respectively.

→ Cookies

Cookies are the files that contain user information that the Web application can use whenever a particular user visits the site. Cookies can be either temporary or permanent files. Temporary cookies are known as session cookies whereas permanent cookies are known as persistent cookies.

→ Session State

Session state allows storing and retrieving values as the user traverses through the different ASP.NET pages of a Web application. By default, session state is enabled for all ASP.NET applications.

→ Global.asax

The `Global.asax` file is used to handle application events. This file is located in the root application directory. The `Global.asax` file is an optional file and is created only if the application or session events need to be handled.

Module - 11

An Introduction to Mobile Applications

Welcome to the module, **An Introduction to Mobile Applications**.

The .NET Framework provides an environment for developing Web applications for wireless devices. The mobile application architecture allows creation of Web pages that can be displayed on browsers of both, desktop as well as mobile devices. ASP.NET provides many mobile Web server controls such as such as Form, TextBox, Label, and PhoneCall.

In this module, you will learn to:

- ➔ Wireless Web Development
- ➔ ASP.NET Mobile Application Development
- ➔ Mobile Web Server Controls

Web Development
<http://www>



11.1 Wireless Web Development

In this first lesson, **Wireless Web Development**, you will learn to:

- Describe briefly the evolution of wireless Web development.
- Outline the challenges faced by wireless networks.

11.1.1 Wireless Web Development

The need to be ever-connected and a need to enable communication regardless of boundaries and barriers led to the advent of the wireless technology. Today, it has progressed to such an extent that various wireless devices like cell phones and Personal Digital Assistants (PDAs) have become a major necessity in our day-to-day life. Most of the advanced cell phones are Internet-enabled, which allows us to browse the Internet, send e-mails, receive news, sports, and weather updates, and so on.

Wireless Web development involves creating Web applications for wireless devices that can access the Internet. The .NET Framework along with ASP.NET provides the environment to develop such applications.

Figure 11.1 demonstrates the concept.



Figure 11.1: Mobile Device Accessibility

11.1.2 WAP

Wireless Application Protocol (WAP) is a set of standards for developing Web applications that use the Internet over different platforms. These standards are applicable for both, the format of the page markup (Wireless Markup Language or WML) and its related protocols. WAP provides technology for carrying Web content to wireless devices that is independent of the carrier, network technology, or the service provider.

The advancement of WAP has been a major step in the evolution of wireless Web development. WAP is available in two versions: WAP 1.0 and WAP 2.0. WAP 1.0 was the widely used standard for mobile Web applications before the release of WAP 2.0.

WAP 2.0 primarily aims at providing easy interaction of the mobile devices with the desktop computer by supporting the standard Internet protocol for communication.

Figure 11.2 demonstrates the concept.

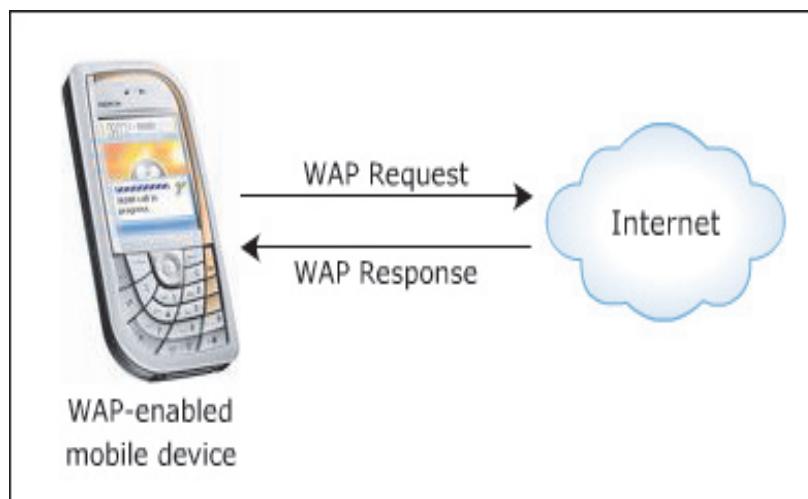


Figure 11.2: WAP Functionality

11.1.3 Challenges

Internet usage through the mobile devices such as cell phones and PDAs is rapidly increasing. The technology to access the Internet through these devices though has some challenges due to:

→ **Variable Screen Sizes**

The display of the Web content depends on the display capability of the mobile device. For example, a normal cell phone can only display four to six lines of text or a small-sized graphic.

→ **Lower Bandwidth**

Wireless applications are more costly to run because of the lower bandwidth of a wireless network as compared to a wired network.

→ **Necessity of Different Markup Languages**

Different mark-up languages are used for different devices. For example, HTML is used for PDAs and WML is used for WAP-enabled cell phones.

Knowledge Check 1

1. Which of the following statements about the evolution of wireless Web networks and the challenges faced by them are false?

(A)	The code of the .NET Compact Framework and the XML Web services enables creation of downloadable Web applications for mobile devices.
(B)	WAP provides technology for carrying Web content to wireless devices depending on the carrier, network technology, or the service provider.
(C)	Wireless applications are more costly to run because of the lower bandwidth of the wireless network as compared to the wired network.
(D)	A common mark-up is enough while working with different mobile devices.
(E)	The WAP is applicable for the format of the page markup (WML) as well as its related protocols such as WTP and WTLS.

(A)	A, B, E	(C)	A, B
(B)	A, C	(D)	B, D

11.2 ASP.NET Mobile Application Development

In this second lesson, **ASP.NET Mobile Application Development**, you will learn to:

- Explain the mobile application architecture in ASP.NET.
- List the advantages of mobile application architecture in ASP.NET 2.0.

11.2.1 Mobile Application Architecture

The Mobile application architecture allows creation of Web pages that can be displayed on browsers of both desktop and mobile devices. The mobile devices can include PDAs, palmtops, cell phones, and so forth.

Mobile devices use the mobile application architecture to request for the Web pages from the IIS server.

The ASP.NET mobile controls are included in the Mobile Application Architecture. These controls use the .NET Framework and Visual Studio to build mobile Web applications by allowing ASP.NET to send markup to different mobile devices.

The different components and services of the .NET Framework such as the Common Language Runtime, XML Web Services, and ADO.NET help develop mobile Web applications.

Figure 11.3 demonstrates this concept.

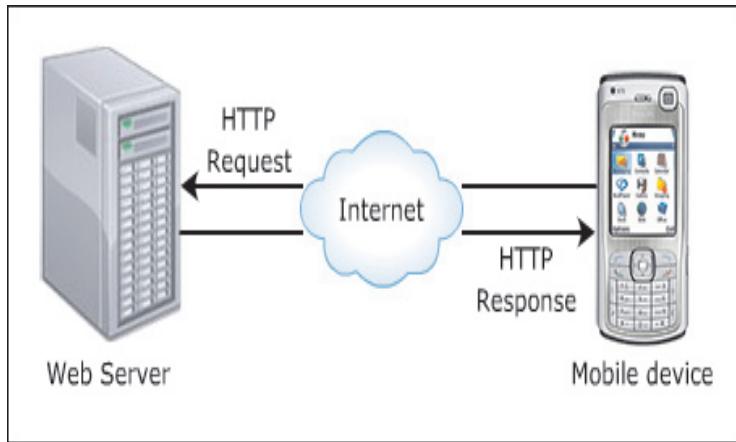


Figure 11.3: Mobile Application Architecture

11.2.2 Working of Mobile Application Architecture

A mobile device sends an HTTP request to the Web server over the Internet. The Web server identifies the requesting device and its attributes such as browser and markup. The Toolkit fills the `machine.config` file of the .NET Framework with the device data. The HTTP request sent using the mobile device contains the user agent string, the header information, and the URL. The user agent string is matched with the entries in the `machine.config` file. The URL from the HTTP request can be used to locate the corresponding ASP.NET mobile Web page.

The ASP.NET page is parsed and compiled and then, stored in the assembly cache. The equivalent machine code for the compiled ASP.NET page and the Mobile controls used on it is created. The device adapters then generate the appropriate markup language. The markup language is then encapsulated and is sent to the mobile device using the HTTP response.

Figure 11.4 illustrates the concept.

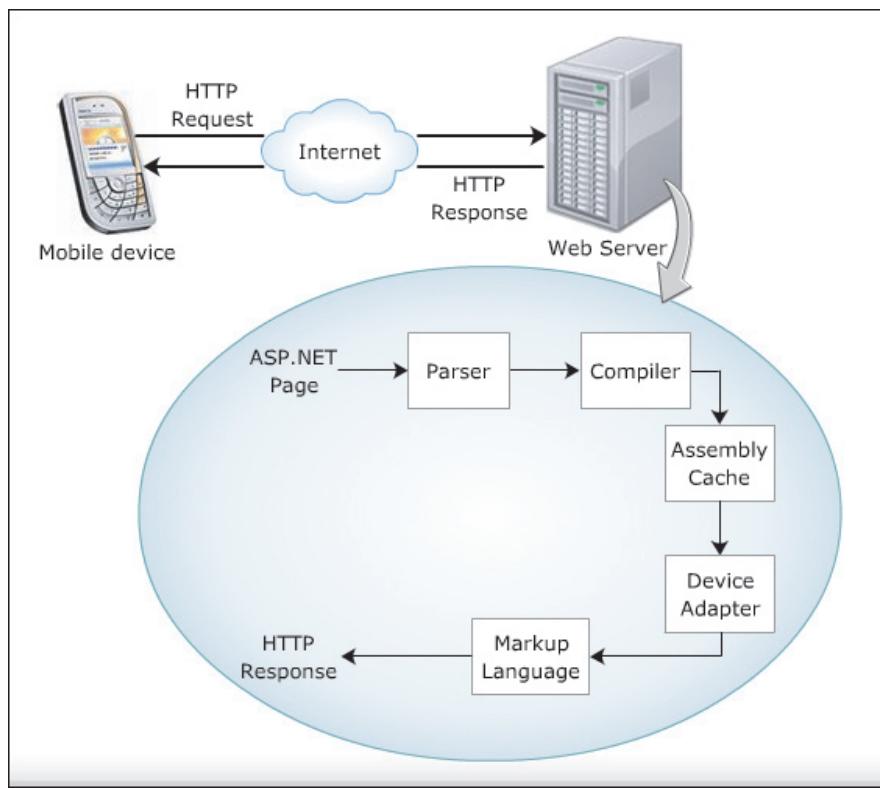


Figure 11.4: Mobile Application Architecture

11.2.3 Advantages of Mobile Application Architecture in ASP.NET 2.0

The mobile application architecture of ASP.NET 2.0 provides the following advantages:

- The device filter can be easily added in an ASP.NET 2.0 mobile Web application by declaring it in the markup.
- The handling of browser capabilities files is more simplified for ASP.NET mobile applications. If the mobile device is changed, you can easily manage the required changes by updating the individual browser file rather than making changes in the `machine.config`, `web.config`, and `config` files.
- The CLR handles execution of code and provides useful services such as memory management, security management, and code verification for execution of the mobile Web applications.

Knowledge Check 2

1. Which of the following statements about the mobile application architecture are false?

(A)	The mobile application architecture allows creation of Web pages that can be displayed on browsers of both, desktop as well as mobile devices.
(B)	The device filter can be added in an ASP.NET 2.0 mobile Web application by declaring it in the markup.
(C)	The .NET Framework services include the Common Language Runtime, XML Web Services, and ADO.NET for data access.
(D)	The HTTP request sent using the mobile device contains the User Control string.
(E)	The mobile application architecture requires making changes in the <code>machine.config</code> , <code>web.config</code> , and <code>config</code> files when a mobile device is changed.

(A)	A, B, E	(C)	A, B
(B)	A, C	(D)	D, E

11.3 Mobile Web Server Controls

In this third lesson, **Mobile Web Server Controls**, you will learn to:

- Describe the `System.Web.Mobile` namespace.
- List and describe the commonly used mobile Web server controls.
- Explain how to work with Web forms that can be used with mobile devices.
- Outline the step by step procedure to create mobile Web applications with ASP.NET 2.0.
- Describe device filtering.

11.3.1 `System.Web.Mobile Namespace`

The `System.Web.Mobile` namespace provides basic functionalities for mobile Web applications. The namespace includes classes for processing errors and perform authentication on Web Forms. The `System.Web.Mobile` namespace also includes classes to retrieve the capabilities of the client mobile device. Table 11.1 describes the important classes included in the `System.Web.Mobile` namespace.

Class	Description
CookielessData	Used internally by ASP.NET to store session information on and retrieve session information from devices that do not allow cookies.

Class	Description
CookielessData	Used internally by ASP.NET to store session information on and retrieve session information from devices that do not allow cookies.
MobileCapabilities	Retrieves the capability information about a client mobile device.
DeviceFilterElement	Specifies a filter for a specific mobile device to be applied to the mobile Web application.
DeviceFiltersSection	Contains the device filters collection to be included in the mobile Web application.

Table 11.1: Different classes in System.Web.Mobile

11.3.2 Mobile Web Form and MobilePage Control

It is mandatory to create a mobile Web Form in order to include mobile Web controls in an application. This is because the mobile Web Form acts as a container of controls. Multiple mobile Web Forms can be added to a mobile Web page but only one can be viewed at a time.

The `MobileControl` class, which is included in the `System.Web.UI.MobileControls` namespace, serves as the base class for all the ASP.NET mobile control classes.

The `MobilePage` class acts as the base class for a mobile Web Form. The `MobilePage` control contains style-related data common to all controls.

Using the ToolBox in the IDE, you can add mobile controls to a mobile Web Form by dragging the control from the **Mobile Web Forms** tab and dropping it in the **Design View**.

Figure 11.5 demonstrates the concept.

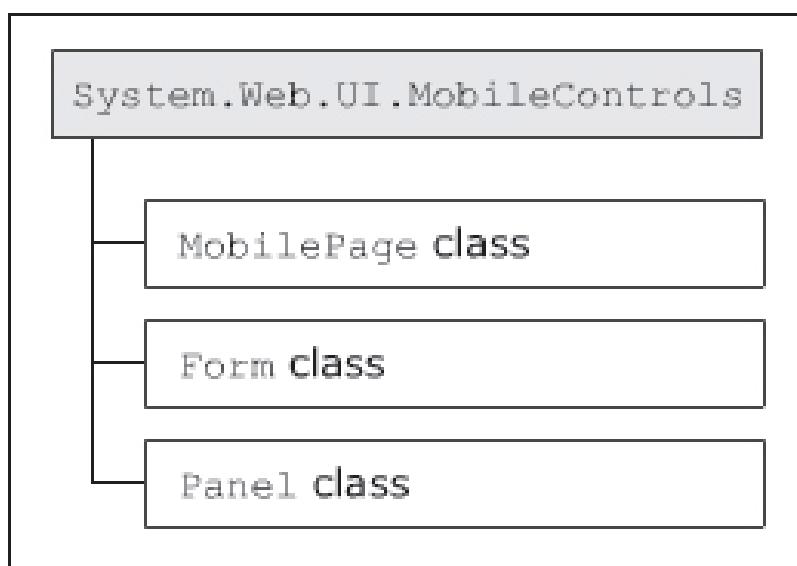


Figure 11.5: Mobile Controls

11.3.3 Container Controls

ASP.NET primarily includes three container controls that can contain other mobile controls used on a mobile Web page. These container controls are as follows: `MobilePage`, `Form`, and `Panel`.

The `MobilePage` control provides the outermost layer of all the containers in an ASP.NET mobile Web application. Each ASP.NET mobile Web Forms page creates an instance of a `MobilePage` control.

The mobile `Form` control acts as a container of different mobile Web server controls. A mobile Web application can contain multiple `Form` controls but these can only be displayed one at a time. A `Form` control can also contain one or more `Panel` controls.

A `Panel` control acts as a container for organizing other mobile Web server controls, such as `TextBox`, `Label`, `Command`, and `Link` on the mobile Web Form. The `Panel` control can be used to show, hide, enable, or disable the controls contained within the panel. `Panel` controls can also contain other `Panel` controls.

Figure 11.6 demonstrates the concept.

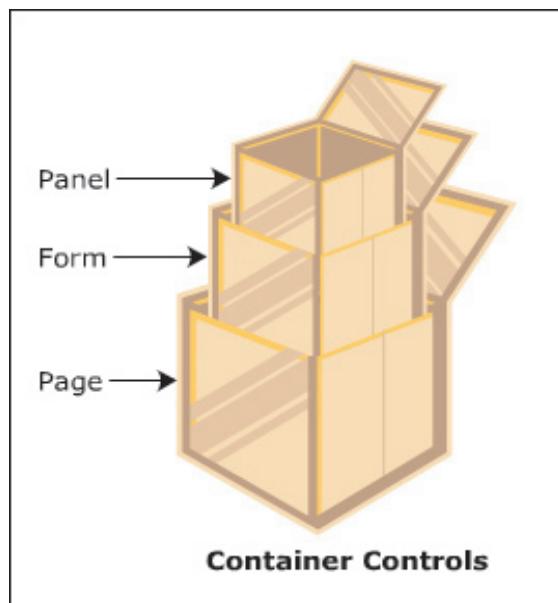


Figure 11.6: Container Controls

11.3.4 Mobile Web Server Controls

ASP.NET 2.0 provides various mobile Web server controls to allow interactivity with the user on a mobile Web page.

Following are some of the important mobile Web server controls:

- ➔ Label
- ➔ Text box
- ➔ PhoneCall
- ➔ List
- ➔ Link

→ **Label**

The `Label` control in a mobile form is similar to the `Label` control in a Web Form. This control is used to display text on the mobile form. The text on the `Label` control is displayed using the `Text` property. You cannot edit text on the `Label` control.

The syntax to create the `Label` control is:

Syntax:

```
Label <objectname> = new Label();
```

The following code demonstrates the use of the `Label` control.

Code Snippet:

```
lblFirstName.Text = "FirstName";
```

In this code, the text content of the `Label` control, `lblFirstName`, is specified using the `Text` property.

→ **TextBox**

The `TextBox` control is used to accept input from the user. The mobile `TextBox` control accepts short text entries and does not provide functionality to accept multiple-line text entry.

The syntax to create the `TextBox` control is:

Syntax:

```
TextBox <objectname> = new TextBox();
```

The following code demonstrates the use of the `TextBox` control.

Code Snippet:

```
txtAge.Numeric = true;
```

In this code, the `TextBox` control, `txtAge`, is specified as a numeric input control with the value as `true`.

→ **PhoneCall**

The `PhoneCall` control is a text-based output control that is used to represent a phone number that is to be called. When the `PhoneCall` control is activated, it makes a phone call.

The syntax to create the `PhoneCall` control is:

Syntax:

```
PhoneCall <objectname> = new PhoneCall();
```

The following code demonstrates the use of the `PhoneCall` control.

Code Snippet:

```
pcManager.PhoneNumber = "1488200912";
```

In this code, the PhoneCall control, pcManager, is specified a string value, 1488200912, using the PhoneNumber property.

→ List

The List control is allows displaying items in a list and provides easy navigation of the items and allows item selection.

The syntax to create the List control is:

Syntax:

```
List <objectname> = new List ();
```

The following code demonstrates the use of the List control.

Code Snippet:

```
lstQualification.ItemsAsLinks = true;
```

In this code, the ItemsAsLinks is set to true indicating that the items in the List control, lstQualification, act as hyperlinks.

→ Link

The Link control allows displaying the text string that is represented as a hyperlink on the mobile devices.

The syntax to create the Link control is:

Syntax:

```
Link <object name> = new Link ();
```

The following code demonstrates the use of the Link control.

Code Snippet:

```
lnkClickHere.NavigateUrl = "http://www.example.com";
```

In this code, the string http://www.example.com is assigned to the NavigateUrl property, which redirects the user to the URL specified by the string when the Link control, lnkClickHere, is clicked.

Figure 11.7 shows an example that uses these controls.

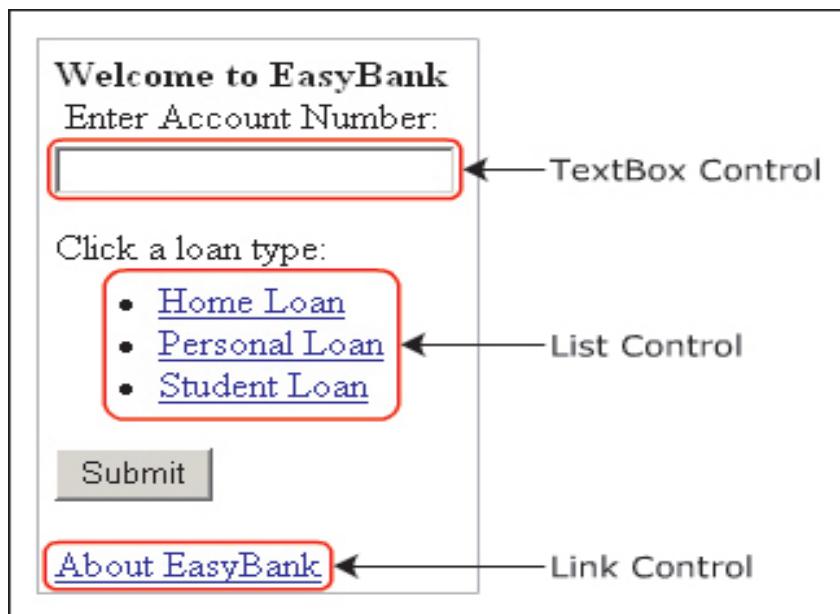


Figure 11.7: Mobile Web Server Controls

11.3.5 Working with Mobile Web Forms

Microsoft Visual Studio IDE allows you to create ASP.NET applications that include mobile Web pages. The supporting tools provided by the Visual Studio 2005 IDE such as Toolbox, Source view, and Design view allow creating Web Forms that can be used with mobile devices. The ASP.NET automatically configures the settings of the mobile Web page relevant to a specific device.

Once a mobile Web application is created, it can easily be tested using an emulator. An Emulator is a software application that allows you to get a layout of the mobile Web application on a specific mobile hardware. Pocket PC 2003 SE Emulator and Smartphone 2003 SE Emulator are two of the commonly used emulators in Visual Studio 2005.

Figure 11.8 demonstrates the concept.



Figure 11.8: Mobile Web Forms

11.3.6 Creating Mobile Web Applications

Mobile Web applications can be created using Visual Studio 2005 IDE. You can create a mobile Web application by performing the following steps:

- Create an empty ASP.NET Web Site
- Add Web.config File
- Add a Mobile Web Form
- Create a Mobile Web Application
- View the Mobile Web Application

Figure 11.9 demonstrates the concept.

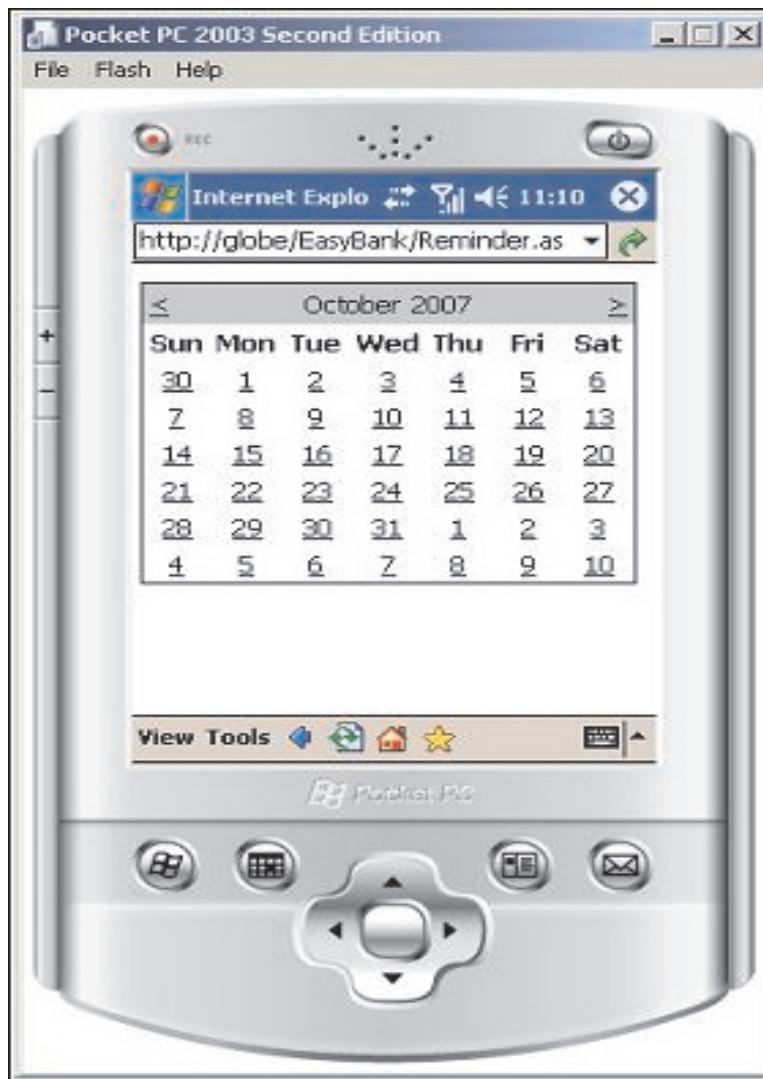


Figure 11.9: The application being tested On Emulator

→ **Create an empty ASP.NET Web Site**

The **New Web Site** option is clicked from the **New** option in the **File** menu. This displays the **New Web Site** dialog box. From the **Visual Studio installed templates** section, the **Empty Web Site** option is clicked. In the **Name** box, the appropriate name for the application is entered. In the **Location** box, the location of the Web server is specified by selecting or entering the URL. Then, the **OK** button is clicked. This displays a plain ASP.NET Web application project with the solution root in the **Solution Explorer** window.

→ **Add Web.config File**

The solution root is right-clicked and the **Add New Item** option is selected. This displays the **Add New Item** dialog box. The **Mobile Web Configuration File** is selected and the **Add** button is clicked. This adds the `Web.config` file to the **Solution Explorer** window.

→ **Add a Mobile Web Form**

The solution root is again right-clicked and **Add New Item** option is selected. From the **Add New Item** dialog box, **Mobile Web Form** option is selected and the **Add** button is clicked. The mobile Web Form is thus added to the **Solution Explorer** window.

→ **Create a Mobile Web Application**

Once a mobile Web Form is created, you can then, drag a mobile control from the Mobile Web Forms tab in the **Toolbox** and drop it on the form. Accordingly, write the code for the different controls.

→ **View the Mobile Web Application**

Assuming that you have created the mobile Web site, you can test and view the Web application in the emulator. You can start the emulator by selecting **Connect To Device** from the **Tools** menu and then, selecting the appropriate emulator from the list provided in the **Connect to Device** dialog box and clicking the **Connect** button.

11.3.7 Device Filtering

People around the world use different mobile devices. These devices may differ in display sizes and capabilities. If a Web developer develops a Web application for a specific mobile device, it may not work when exported to other mobile devices. To overcome this problem, there is a need of some means of device filtering. Device filtering is the process of customizing mobile Web server controls to correctly display them on select mobile devices.

Using device filters, mobile Web applications can customize the appearance of controls for specific hardware devices. The customization is based on the capabilities of the hardware device being used to browse the application. Device filters are used to customize the behavior of Web server controls depending on the browser or device that accesses them.

Figure 11.10 demonstrates the concept.

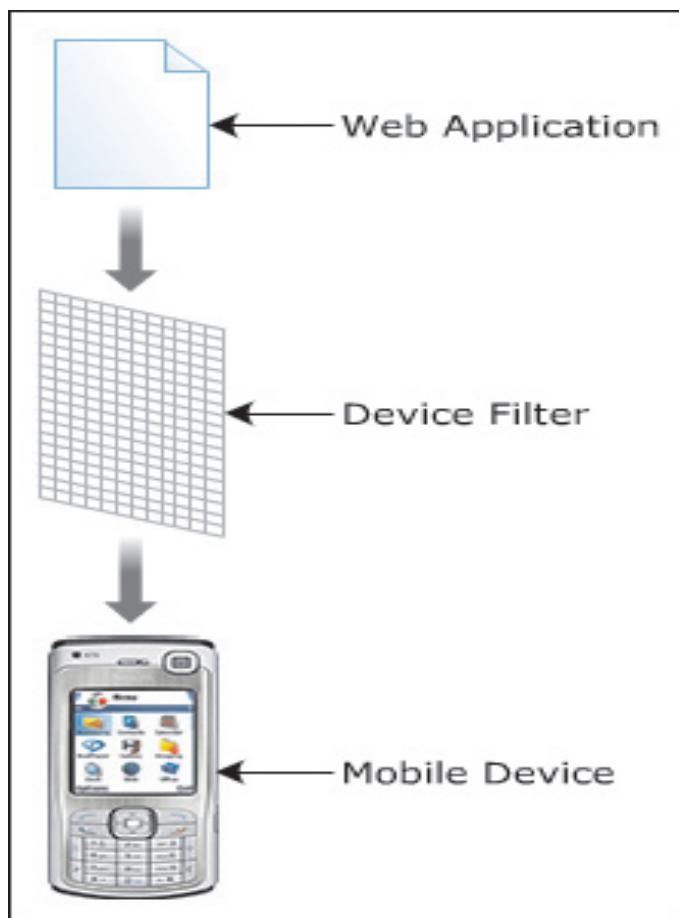


Figure 11.10: Device Filtering

11.3.8 Device Filtering in ASP.NET 2.0

When device filtering has been added, the browser sends a request containing the information such as the user agent and headers when a Web page is requested to the server. The request information identifies the browser capabilities such as its type, version, and so on. ASP.NET matches the identifier with the mobile device that is defined in the browser file. The mobile device filters the output using the identifier in Web server controls and thus, renders device-specific output.

Figure 11.11 demonstrates the concept.

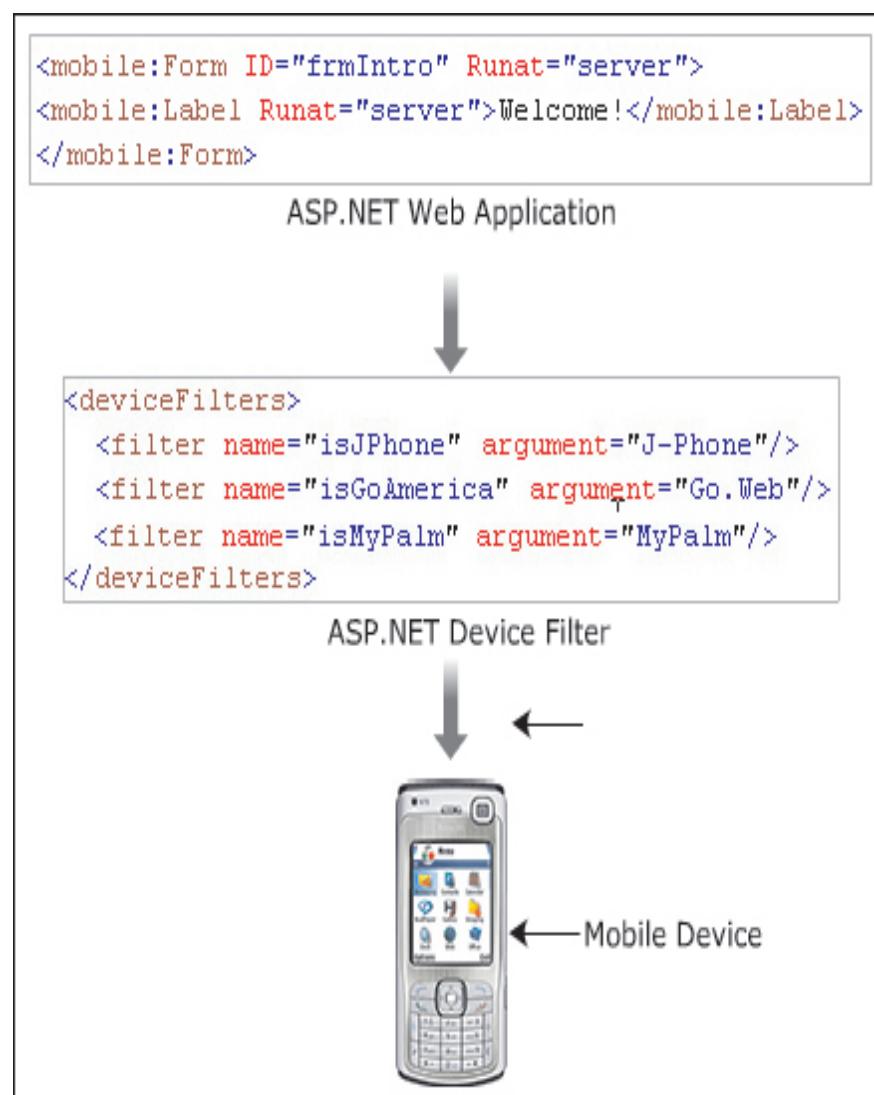


Figure 11.11: Device Filtering in ASP.NET 2.0

Knowledge Check 3

1. Which of the following statements about mobile Web server controls, mobile Web forms, and device filtering are false?

(A)	To create an ASP.NET mobile Web application, you have to first create an empty ASP.NET Web site
(B)	The <code>System.Web.UI.MobileControls</code> namespace contains a set of ASP.NET server controls to be used for developing Web applications for different mobile devices.
(C)	The mobile Web application can be tested and viewed using an emulator.
(D)	The <code>TextBox</code> mobile Web server control allows multiple-line text entry.
(E)	An emulator is a software application that allows you to get a layout of the mobile Web application on a specific mobile hardware device.

(A)	A, B, E	(C)	A, B
(B)	A, C	(D)	B, D

2. Can you match the Mobile Web Server Controls with their corresponding descriptions?

	Description	Control	
(A)	Represents a phone number to be called.	(1)	Label
(B)	Allows displaying the text string as a hyperlink.	(2)	Form
(C)	Displays non-editable text on the control.	(3)	PhoneCall
(D)	Accepts short text entry input from the user.	(4)	Link
(E)	Acts as a container of the different mobile Web server controls.	(5)	TextBox

(A)	(A)-(2), (B)-(4), (C)-(5), (D)-(1), (E)-(3)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(4), (C)-(1), (D)-(5), (E)-(2)	(D)	(A)-(4), (B)-(1), (C)-(2), (D)-(5), (E)-(3)

Module Summary

In this module, **An Introduction to Mobile Applications**, you learnt about:

→ **Wireless Web Development**

Wireless Web development is used to create Web applications for wireless devices. The .NET Framework along with ASP.NET provides the environment to develop Web applications for wireless devices. WAP provides technology for carrying Web content to wireless devices independent of the carrier, network technology, or the service provider.

→ **ASP.NET Mobile Application Development**

The mobile application architecture allows creation of Web pages that can be displayed on browsers of both, desktop as well as mobile devices. The Mobile Web Server controls are important components of the mobile application architecture.

→ **Mobile Web Server Controls**

The MobileControl class serves as the base class for all the ASP.NET mobile controls classes. ASP.NET provides many mobile Web server controls such as Form, TextBox, Label, and PhoneCall. Emulators can be used to simulate the layout of the mobile Web application for specific mobile hardware devices. Device filters are used to customize the behavior of the Web server controls for specific browsers or devices.

Module - 12

Tracing, Debugging, and Diagnostics

Welcome to the module, **Tracing, Debugging, and Diagnostics**.

Tracing is the process of monitoring the execution of an application. It helps in detecting errors and debugging the code. Debugging is the processes of finding and eliminating errors. Errors can be debugged either at the page level or at the application level. The `System.Diagnostics` namespace provides classes that allow interaction with system processes, event logs, and performance counters, which assists in debugging applications and tracing their execution.

In this module, you will learn to:

- Tracing
- Debugging
- Error handling

Web Development

http://www



12.1 Tracing

In this first lesson, **Tracing**, you will learn to:

- Define tracing and describe the need for it.
- Explain application level tracing.
- Explain how to enable and configure trace switches.
- Describe how to display trace information.
- List and describe briefly the new features of tracing in ASP.NET 2.0.

12.1.1 Need for Tracing

Consider a traveler checking into a hotel. The receptionist in this case notes down the time the customer has checked in and also enters the personal details of the customer such as his/her name, age, address, and contact number. The hotel also maintains a record of all items ordered by the traveler during his/her stay and the number and details of phone calls made. This record helps the hotel management to trace the traveler in case there is some urgent need.

In the same manner, tracing is used in ASP.NET Web applications to trace the working of a Web application and provide informative messages at runtime which will help to track the progress of the application.

12.1.2 Introduction to Tracing

Tracing in ASP.NET refers to the process of monitoring the execution of ASP.NET Web applications. It can also involve recording exceptions that occur during runtime. Additionally, tracing also records the time that is spent rendering the page, the login and logout times, and so on. The trace process does not affect the flow of the application.

Tracing can be used to monitor the execution of a single Web page or the entire Web application.

The `Trace` class in the `System.Diagnostics` namespace provides methods and properties that help in tracing the execution of your code. This class cannot be inherited.

Figure 12.1 demonstrates the concept.

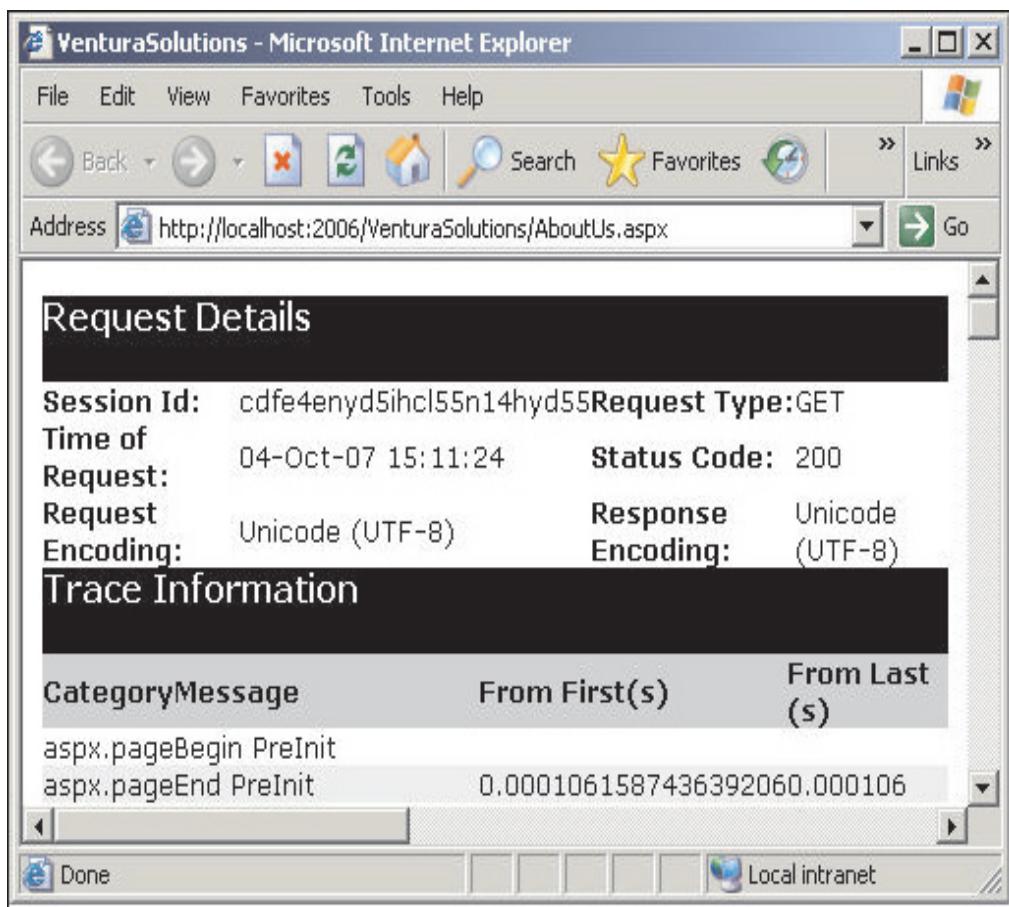


Figure 12.1: Tracing in ASP.NET

12.1.3 Application Level Tracing

Page-level tracing allows tracing individual pages in the Web application. Such tracing can be done by enabling the `@ Page` directive in the required Web pages. For disabling tracing for the page, the `@ Page` directive has to be removed. However, the task of enabling and disabling tracing for individual pages of the application is tedious if the application is too large. Also, it is prone to human errors.

ASP.NET provides a better solution for tracing Web applications by introducing Application-level tracing. Application-level tracing can be enabled in the `Web.Config` file by adding `<trace>` node in the Web application. The `Web.config` file is an XML file that contains the information regarding authentication, security, tracing, and error-reporting needed by the Web server to process the ASP.NET application.

Unlike, page-level tracing, application-level tracing can be disabled directly in the `Web.Config` file by removing the `<trace>` node or setting it to false.

Figure 12.2 demonstrates the concept.

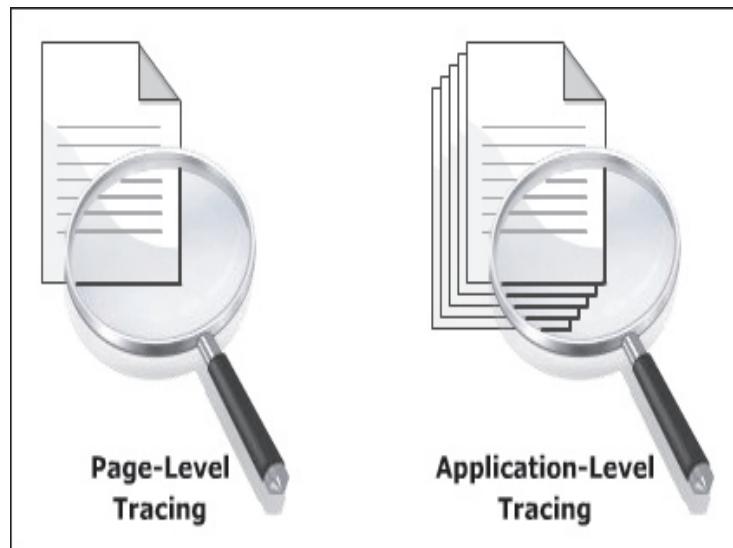


Figure 12.2: Levels of Tracing

The following code demonstrates how to enable page-level tracing.

Code Snippet:

```
<%@ Page Trace="true" %>
```

In this code, the trace attribute is set to true in @ Page directive. This enables page-level tracing.

To enable application-level tracing, in the Web.config file, add the following code:

Code Snippet:

```
<configuration>
  <system.web>
    <trace enabled="true" pageOutput="true" requestLimit="10"
      localOnly="false"/>
  </system.web>
</configuration>
```

Here, in this code, a trace element is added as a child of the system.web element. Tracing can be enabled by setting the enabled attribute to true. The trace information is displayed at the end of the page by setting the trace element's pageOutput attribute to true. The requestLimit attribute is set to 10 to collect the trace information for 10 requests. The localOnly attribute is set to false to allow client browsers to display the trace viewer.

12.1.4 Attributes of trace Node

The `<trace>` node tag is added to the `Web.Config` file to enable application-level tracing. The various attributes of this node define its functionality. Some of the important attributes are as follows:

→ **enabled**

The `enabled` attribute is used to enable or disable tracing. By default, this attribute is set to `false` and tracing is disabled. To enable tracing, you have to set the `enabled` attribute to `true`.

The following code demonstrates the use of `enabled` attribute of the `trace` node.

Code Snippet:

```
<trace enabled="true"/>
```

In this code, the attribute `enabled` is set to `true` to enable tracing.

→ **pageOutput**

The `pageOutput` attribute is used to display the trace information in the Web application's page or in the `Trace.axd` page. The `Trace.axd` page provides a list of all traced pages. The detailed trace information about these pages is provided using hyperlinks in the `Trace.axd` page. The `pageOutput` attribute, by default, is set to `false`.

The following code demonstrates the use of the `pageOutput` attribute of the `trace` node.

Code Snippet:

```
<trace pageOutput="true"/>
```

In this code, the `pageOutput` attribute is set to `true` to trace information that appears at the end of the page.

→ **requestLimit**

The `requestLimit` attribute stores the trace information on the server. By default, it stores 10 requests.

The following code demonstrates the use of the `requestLimit` attribute of the `trace` node.

Code Snippet:

```
<trace requestLimit="30"/>
```

In this code, the `requestLimit` attribute is assigned to 30 so that it collects trace information for only 30 requests.

→ **traceMode**

The `traceMode` attribute specifies the way to display the trace information. It can be `SortByTime`, if the trace information is to be displayed in order it was processed, or `SortByCategory`, if the

trace information is to be displayed in alphabetical order.

The following code demonstrates the use of the `traceMode` attribute of the `trace` node.

Code Snippet:

```
<trace traceMode="SortByTime"/>
```

In this code, the `traceMode` attribute is set to `SortByTime` to sort the methods according to the CPU time they have taken up.

→ **localOnly**

The `localOnly` attribute, when set to true, allows the user to view the trace information from the remote computer. By default, this value is set to true.

The following code demonstrates the use of the `localOnly` attribute of the `trace` node.

Code Snippet:

```
<trace localOnly="true"/>
```

In this code, the `localOnly` attribute is set to `true` to display the trace viewer only on the server.

12.1.5 Enabling Tracing

It is possible to enable tracing for one or more pages or for an entire ASP.NET Web application. The entire Web application can be traced by changing the configuration system of the application in the `Web.Config` file. The page-level tracing can be enabled by adding the `@ Page` directive in pages that require to be traced.

→ **Tracing an ASP.NET application**

Application-level tracing can be enabled or disabled without editing individual pages in the application. The tracing for all the pages in the application can be turned off after the application has completed.

The tracing information for each request is stored when tracing for the application is enabled. By default, ASP.NET stores tracing information for 10 requests. When the trace viewer reaches its request limit, the ASP.NET application stops saving the trace request. The trace information can be viewed by a trace viewer.

The setting for the application-level tracing can be configured such that it only saves the most recent data and discards the oldest data when the request has reached its maximum level.

Figure 12.3 demonstrates the concept.

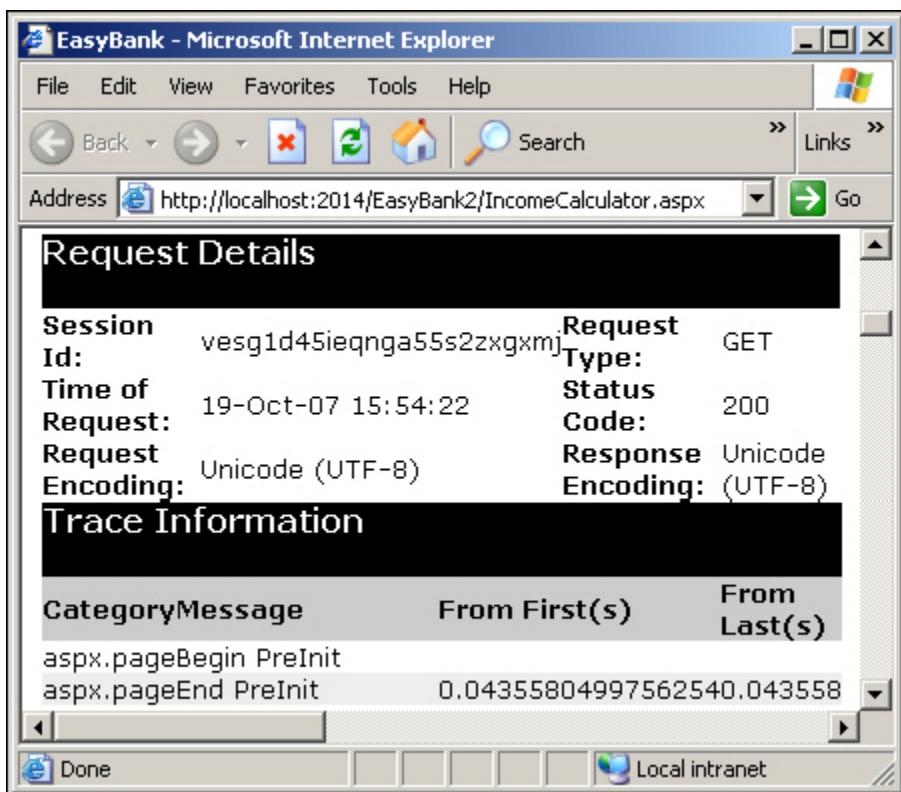


Figure 12.3: Enabling Tracing in ASP.NET

The following code is added to the Web.config file. It demonstrates how to enable tracing for an ASP.NET application.

Code Snippet:

```
<configuration>
  <system.web>
    <trace enabled="true" pageOutput="true" requestLimit="5" traceMode="SortByCategory"/>
  </system.web>
</configuration>
```

In this code, pageOutput attribute is set to true and requestLimit attribute is set to 5. Hence, the trace information is stored for 5 requests and is displayed on the page.

→ Steps to enable tracing in the application

To enable application-level tracing, the Web.Config file of the Web application is opened from the **Solution Explorer** window. If this file is not present in the **Solution Explorer** window, then a new file is to be created in the root folder of the Web application. After creating the Web.Config file, within the `<configuration>` tag, the opening and closing tags for `<system.web>` are added.

These tags are added before the closing `</configuration>` tag. Within the `<system.web>` tag, the trace node is added and its `Enabled` attribute is set to `true`. The changes made to the `Web.Config` file are then, saved. This enables tracing in the application.

Next, a trace element is added as a child of the `System.Web` element. The `enabled` attribute of the trace element is set to `true`.

The application can be configured based on where the trace information needs to be displayed. For example, to display the trace information at the end of the page, the trace element's `pageOutput` property is set to `true`. If the trace element's `pageOutput` property is set to `false`, then the trace information is saved in the **Trace Viewer**. It is important to note that the ASP.NET configuration system is case-sensitive.

12.1.6 Trace Switches

Trace switches help you enable and disable trace output. This can be done by configuring the `.config` file. Trace switches are also useful for filtering trace information. For example, you might want to see login and session information for the login form in the application whereas see the error messages for the rest of the application. To do this, you will need to use two trace switches, one for the login form and the other for the rest of the application.

Trace switches are represented by trace switch classes. These classes are inherited from the base `Switch` class in the `System.Diagnostics` namespace. There are three trace switch classes provided by the .NET Framework.

Figure 12.4 demonstrates the concept.

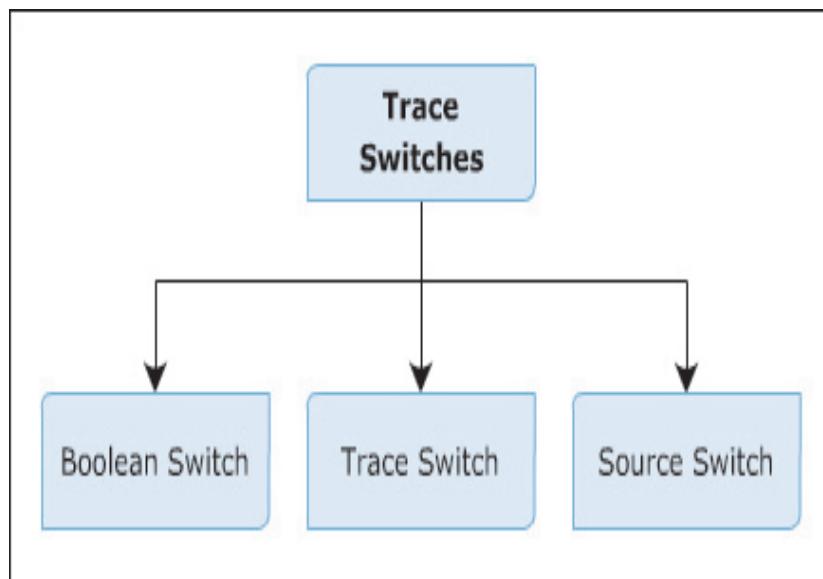


Figure 12.4: Trace Switches

These are described as follows:

→ BooleanSwitch Class

`BooleanSwitch` class creates a toggle switch. This switch is used for enabling or disabling the debugging and tracing output. The instance name of the `BooleanSwitch` class created in the application code should match the switch that is declared in the `.config` file. If it does not match, then the switch will be disabled by default.

The following constructor is used to initialize a new instance of the `BooleanSwitch` class with the specified display name and description:

Syntax:

```
public BooleanSwitch (string displayName, string description)
```

where,

`displayName`: Specifies the name that will be displayed on the user interface.

`description`: Gives the description of the switch.

The following code demonstrates the use of `BooleanSwitch` class.

In `Web.config` file, the following is written:

Code Snippet:

```
<configuration>
  <system.diagnostics>
    <switches>
      <add name="ShowMessage" value="1" />
    </switches>
  </system.diagnostics>
</configuration>
```

In this code, in the `<add>` element, the `name` property is set to `ShowMessage` and `value` property is set to 1 to enable the `BooleanSwitch`.

In the Code View, the following statements are written:

Code Snippet:

```
BooleanSwitch blnswShowMessage = new BooleanSwitch ("ShowMessage",
"Displays the welcome message");
Request.Write("Boolean Switch Enabled: " + blnswShowMessage.Enabled);
Trace.WriteLineIf(blnswShowMessage.Enabled, "Welcome!");
```

In this code, `blnswShowMessage`, an object of `BooleanSwitch` class, is created with parameters

ShowMessage as the display name and Displays the welcome message as the description of blnswShowMessage. The message Boolean Switch Enabled is displayed followed by either True or False depending on the value of Enabled property of the BooleanSwitch object blnswShowMessage. This is done using Response.Write() method. WriteLineIf method of Trace class checks whether or not the BooleanSwitch is enabled. If BooleanSwitch is enabled, it displays the message Welcome! on the trace viewer screen.

→ **TraceSwitch Class**

TraceSwitch can be used to enable tracing at various levels. Trace information is displayed at the specified level and all levels below it. This switch is generally useful for resolving problems in applications that are under development.

The following constructor is used to initialize a new instance of the TraceSwitch class with the specified display name and description:

Syntax:

```
public TraceSwitch (string displayName, string description)
```

where,

displayName: Specifies the name that will be displayed on the user interface.

description: Gives the description of the switch.

The code demonstrates the use of TraceSwitch class.

In Web.config file, the following is written:

Code Snippet:

```
<configuration>
  <system.diagnostics>
    <switches>
      <add name="TraceLevel" value="3" />
    </switches>
  </system.diagnostics>
</configuration>
```

In this code, in the <add> element, the name property is set to TraceLevel and value property is set to 3 to assign the TraceSwitch level as information.

Within the code-behind file, the following code is written:

Code Snippet:

```
TraceSwitch traceswTraceLevelSwitch = new TraceSwitch("TraceLevel", "Trace
Level for database application");

Request.Write("Trace Level: "+traceswTraceLevelSwitch.Level);

Trace.WriteLineIf(traceswTraceLevelSwitch.TraceInfo,"Database
connected");
```

In this code, `traceswTraceLevelSwitch`, an object of `TraceSwitch` class, is created with parameters `TraceLevel` as the display name and `Trace Level for database application` as the description for `traceswTraceLevelSwitch`. The `Trace Level` message followed by the value of `Level` property of the `TraceSwitch` object `traceswTraceLevelSwitch` is displayed by using `Response.Write()` method. `WriteLineIf` method of `Trace` class checks whether or not the level of the `TraceSwitch` is `TraceInfo`. If it is `TraceInfo`, then it displays the message `Database connected` on the trace viewer screen.

→ **SourceSwitch Class**

`SourceSwitch` is used as a multilevel switch to control tracing and debugging output without recompiling the code. This switch is used when you wish to trace only a specific section of code. The following constructor is used to initialize a new instance of the `SourceSwitch` class with the specified display name and description:

Syntax:

```
public SourceSwitch (string name)
```

where,

`name`: Specifies the name of the source.

The code demonstrates the use of `SourceSwitch` class.

In the `Web.Config` file, the following is written:

Code Snippet:

```
<configuration>
  <system.diagnostics>
    <switches>
      <add name="SrcSwitch" value="Verbose"/>
    </switches>
  </system.diagnostics>
</configuration>
```

In this code, in the `<add>` element, the `name` property is set to `SrcSwitch` and `value` property is set to `Verbose`, which provides the detailed description of the switch.

In the Code View, the following statements are written:

Code Snippet:

```
SourceSwitch sourceSwitch = new SourceSwitch("SrcSwitch");  
Response.Write("Name of SourceSwitch: " + sourceSwitch.DisplayName);
```

In this code, `sourceSwitch`, an object of `SourceSwitch` class, is created with the parameter `SrcSwitch` as its display name. The message `Name of SourceSwitch:` followed by the value of `DisplayName` property of `sourceSwitch` object is displayed on the Web page by using `Response.Write()` method.

12.1.7 Steps to Create and Configure Trace Switches

Configuring a switch involves changing its value from the initial or default value. The values can be changed using the configuration file. Trace switches are configured to turn them on or off, set their level, determine the type of messages they pass, and so on.

→ Creating Trace Switches in an ASP.NET Application

For creating a Boolean switch in the application code, the project should contain a configuration file such as `app.config` or `Web.config`. If none are present, then, from the **Project** menu, the **Add New Item** option is selected. In the **Add New Item** dialog box, the **XML File** option is selected. The file is saved as `app.config`. This creates the `.config` file, which is an **XML** document whose root element is `<configuration>`. Within this tag, a Boolean switch can be configured by adding the appropriate XML document.

Figure 12.5 displays the **Add New Item** dialog box with **XML file** selected. .

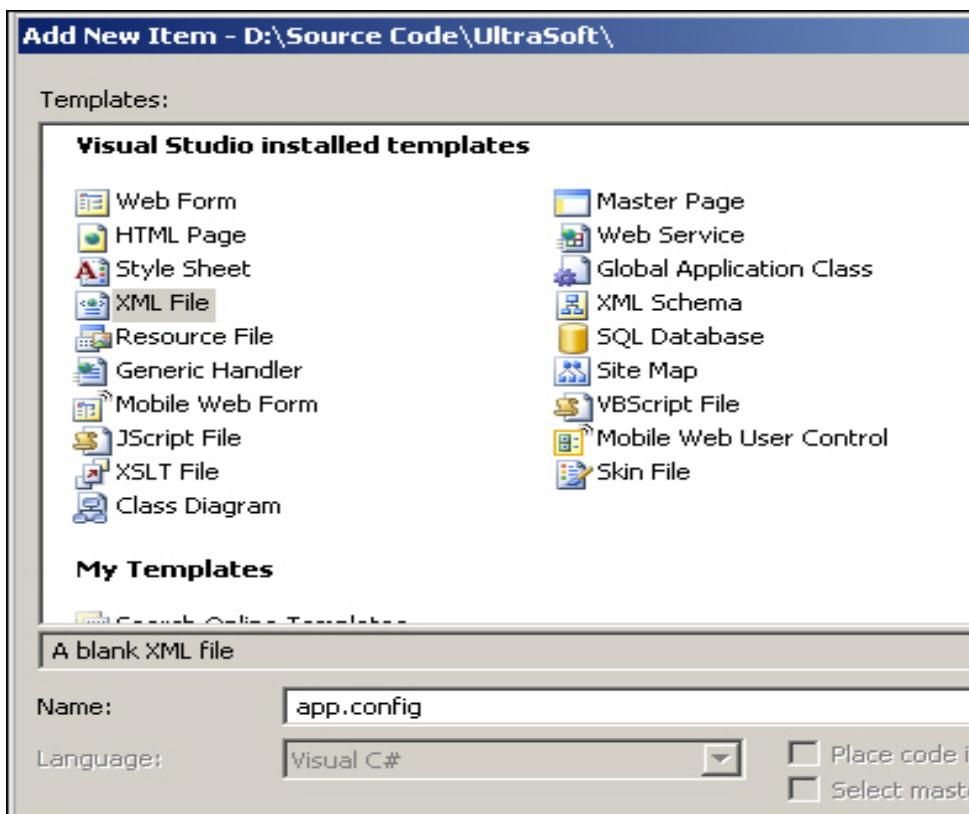


Figure 12.5: Creating an XML Configuration File

The following code demonstrates the way to configure trace switches in `app.config` file.

Code Snippet:

```
<?xml version="1.0"?>
<configuration>
  <system.diagnostics>
    <switches>
      <add name="ShowMessage" value="1" />
    </switches>
  </system.diagnostics>
</configuration>
```

In the code, the first line specifies the XML version and is set to 1.0. The `<configuration>` tag is the root element and has the closing tag `</configuration>`. The `<system.diagnostics>` is the sub-element of `<configuration>` element. In between the `<system.diagnostics>` element tags, the `<switches>` element is added. In between the `<switches>` element tags, the `<add>` element is added.

Here, no closing tag is required. In the `<add>` element, the name attribute is used to set the name of the Boolean switch as `ShowMessage` and the value attribute is set to 1 to enable Boolean switch.

→ Configuring Switches in an ASP.NET Application

For configuring switches in your application, the appropriate XML code is added between `<configuration>` and `</configuration>` tags. Various comments can be added for better understanding of what values can be changed to configure the switches. After making the appropriate changes in the `app.config` file, save the changes. While compiling the project, the `app.config` file is copied to the project output folder, which is named `applicationname.exe.config`.

Figure 12.6 demonstrates the concept.

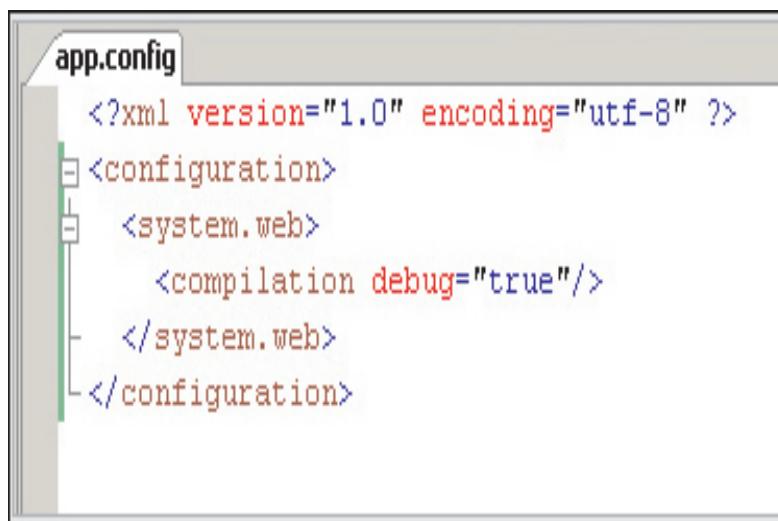


Figure 12.6: Using the `<configuration>` Tags

12.1.8 Trace Output Sections

The trace information can be viewed at the end of the ASP.NET page or in the `Trace.axd` page. The trace information is displayed in tables. This information appears in different trace output sections. Some of the important sections are as follows:

- **Request Details**—Displays the information regarding the current request and response.
- **Trace Information**—Allows displaying customized trace messages.
- **Control Tree**—Displays the information about the ASP.NET server controls.
- **Session State**—Displays information about session state values.
- **Application State**—Displays information about application state values.

12.1.9 Trace Information

Trace information section at the end of a Web page in a browser window displays information regarding page-level events in the sequence in which the events occur. If custom trace messages have been created, these messages are also displayed in the Trace Information section. Any request taking place at any level in the page will be traced and displayed in this section.

Figure 12.7 demonstrates the concept.

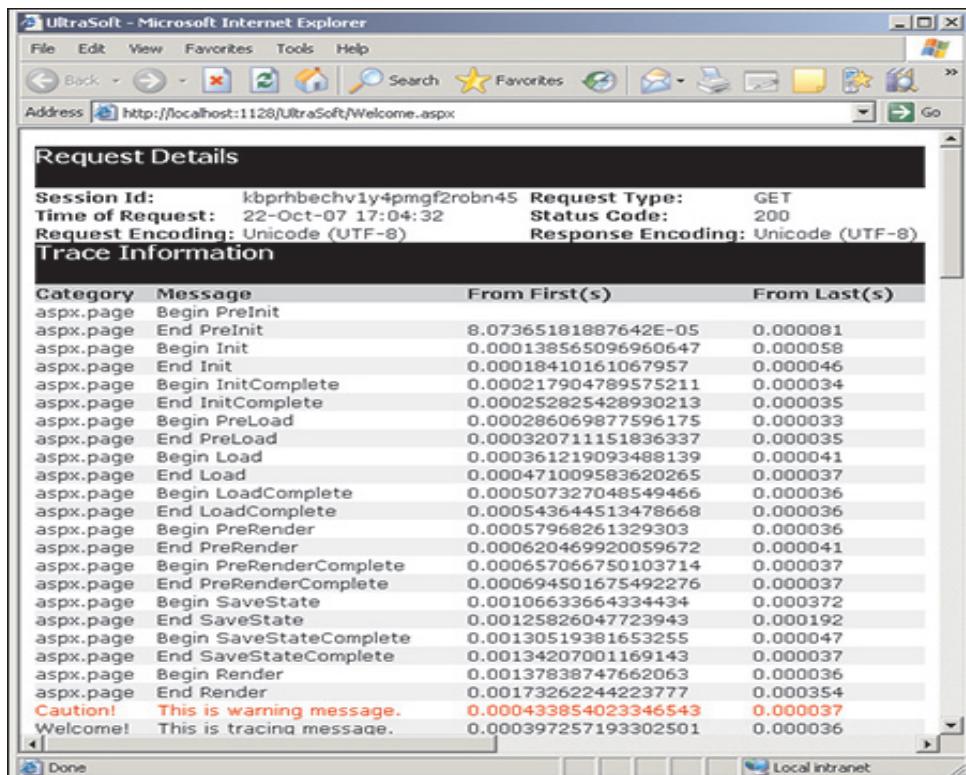


Figure 12.7: Trace Information

The different columns under which the custom trace information is displayed are as follows:

→ Category

Category displays the custom trace category as specified by the `TraceContext.Warn` or `TraceContext.Write` method. The `TraceContext.Warn` method writes the trace messages that are displayed as warnings in the trace log. The `TraceContext.Write` method writes the trace messages to the trace log.

→ Message

Message displays the custom trace message as specified by the `TraceContext.Warn` or `TraceContext.Write` method.

→ **From First (s)**

From First (s) displays the time, in seconds, since the processing of the very first trace message.

→ **From Last (s)**

From Last (s) displays the time, in seconds, since the processing of the previous trace message.

Code Snippet:

```
Trace.Write("Welcome!","This is tracing message.");  
Trace.Warn("Caution!","This is warning message.);
```

In this code, the `Trace.Write()` method is used to display the message This is tracing message on the trace log. The `Trace.Warn()` method is used to display the message This is warning message on the trace log. The message is displayed in red color after you run the page.

12.1.10 View Trace Information with Trace Viewer

When tracing is enabled for an application and a page is requested, that page collects the trace information and executes the trace statements, if any. The trace viewer enables you to see the trace output for a specific request. The `Trace.axd` file is an HTTP handler application used to view the trace output. This application allows you to handle the HTTP requests and responses.

If multiple requests arrive for an application that has tracing enabled, the trace viewer lists the various requests in the order in which they are processed. The request number does not exceed the value specified by the `requestLimit` attribute in the `Web.config` file. The information on the trace viewer page consists of the request time, name of the requested file, the status code of the request, a **View Detail** link that enables viewing information about the request, and the HTTP associated with the request.

Figure 12.8 demonstrates the concept.

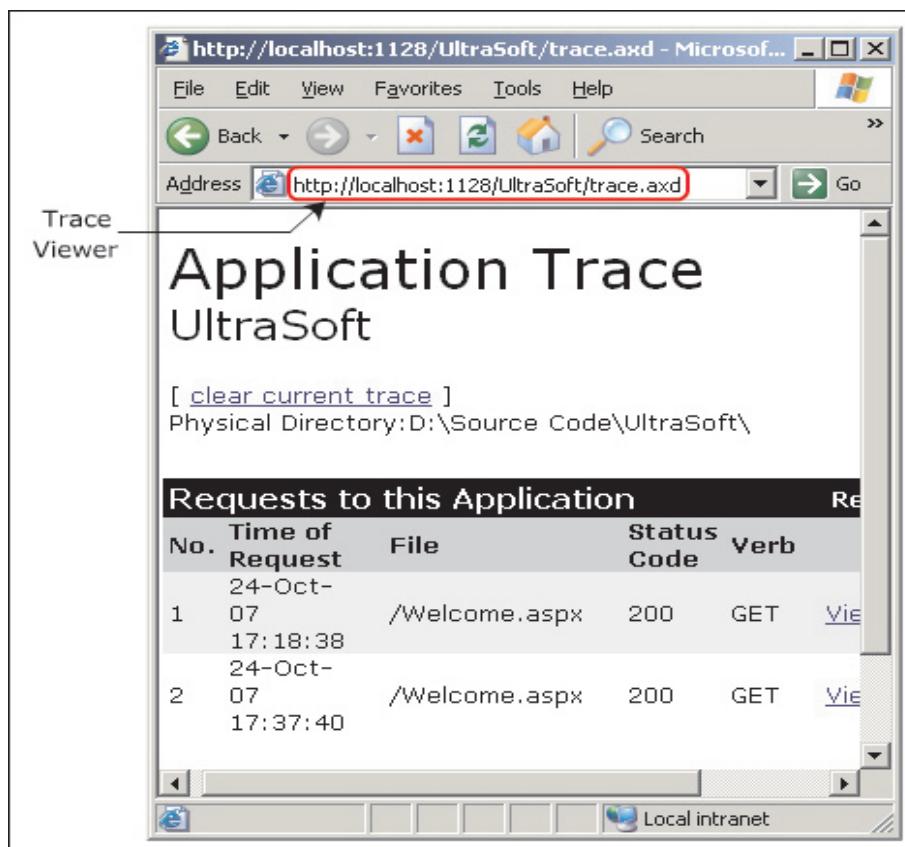


Figure 12.8: View Trace Information

12.2.6 New Features in Tracing

ASP.NET provides tracing functionality for its applications to record all activities performed on the Web pages. Some of the new important tracing features provided by ASP.NET are as follows:

- Performance Counters
- ASP.NET Trace
- ASP.NET Error Handling
- Health Monitoring with Web Events
- Event Log
- Performance Counters

ASP.NET provides a number of performance counters that are used to monitor the performance of the application. A performance counter is the first step for diagnosing and predicting the errors. This feature gives a better idea about the performance and the functional issues concerned with the application.

→ **ASP.NET Trace**

An ASP.NET trace is generated when the ASP.NET page is executed. It traces details about the request, the page control tree, various stages of the page lifecycle, and custom messages. The ASP.NET Trace feature is used for debugging during application development.

→ **ASP.NET Error Handling**

ASP.NET error handling feature allows tracking and reporting errors. This feature is used to control the information that is shown to the client when an error occurs, save the error, report to the administrator, and so on.

→ **Health Monitoring with Web Events**

The Web event feature allows generation of events during the application's lifetime. ASP.NET 2.0 uses the Web event feature to manage a number of system events like custom application events to provide useful state information about the ASP.NET application. This is useful in monitoring the performance of an application to ensure that it is healthy and also useful in instantly diagnosing applications that are failing.

→ **Event Log**

Conditions such as inaccessibility of required resources or inability to create worker processes to host ASP.NET applications cause fatal failures in the ASP.NET infrastructure. In such cases, ASP.NET writes messages to the event log. These messages can be analyzed to find solutions to the problems causing the fatal failures.

Knowledge Check 1

- Which of the following statements regarding tracing are true and which statements are false?

(A)	Tracing in ASP.NET refers to the process of monitoring the execution of ASP.NET Web applications.
(B)	Application-level tracing can be disabled directly in the <code>Web.Config</code> file by adding the <code><trace></code> node or setting it to false.
(C)	To view the trace information from the remote computer, the <code>enabled</code> attribute is set to <code>true</code> .
(D)	<code>BooleanSwitch</code> acts as the toggle switch that is used for enabling or disabling the tracing output.
(E)	ASP.NET error handling feature allows tracking and reporting errors.

(A)	A, D, E	(C)	A, B
(B)	A, C	(D)	C, D, E

12.2 Debugging

In this second lesson, **Debugging**, you will learn to:

- Describe briefly the need and process of debugging.
- Explain how to configure ASP.NET application for debugging.
- Describe how to debug Web forms.

12.2.1 Introduction to Debugging

An application code may, at times, contain errors and bugs. Errors may be either runtime errors or syntax errors. Debugging is a process wherein the errors are located and eliminated. Debugging is, therefore, an essential phase of application development. In ASP.NET Web applications too, debugging is an important activity.

Errors can be classified into three basic categories. These are as follows:

- Syntax Errors
- Crashing Semantic Errors
- Non-Crashing Semantic Errors

Figure 12.9 depicts these types using some examples.

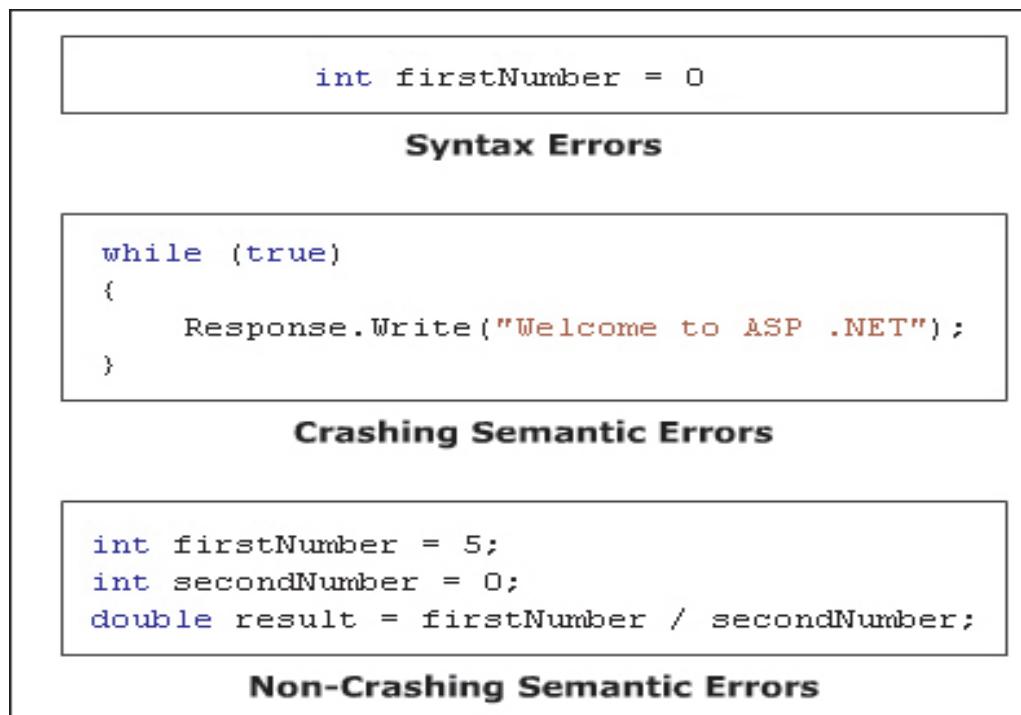


Figure 12.9: Debugging

These are described as follows:

→ Syntax Errors

They are the most common errors that occur due to faulty techniques of writing codes. For example, while writing a code in C#, you forget to declare the parameters for the in-built methods. This mistake would lead to a syntax error. Syntax errors are caught by the compilers and are the easiest errors to debug.

→ Crashing Semantic Errors

This error occurs when the syntax of the code is correct but the conditions declared within the code cause the program to hang or terminate. Depending on the condition that results in the termination of the program, an error message appears indicating the cause of the error and the line number where the error occurred.

→ Non-Crashing Semantic Errors

This error occurs when the syntax of the code and the conditions declared are correct but the variables that contain data are outside the expected range or the program is executing in an unexpected manner. This is the most difficult error to debug.

12.2.2 Debugging During Development

ASP.NET applications can be configured to enable and allow debugging. Once an application is created, then, in the right-hand-side pane of the **Property Pages** dialog box, ASP.NET is selected in the **Debuggers** section. Debugging can be done either using **ASP.Net Web Application Administration** page or using the **Web.config** file.

→ Debugging using ASP.Net Web Application Administration Page

After an ASP.NET Web application is created, the **ASP.NET Configuration** option is selected from the **Website** menu in the IDE. This displays the **ASP.Net Web Application Administration** page. Now, the **Application** tab is clicked. From the **Debugging and Tracing** section, the **Configure debugging and tracing** option is clicked. The **Enable debugging** checkbox is selected in the **Configure the settings for debugging and tracing your application** section (by default, this checkbox is selected).

This approach of debugging does not require any coding for enabling debugging and is completely GUI based.

→ Debugging Using Web.config File

Debugging is enabled in the **Web.config** file using the following steps:

- 1) From the **Solution Explorer** window, the **Web.config** file is opened using an XML parser or any standard text editor. In the absence of **Web.config** file, the root directory of that application is opened. For security reasons, the ASP.NET engine configures with the IIS to prevent direct access to the **.config** files. If anyone tries to access the file using a browser, an HTTP access error occurs.
- 2) **Web.config** is an XML file containing nested sections marked using tags. Inside the file, the **<compilation>** tag is to be located that marks the beginning of the compilation. Within the **<compilation>** tags, the **debug** attribute is added, which is set to **true**.
- 3) If the **debug** attribute is not set to **true** and an attempt is made to debug an application, a dialog box appears that offers to create a **Web.config** file. The offer is accepted and debugging continued.

This technique of debugging helps in creating a secured Web application as the **Web.config** file is not accessible unless the user is given the access.

The following code demonstrates how to enable debugging through `Web.config` file.

Code Snippet:

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <compilation debug="true"/>
    </system.web>
</configuration>
```

In this code, the first line indicates the XML version is set to 1.0. The `<configuration>` is the root element and the `<system.web>` is the sub-element that is added to the `<configuration>` element. Between the `<system.web>` element tags is the `<compilation>` element. In the `<compilation>` element, the `debug` attribute is set to `true` to enable debugging.

12.2.3 Introduction to Listeners

A mechanism is required for collecting and tracing messages in order to use the `Trace`, `Debug`, and `TraceSource` classes. Trace messages are received by the Trace Listeners and debug messages are received by the Debug Listeners. Trace and Debug Listeners are classes used to collect, store, display, and route the trace and debug messages. They direct the trace output and the error messages to the appropriate target such as a window, log, console, or text file.

Following are some of the important Listener classes:

- ➔ **WebPageTraceListener**
- ➔ **TextWriterTraceListener**
- ➔ **DefaultTraceListener**
- ➔ **ConsoleTraceListener**
- ➔ **WebPageTraceListener**

The `WebPageTraceListener` class is used for forwarding the trace messages written in the trace log to the ASP.NET output. Trace forwarding can be enabled by adding the object of the `WebPageTraceListener` class to the `Web.config` file or using the `Application_Start()` method in the `Global.asax` file.

- ➔ **TextWriterTraceListener**

The `TextWriterTraceListener` class redirects the tracing output to the object of the

`TextWriter` class. This class can be used to display the tracing output on the console or direct the output to the specified file.

→ **DefaultTraceListener**

The `DefaultTraceListener` class is used to send messages to the `OutputDebugString` function and `Debugger.Log()` method, causing the debugging messages to appear in the output window of the Web application. This is the default behavior for the debug and trace messages as `DefaultTraceListener` is the only listener automatically added to the listeners' list.

→ **ConsoleTraceListener**

The `ConsoleTraceListener` class directs the tracing and the debugging outputs. These outputs are displayed on the console using the `Console.Out` and `Console.Error` properties.

12.2.4 Steps to Debug a Web Form

The primary activities to be performed in order to debug Web Forms are as follows:

→ **Adding Breakpoints**

Breakpoints must be set before debugging the ASP.NET Web application. A breakpoint is a point in the source code at which the application execution halts temporarily. It can be set by clicking the left margin of the **Code Editor**.

→ **Starting the Debugging Process**

The first step to begin debugging a Web application is to select the start page of the Web application. This is the page that will load when the application executes. Once this page has been selected, you need to enable debugging either through the Website Administration tool or the `Web.config` file.

→ **Setting up Watches for Monitoring Values**

The next step is to add watches in order to monitor values during program execution. The **Watch** window (under **Debug** menu, the **Windows** option) is used to add and manipulate watches.

→ **Removing Breakpoints and Stopping Debugging**

In the `Default.aspx.cs` window, the red dot in the left margin is clicked to remove the breakpoint. The **Stop Debugging** option is clicked from the **Debug** menu to stop debugging.

Knowledge Check 2

- Which of the following statements regarding the need for debugging and configuring ASP.NET for debugging are true and which statements are false?

(A)	A process in which the errors are located and eliminated is called debugging.
(B)	TextWriterTraceListener class is used to collect, store, and route the trace messages to the trace output.
(C)	A process for debugging can be started by adding the DEBUG attribute in the <compilation> tag.
(D)	The trace messages written in the trace log are forwarded to ASP.NET output by using WebPageTraceListener class.
(E)	A syntax error occurs when the code is correct but the condition declared within the code causes the program to terminate.

(A)	A, D	(C)	A, B
(B)	A, C	(D)	C, D, E

12.3 Error Handling

In this last lesson, **Error Handling**, you will learn to:

- Explain the process of error handling in Web applications.
- Describe application level error handling.
- Describe page level error handling.

12.3.1 Error Handling

Different types of errors can occur when working with Web applications or surfing through Web pages. Broken link is a common error generally occurring in large Web applications. This error occurs when a Web page in the application has been moved but the appropriate changes have not been made to the application to re-establish the links.

Various techniques are implemented by the developers to handle errors that can occur when running a Web application. Most errors are handled using the try-catch block. Errors are generally handled using exception handlers or by redirecting the user to another page.

Errors can be either page-level errors such as syntax errors or application-level errors such as missing pages.

Figure 12.10 demonstrates the concept.

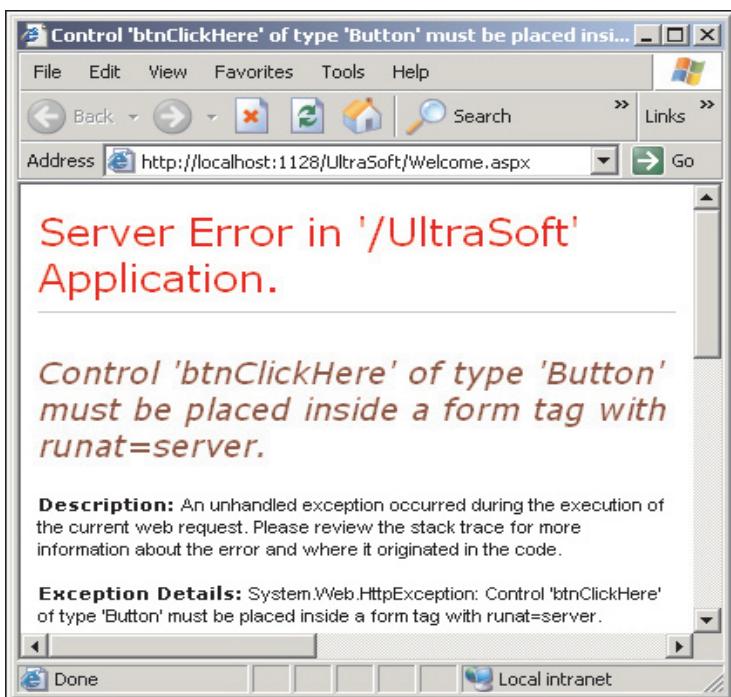


Figure 12.10: Error Handling

12.3.2 Properties of `System.Exception` Class

Exception handling is a mechanism for performing appropriate actions when errors occur while working with Web applications. `Exception` class is a member of `System` namespace. This class represents errors occurring when executing a Web application.

Following are some of the important properties of `System.Exception` class:

- ➔ `HelpLink`
- ➔ `InnerException`
- ➔ `Message`
- ➔ `Source`
- ➔ `TargetSite`

Figure 12.11 shows some of the properties of the System.Exception class.



Figure 12.11: Properties of System.Exception

→ **HelpLink**

The `HelpLink` property specifies or retrieves a link to the help file associated with the exception. When an exception occurs, the user can get help information related with that exception using `HelpLink` property of `Exception` class.

The following code demonstrates the use of the `HelpLink` property of `System.Exception` class.

Code Snippet:

```

// Retrieves an exception related help file link and displays
// it using Response.Write().

Response.Write("Help Link: " + objEx.HelpLink);
  
```

In this code, `HelpLink` property retrieves a URL string related to the exception and displays the message `Help Link` followed by the URL string using `Response.Write()` method.

→ **InnerException**

The `InnerException` property retrieves the instance of the `Exception` class that led to the current exception. This property is useful in situations where one exception causes another exception to occur. The `InnerException` property of the second exception contains a reference to the first exception.

The following code demonstrates the use of `InnerException` property of the `System.Exception` class.

Code Snippet:

```
// Retrieves the inner exception instance that caused the
// current exception and displays it using Response.Write().

Response.Write("Inner Exception is: " + objEx.InnerException);
```

In this code, the `InnerException` property is used to retrieve the current exception and display the appropriate message using the `Response.Write()` method.

→ **Message**

The `Message` property retrieves the message associated with the exception that has currently occurred. This message is useful in understanding the reason for the occurrence of the exception.

The following code demonstrates the use of the `Message` property of the `System.Exception` class.

Code Snippet:

```
// Retrieves a message that describes the current exception
Response.Write("Error message: " + objEx.Message);
```

In this code, the `Message` property retrieves a message describing the current exception and displays it using `Response.Write()` method.

→ **Source**

The `Source` property specifies or retrieves the name of the application or object that has caused the error. This property is used when you wish to provide additional information regarding the object throwing the exception.

The following code demonstrates the use of the `Source` property of the `System.Exception` class.

Code Snippet:

```
// Retrieves the name of the application that causes the error
Response.Write("Source: " + objEx.Source);
```

In this code, the `Source` property retrieves the name of the application that causes the error and displays the appropriate message using the `Response.Write()` method.

→ **TargetSite**

The `TargetSite` property retrieves the method that throws the exception that has currently occurred. This helps in the process of debugging the application as you can identify the method generating the error and do the necessary modifications in the code to resolve the problem.

The following code demonstrates the use of the `TargetException` property of the `System.Exception` class.

Code Snippet:

```
// Retrieves the method that throws the current exception
Response.Write("Target Site: " + objEx.TargetSite);
```

In this code, the `TargetException` property retrieves the method that throws an exception and displays the appropriate message using the `Response.Write()` method.

12.3.3 Application-Level Error Handling

Application-level error handling is needed if the error can occur anywhere in the whole application. If a page encounters an unhandled error, the error is automatically handled by the application-level error handling mechanism.

The error handling configuration is stored in `web.config` file. A `web.config` file in ASP.NET 2.0 stores the configurations required for the corresponding application.

To handle an error at the application level, the attribute `defaultRedirect` is added to the `<customErrors>` element, which specifies the default error page, and the `mode` attribute is set to `On`.

Figure 12.12 demonstrates the concept.

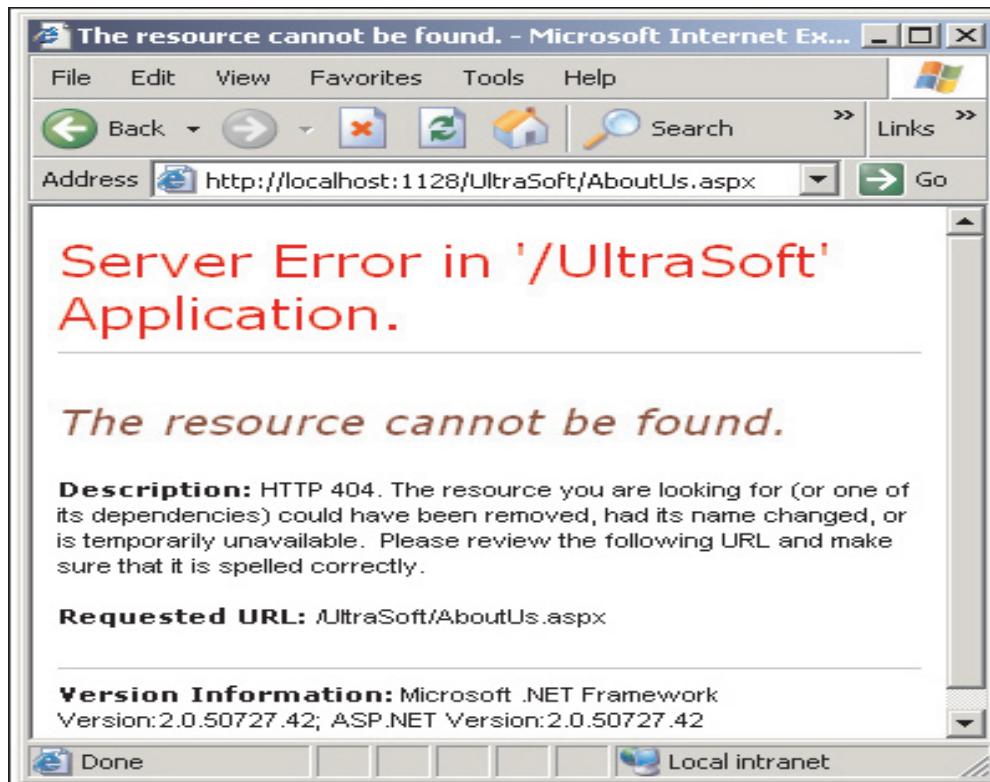


Figure 12.12: Application Level Error Handling

The following code demonstrates application-level error handling by adding the code to the `Web.config` file.

Code Snippet:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <customErrors mode="On" defaultRedirect="error.htm" />
  </system.web>
</configuration>
```

In this code, the XML version is set to 1.0. The `customErrors` element is added between the `<system.web>` element tags. In `customErrors` element, the `mode` attribute is set to `On`, which specifies that the custom error pages are always shown. The `defaultRedirect` attribute is set to `error.htm` and it specifies the path to a generic error page.

12.3.4 Page-Level Error Handling

In a Web application, such as Amazon.com, there can be large number of Web pages. Sometimes, while traversing through these pages, chances are there that on a particular Web page, an unhandled exception may occur. To handle such errors occurring on a specific page, page-level error handling is done.

In a Web page, when an exception goes unhandled, the `Error` event of the `Page` class is fired. First the page-level error handler retrieves the exception using the `Server.GetLastError` method. This method retrieves a reference to the last `Exception` object that was thrown. Once the `Exception` object is retrieved, the user is redirected to the error page. Redirection is done either using the `errorPage` attribute of the `Page` class (design time) or by using the `Page.ErrorPage` property (runtime).

Figure 12.13 demonstrates the concept.



Figure 12.13: Page Level Error Handling

The following code demonstrates page-level error handling.

Code Snippet:

```
<%@ Page language="c#" Codebehind="CustomerDetails.aspx.cs"
AutoEventWireup="false" Inherits="CustomerDetails" errorPage="PageError.
aspx"%>
```

In this code, the @ Page directive's errorPage attribute is set to PageError.aspx, which will redirect the user to the page when an unhandled exception occurs in that page.

Knowledge Check 3

- Can you match the properties of System.Exception class with their corresponding descriptions?

Description		Property	
(A)	Retrieves the method that throws the current exception.	(1)	Source
(B)	Retrieves the instance of Exception class.	(2)	HelpLink
(C)	Retrieves the message that explains the current exception.	(3)	TargetSite

(D)	Specifies or retrieves the name of the application or object that caused the error.	(4)	Message
(E)	Specifies or retrieves the link to the help file.	(5)	InnerException

(A)	(A)-(3), (B)-(5), (C)-(4), (D)-(1), (E)-(2)	(C)	(A)-(2), (B)-(3), (C)-(5), (D)-(3), (E)-(1)
(B)	(A)-(3), (B)-(5), (C)-(1), (D)-(4), (E)-(2)	(D)	(A)-(4), (B)-(1), (C)-(2), (D)-(5), (E)-(3)

2. Which one of the following codes will display error pages for both, default errors as well as HTTP status code errors?

(A)	<pre><configuration> <system.web> <customErrors mode="On" defaultRedirect="PageError.aspx"/> <error statusCode="204" redirect="NoResponse.aspx"/> <error statusCode="400" redirect="BadRequest.aspx"/> </customErrors> </system.web> </configuration></pre>
(B)	<pre><configuration> <system.web> <customErrors mode="On" defaultRedirect="PageError.aspx"> <error statusCode="204" redirect="NoResponse.aspx"/> <error statusCode="400" redirect="BadRequest.aspx"/> </customErrors> </system.web> </configuration></pre>

(C)	<pre><configuration> <system.web> <customErrors mode="On" defaultRedirect="PageError.aspx"> <error statusCode="204" redirect="NoResponse.aspx"> <error statusCode="400" redirect="BadRequest.aspx"> </customErrors> </system.web> </configuration></pre>
(D)	<pre><configuration> <system.web> <customErrors mode="On" defaultRedirect="PageError.aspx"> <error statusCode="204" redirect="NoResponse.aspx"> <error statusCode="400" redirect="BadRequest.aspx"> </customErrors> </system.web> </configuration></pre>

(A)	B	(C)	A
(B)	C	(D)	D

Module Summary

In this module, you learnt about:

→ **Tracing**

Tracing in ASP.NET refers to the process of monitoring the execution of ASP.NET Web applications. Tracing can also involve recording exceptions during runtime. Tracing can be performed on a single Web page or the entire ASP.NET Web application.

→ **Debugging**

Debugging is the process of detecting and removing errors in the ASP.NET code. Errors can be either syntax errors, crashing semantic errors, or non-crashing semantic errors. Debugging in ASP.NET can be done either using the **ASP.Net Web Application Administration** page or using the `Web.config` file. Trace listener classes are used to collect, store, and route debugging and tracing messages.

→ **Error Handling**

Exception handling is the process of performing appropriate actions when errors occur while working with Web applications. Error handling can be done either at the application level or at page level. Application-level error handling is needed if the error can occur anywhere in the entire application. Page-level error handling involves handling the unhandled exception errors on a specific ASP.NET Web page.

Module - 13

Introduction to ASP.NET 3.5

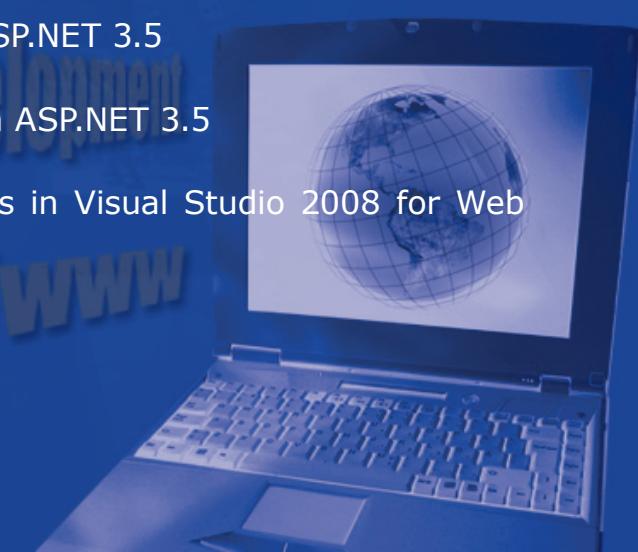
Welcome to the module, **Introduction to ASP.NET 3.5**.

Microsoft ASP.NET is one of the most popular platforms for developing Web applications. ASP.NET 2.0 added many new features as compared to the previous version ASP.NET 1.1. With Microsoft ASP.NET 3.5, there are several more features added that facilitate you to create rich and complex Web applications. This module will cover some of these features such as new controls and assemblies, and new features in the Integrated Development Environment (IDE) for ASP.NET Web applications.

The module also explores event procedures, page events, and control events.

In this module, you will learn to:

- Describe the standard controls in ASP.NET
- Describe event procedures in ASP.NET
- Describe page and control events
- Explain the new features in ASP.NET 3.5
- List the new controls in ASP.NET 3.5
- List the new assemblies in ASP.NET 3.5
- Describe the new features in Visual Studio 2008 for Web applications



13.1 Introduction

Microsoft ASP.NET is one of the most popular platforms for developing Web applications.

13.2 Standard Controls in ASP.NET

ASP.NET provides a number of standard controls. Some of the controls such as `HyperLink` and `Wizard` are explored in the following sections.

13.2.1 HyperLink Control

The `HyperLink` control enables you to link a number of Web pages to one another in a Web site. In other words, if you want to navigate in a Web site from one page to another page, you can use the `HyperLink` control.

Table 13.1 lists some of the properties of the `HyperLink` control.

Property	Description
<code>ImageUrl</code>	Retrieves or sets the path of the image that may be displayed with the <code>HyperLink</code> control
<code>NavigateUrl</code>	Retrieves or sets the destination Uniform Resource Locator (URL) to which the page will navigate when the <code>HyperLink</code> control is clicked
<code>Target</code>	Retrieves or sets the target frame or window in which you can display the Web page contents of the destination URL
<code>Tooltip</code>	Retrieves or sets the text to be displayed when the mouse pointer is moved over the Web server control
<code>Text</code>	Retrieves or sets the caption for the control

Table 13.1: Properties of Hyperlink Control

The following steps enable you to add a `HyperLink` Web server control to a Web Forms page:

1. Add a `HyperLink` control to the form on your Web page.
2. Specify the format of the link in the page by doing one of the following:
 - Select the `Text` property if you are creating a text link and include Hypertext Markup Language (HTML) formatting in the property.
 - Select the `ImageUrl` property of the control if you are creating a graphic link and set it to a .gif, .jpg, or other graphic file formats.
3. Select the `NavigateUrl` property and set it to the URL of the page to which you want to link.
4. Set the id of a target frame in which you want to display the linked page.

The following code snippet will display a `HyperLink` image. When you click the link at runtime, the specified URL will be displayed.

Code Snippet:

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

    <h3>
        HyperLink
    </h3>

    Click here!!!
    <br />

    <asp:HyperLink id="lnkSearch"
        ImageUrl="~/HyperLink.jpg"
        NavigateUrl="http://www.google.com"
        Text="Google- Search Engine"
        Target="new_Search"
        runat="server">
    </asp:HyperLink>

</form>
```

The code defines a **HyperLink** control containing an **ImageUrl** property. The **NavigateUrl** property sets the destination URL. The output is seen in figure 13.1.

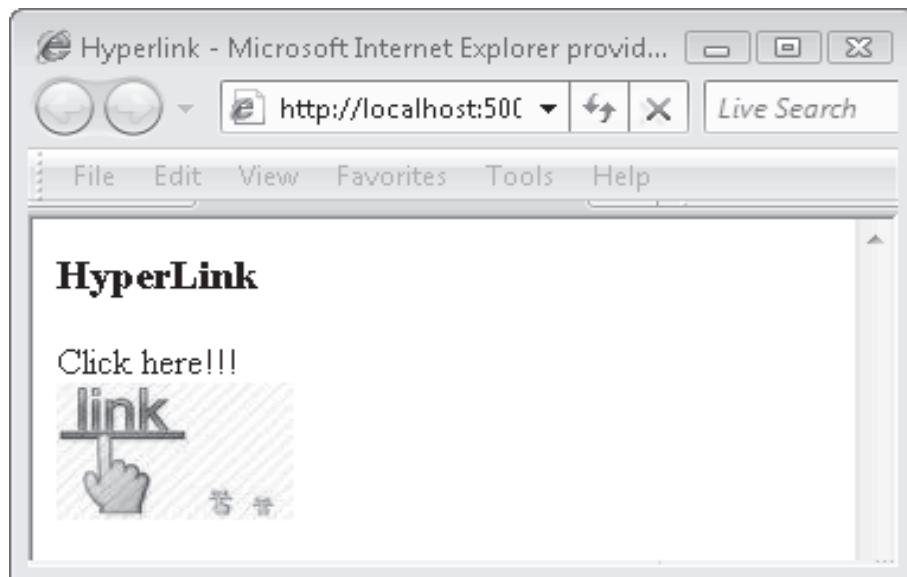


Figure 13.1: Using the HyperLink Control

In figure 13.1, a Web page is displayed with a link.

When you click the link, it will navigate to the specified URL as shown in figure 13.2.

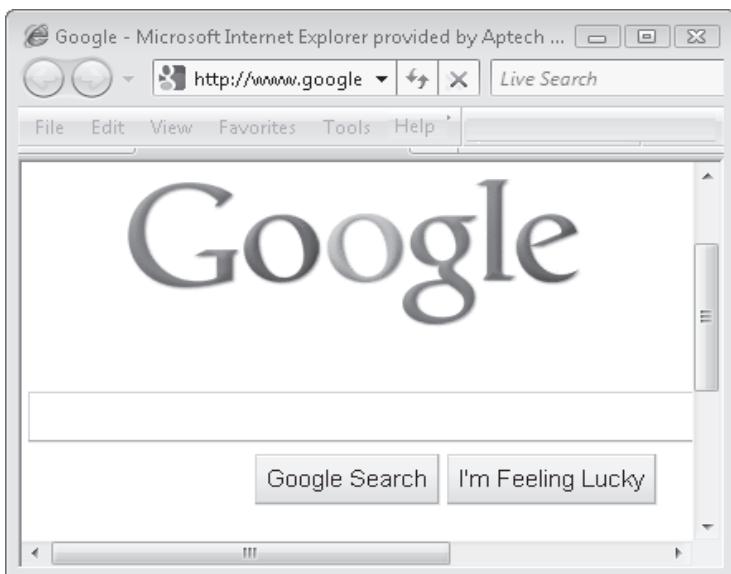


Figure 13.2: Result of Clicking the HyperLink

13.2.2 Wizard Control

The `Wizard` control is used to create a sequence of forms to gather a wide range of data from users. This control allows you to create a new step, add a new step, or reorder the steps in a Web site. The control also simplifies the collection of data as a chain of independent steps without writing code or persisting the data across the steps. Table 13.2 lists some of the properties of the `Wizard` control.

Property	Description
<code>ActiveStep</code>	Retrieves the current step from the <code>WizardSteps</code> collection that is displayed
<code>ActiveStepIndex</code>	Retrieves or sets the value of the current <code>WizardStepBase</code> object
<code>StepStyle</code>	Retrieves an instance to a <code>Style</code> object that describes the settings for <code>WizardStep</code> object
<code>WizardSteps</code>	Retrieves the collection that consists of <code>WizardStepBase</code> objects described for the control

Table 13.2: Properties of a Wizard control

Consider a scenario where you want to create a Web application that collects the candidate information such as personal, professional and finished steps. The following code snippet shows the example of a `Wizard` control designed for this scenario:

Code Snippet:

```
<asp:Wizard ID="wizInfo" runat="server" Width="330px"
    ActiveStepIndex="0" Height="230px"
    onfinishbuttonclick="Wizard1_FinishButtonClick">

<WizardSteps>
    <asp:WizardStep ID="wizStepone" runat="server" title="Personal">
        Name:
        <asp:TextBox ID="txtName" runat="server" />
        &nbsp;&nbsp;&nbsp;<br /><br />
        Address:
        <asp:TextBox ID="txtAddress" runat="server"
            style="z-index: 1; left: 155px; top: 101px; position: absolute" />
        <br />
        <br />
    </asp:WizardStep>
    <asp:WizardStep ID="wizSteptwo" runat="server" title=" Profession">
        Designation:
        <asp:TextBox ID="txtDesignation" runat="server" />
        <br />
        <br />
        Status:
        <asp:TextBox ID="txtStatus" runat="server"
            style="position: relative; top: 4px; left: 31px" />
    </asp:WizardStep>
    <asp:WizardStep ID="wizStepthree" runat="server" StepType="Complete"
        Title="Finished">
        <asp:Label ID="lblName" runat="server" Text="Label">
        </asp:Label>
        <asp:Label ID="lblAddress" runat="server" Text="Label">
        </asp:Label>
```

```
<asp:Label ID="lblDesignation" runat="server" Text="Label">
</asp:Label>

<asp:Label ID="lblStatus" runat="server" Text="Label">
</asp:Label>

</asp:WizardStep>

</WizardSteps>

</asp:Wizard>
```

Code Snippet 2 will display a Wizard control with three steps Personal, Profession, and Finished. When you click the Personal link, the code will display two labels and text boxes to accept the name and address of the candidate. When you click the Profession link, it will display two labels and text boxes to accept the designation and the status of the candidate. Figure 13.3 displays the output of Code Snippet 2.

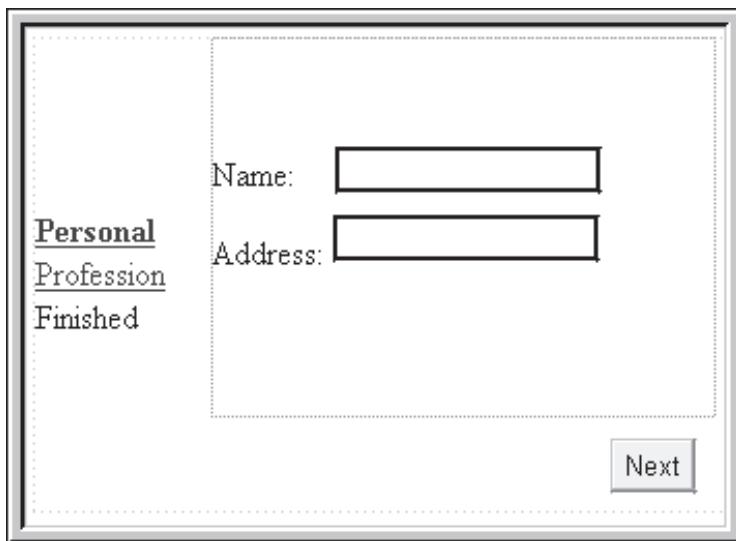


Figure 13.3: Using the Wizard Control

13.3 Event Procedures

Event procedures are used for handling user interactions on the Web page. When you click a button on a Web form, an event occurs. Event procedures are the actions that occur in response to the caused event. Consider a scenario of an online registration form for an airlines company. You have to fill in the appropriate details and then, click the Register button. When you click the button, the event procedure will send the entered data to the database server. Event procedures can be divided into two types namely, client-side and server-side event procedures.

13.3.1 Client-side Event Procedures

The client-side event is the event that is handled on the client machine. When an event occurs, no data is sent to the server; instead, the browser reads the code and performs the necessary action.

The client side procedures are used only with HTML controls.

These procedures do not have access to server-side data. The advantage of these procedures is that they do not have to wait for the response from the server. For example, if you want to validate the data in the TextBox before it is submitted to the server, you can write a client-side script. This script will validate the data efficiently and quickly before sending the information for further processing.

The following code snippet shows the client-side event procedure by using a script block in a Web page:

Code Snippet:

```
<form method="post">  
    <script language="JavaScript">  
        alert('Client Side Scripting.');//  
    </script>  
</form>
```

Figure 13.4 shows the output for client-side scripting.

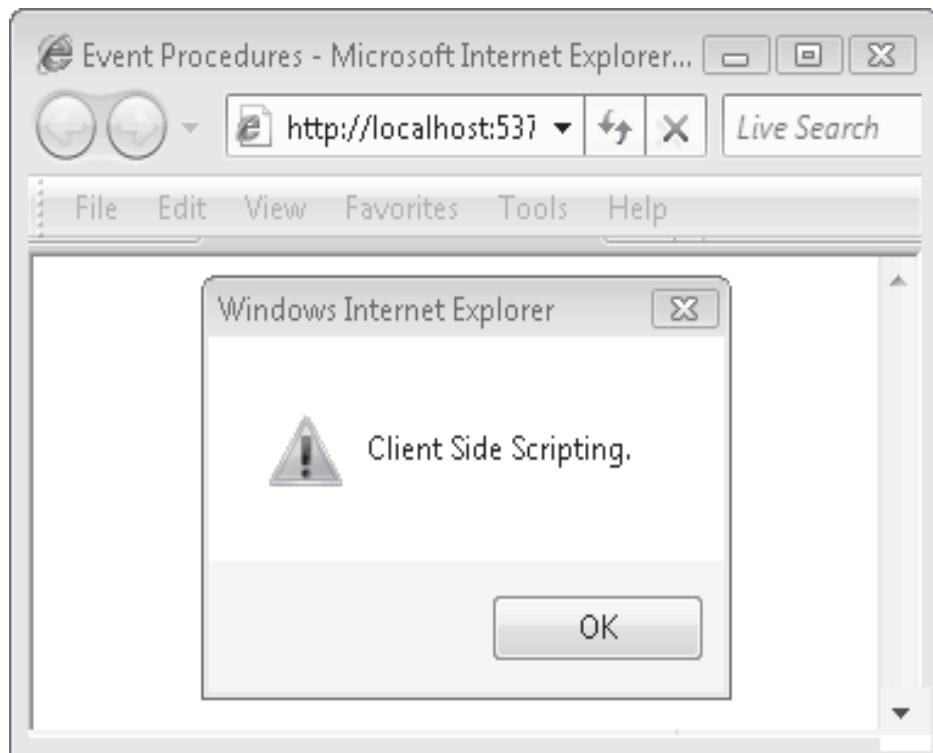


Figure 13.4: Client-side Scripting

13.3.2 Server-side Event Procedures

Server-side event procedures are not validated on the client machine. Instead, they are sent to the Web server for processing. These procedures are much more powerful than the client-side procedures as they consist of the compiled code that is located on the Web server.

You can use these procedures to handle the actions generated by HTML and Web server controls. The procedures can access server resources that are not accessible to the client side procedures.

The following code snippet shows a server-side procedure in the `script` tag using the `runat="server"` attribute:

Code Snippet:

```
<script runat="server">
    void btnDisplay_Click(object sender, EventArgs e)
    {
        Response.Write("Server-Side Scripting");
    }
</script>

<html>
<head>
</head>

<body onload="javascript:document.forms[0]['btnDisplay'].value=Date();">
    <form id="frmDisplay" runat="server">
        <asp:Button id="btnDisplay" onclick="btnDisplay_Click"
            Font-Names="Arial" Font-Italic="True" Font-Size="Medium"
            runat="server" />
    </form>
</body>
</html>
```

Figure 13.5 shows the output for server-side scripting.

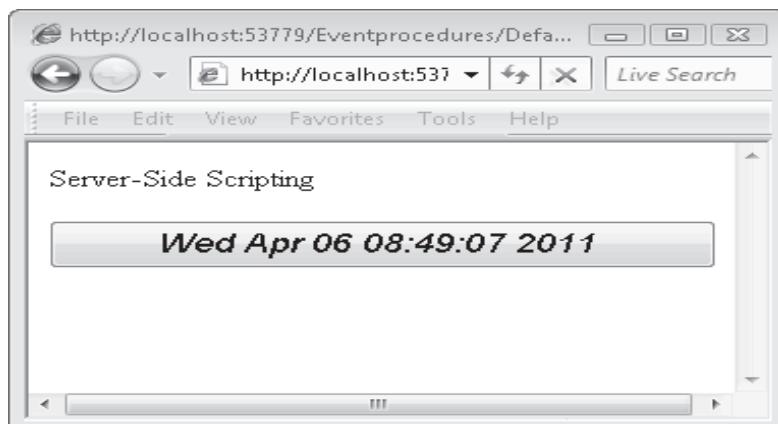


Figure 13.5: Server-side Scripting

13.4 Page Events

In ASP.NET, a page is processed through a number of stages; with every stage, there are several events that are fired for a Web page and the controls on it. In order to handle these events, you need to write the code that will respond to multiple actions on the page. For example, you can write a code that is called when the page is loaded for the first time just to check whether the user has requested the page for the first time or has posted the page back to the server.

`Page_Load` and `UnLoad` are the most important events in the page lifecycle. Table 13.3 lists some of the other events that are fired when a page is processed.

Property	Description
PreInit	This event is used when you want to dynamically set the values for a master page or theme. Master page gives a consistent look and feel across your Web pages. If you are working with dynamic controls, then you need to create these controls inside this event
InitComplete	This event is fired when the page and its controls have completed their initialization
PreLoad	This event is fired before the postback processing and also before the view state is loaded for a particular page
LoadComplete	This event completes the loading of the controls. It also allows additional processing
SaveStateComplete	This event will occur after the view state for the page and the controls are set
Render	This event causes the client-side HTML, Dynamic Hypertext Markup Language (DHTML) and scripts that are required to display the controls to the browser

Table 13.3: Page Events

13.5 Control Events

The `Page` class in ASP.NET is inherited from the `Control` class from which the server controls also inherit. Hence, the server controls share the same lifecycle as that of the `Page` class and have events like `Load`, `Init`, `Render`, and `UnLoad`. When the page `Load` event occurs, at the same time, the event occurs for each and every child control on that page. If the child controls have nested child controls, then, they too perform the load events. These synchronized events will be executed for all the controls that are added to the page at the design time. If the controls are added dynamically, then the events are not executed sequentially.

13.6 New Features in ASP.NET 3.5

ASP .NET 3.5 is the latest version of ASP.NET and is built-in with the .NET Framework version 3.5. It helps you build powerful, rich, and secure Web applications. The .NET Framework version 3.5 recommends the Visual Studio 2008 IDE for developing Web applications. There are many new features added to the IDE such as multi-framework targeting, JavaScript debugging, improved designer experience, and IntelliSense support.

Some of the new features introduced in ASP.NET 3.5 are as follows:

- New server controls, types and client-script library allow you to build Asynchronous JavaScript and XML (AJAX) related Web applications.
- There are extensions of server-based forms authentication, profile services as Web services, and role management that can be used by Web applications.
- The new control called `EntityDataSource` represents the Entity Data Model (EDM) through the ASP.NET data source control architecture.
- The new control `ListView` presents the data and offers a customizable User Interface (UI).
- The new control `LinqDataSource` provides LINQ capabilities.
- The ASP.NET Merge tool is a new command-line tool used for merging the pre-compiled assemblies and maintaining flexible deployment and release management.
- Dynamic data support is a feature that allows you to create Web sites through which you can work with a LINQ to SQL object model and also develop pages manually.

13.7 New Controls in ASP.NET 3.5

There are many new enhancements and additions made to ASP.NET 3.5. An overview of these is included in the following sections.

13.7.1 ASP.NET AJAX

AJAX stands for Asynchronous JavaScript and XML. AJAX is a combination of JavaScript, HTML, Cascading Style Sheets (CSS), XML, Document Object Model (DOM) and `HttpRequest` object. This technology is basically used for sending and receiving data for server-side applications using JavaScript. ASP.NET AJAX is integrated with .NET Framework 3.5. You can use the features of ASP.NET AJAX to build Web applications that will give advantages like partial page updates and so forth. It has client-side and server-side features as well as it contains specific UI components like panels and progress bars.

ASP.NET AJAX comprises the AJAX Control Toolkit, which offers a library of server controls and extenders.

13.7.2 ListView Data Control

`ListView` data control is a new data control introduced in ASP.NET 3.5. This data control has a support for all Create, Read, Update, and Delete (CRUD) operations. It also has support for paging and sorting data using styles and templates. This control provides similar built-in features that are available in `GridView`, but has greater improved control over the output that is rendered.

The ListView control resembles a DataList and Repeater control. The basic controls used with a ListView controls are ItemTemplate and LayoutTemplate.

Figure 13.6 shows an example of a ListView control.

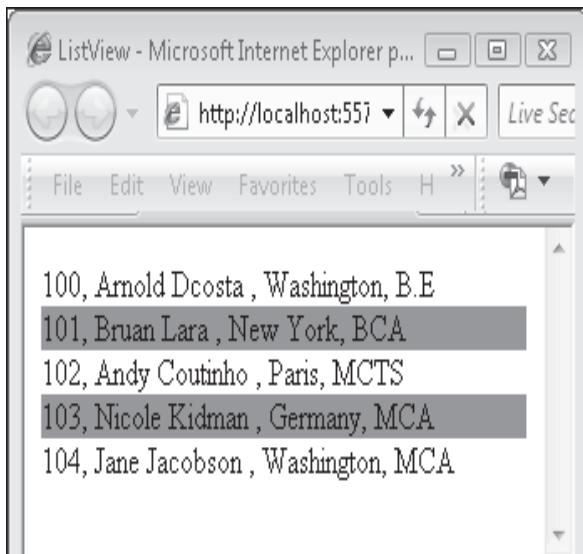


Figure 13.6: ListView Control

13.7.3 DataPager Control

The DataPager control provides paging features. The PagedControlID property allows you to associate the DataPager control with the data-bound control. The DataPager class, used for paging the data and displaying the navigation controls for the data-bound controls which implements the IPageableItemContainer interface. You can place the DataPager control inside the ListView Layout template. Customization of DataPager control can be allowed by changing the properties of PageSize property.

Figure 13.7 displays an example of DataPager control.

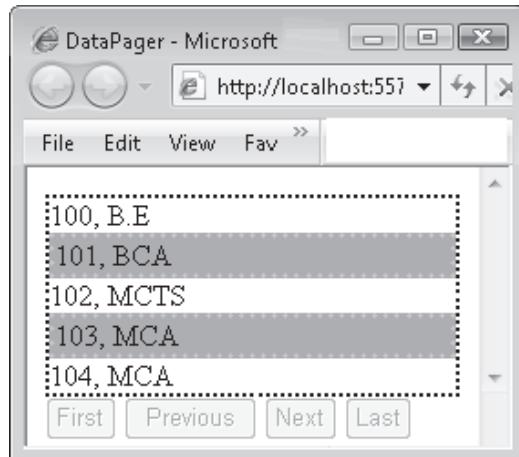


Figure 13.7: DataPager Control

13.7.4 LinqDataSource Control

The `LinqDataSource` control allows you to bind data to ASP.NET data controls by implementing LINQ queries against a dataset. This control is used to execute complex queries and stored procedures. You can use the control to bind the data to a wide range of data sources like custom business objects, collections, or databases. With the LINQ feature that is available with `LinqDataSource`, you can query a data source and also build a data model which enables mapping of data source objects. This control is also used to bind a list-bound control to a data model and show data from the data source without writing any code.

13.7.5 Integration with Internet Information Services (IIS) 7

The .NET Framework 3.5 offers many application programming interfaces that facilitate you to work particularly with IIS 7. It also consists of the `HttpResponse.Headers`, the `HttpServerUtility.TransferRequest`, and the `HttpApplication.LogRequest` events. In short, the `HttpResponse.Headers` allows you to provide information about the HTML document in a meta tag or to collect the information about the document. The `HttpServerUtility.TransferRequest` method carries out asynchronous execution of a particular URL. The `HttpApplication.LogRequest` event fires before ASP.NET does any logging for the present request.

13.8 New Assemblies

The new assemblies introduced in ASP.NET 3.5 are as follows:

- **System.Core:** It provides LINQ to objects implementation.
- **System.Data.Linq:** It provides classes that support interaction with databases in LINQ to SQL applications.
- **System.Xml.Linq:** It provides types for LINQ to XML, which also has an in-memory XML programming interfaces that allow you to modify XML documents easily and efficiently.
- **System.Data.DataSetExtensions:** It provides the implementation for LINQ to DataSet class.
- **System.Web.Extensions:** It provides support for implementing new Web controls and ASP.NET AJAX controls.

13.9 New Features in the IDE for ASP.NET Applications

There are many new features introduced for ASP.NET 3.5 in Visual Studio 2008 IDE for developing Web applications. The introduction of Visual Studio 2008 has made the task of developing powerful and reliable enterprise Web solutions easier than ever before. It has also increased the efficiency of developers by giving them a common and shared development environment. Some features included for Web development are as follows ASP.NET AJAX extensions, multiple project types for Web, and multi-framework targeting.

Visual Studio 2008 IDE also provides JavaScript and CSS IntelliSense support that allows you to create correct code faster. It also has a cross-language, end-to-end, powerful debugging support that is useful to make your applications operational.

13.9.1 ASP.NET AJAX

ASP.NET AJAX is one of the new features introduced in ASP.NET 3.5. The AJAX Extensions tab is included in the Toolbox in Visual Studio (VS) 2008 IDE. You do not have to install the AJAX component separately. If you are creating a new ASP.NET Web site or application in VS 2008 that aims at .NET Framework 3.5, then, Visual Studio will add the suitable markup automatically in the `web.config` file and the ASP.NET AJAX server controls will be displayed in the Toolbox.

Figure 13.8 shows the AJAX Extensions tab in Visual Studio 2008.

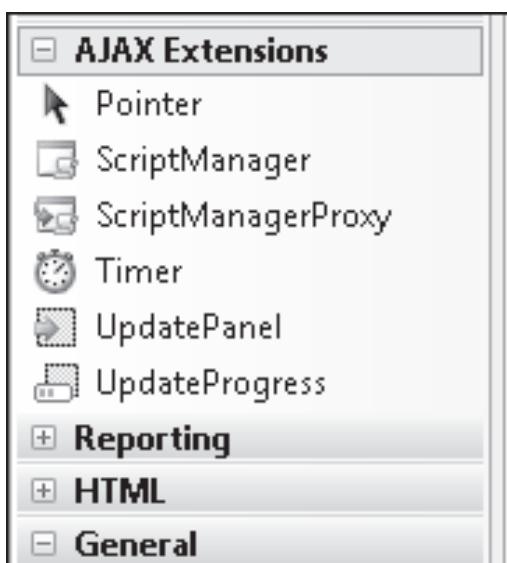


Figure 13.8: AJAX Extensions

ASP.NET AJAX has added a new set of improvements such as support for usage of UpdatePanel controls along with Web Parts and also supports Windows Communication Foundation (WCF) based JavaScript Object Notation (JSON) endpoints. A Web part is a collection of set of controls used for creating Web applications that allows you to change the appearance, content, and behavior of Web forms on the browser. WCF is a framework used for developing service-oriented applications. AJAX feature also has support for the

ASP.NET Role, Profile, and Login application Services that use JavaScript and allow various bug fixing techniques and improvements in performance.

13.9.2 Multiple Project Types

Using Visual Studio 2008 you can develop projects of several types, such as Microsoft Windows-based applications, ASP.NET Web-based applications, and XML Web services. This feature of multiple project types allows developers to work on various projects under a single IDE.

The new project templates introduced in Web project types include ASP.NET AJAX Server Control, ASP.NET AJAX Server Control Extender, and WCF Service Application.

Figure 13.9 shows the new project templates for Web application in Visual Studio 2008.

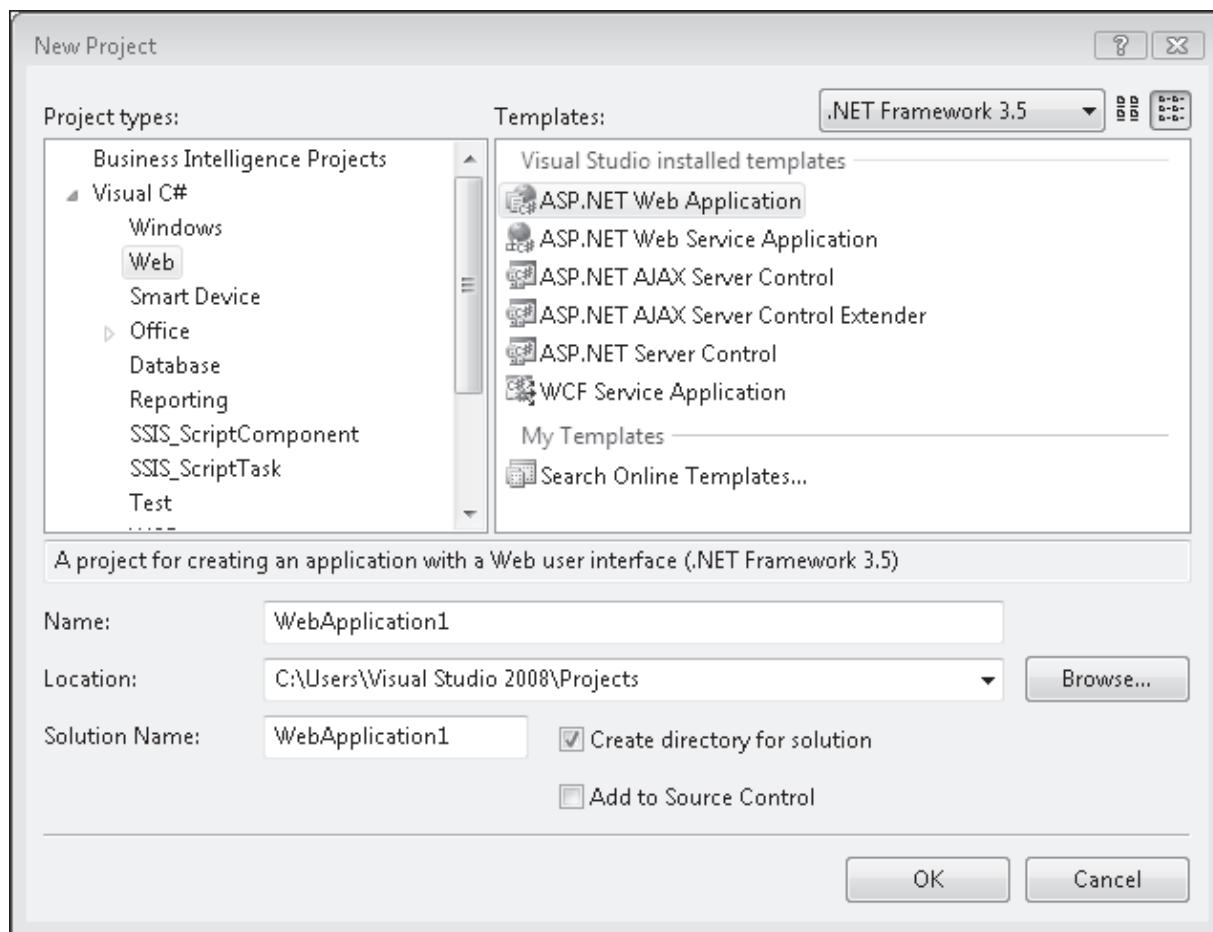


Figure 13.9: ASP.NET New Project Templates

13.9.3 Multi-Targeting Feature

In Visual Studio 2008, you can compile an assembly and allow it to run on multiple versions of .NET Framework (like 2.0, 3.0, and 3.5). By default, it compiles an assembly to run only on .NET Framework 3.5.

Figure 13.10 displays old and new versions of .NET Framework available in Visual Studio 2008.

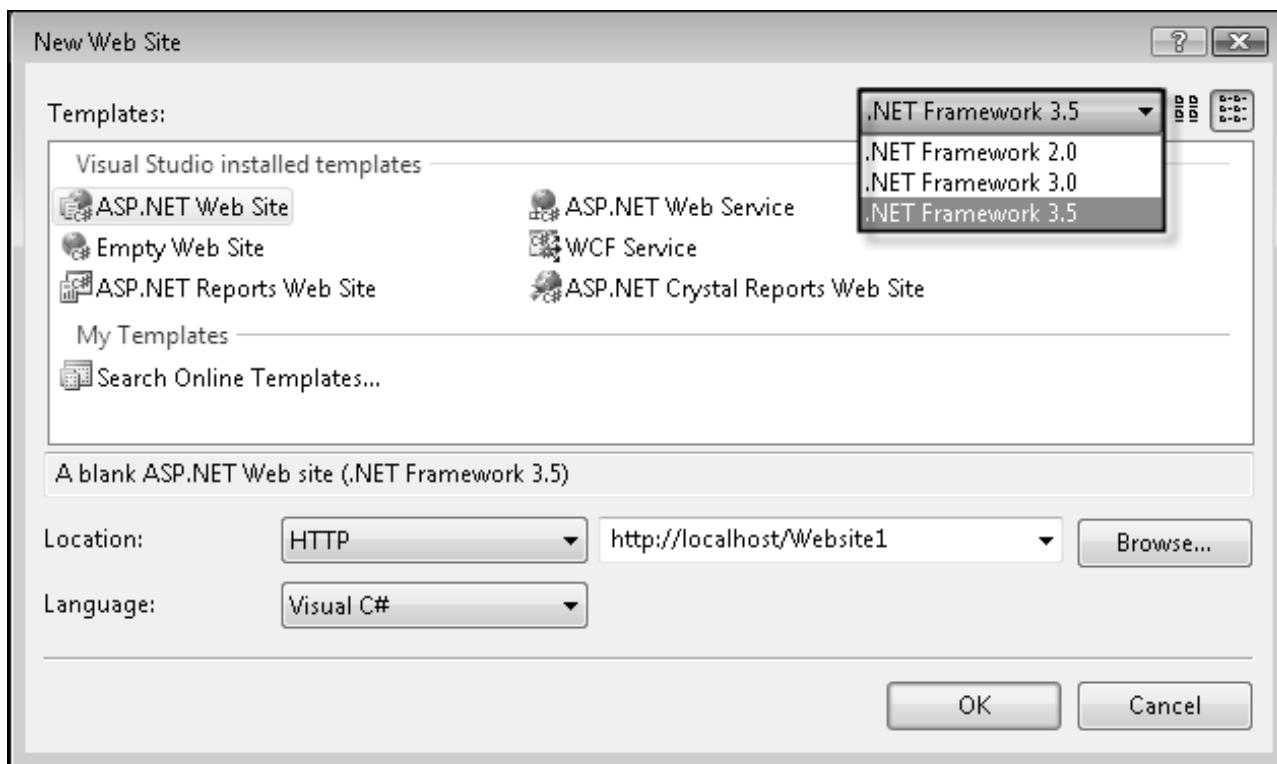


Figure 13.10: Multi-Targeting Feature

13.9.4 Nested Master Page Support

Master page is one of the most prominent features in ASP.NET 2.0. Visual Studio 2008 has now introduced the concept of Nested master page support in ASP.NET 3.5. Now, you can have nested master pages and also have a flexible layout. Using nested master pages you can create a parent master page, a child master page, and another child master page. You can also create Web site templates using this feature. Nested master pages allow developers and designers to make changes to the layout of any online site with minimum of code. This feature also helps you to develop Web applications quicker and faster as compared to previous versions like ASP.NET 2.0.

13.9.5 Enhanced Web Design Experience

In the earlier versions of Visual Studio, developers could select the Design or the Source view where they can view either of the panes at one time. Visual Studio 2008 provides a new feature for enhanced Web design experience named as Split screen mode that displays the markup in one pane and the design in another pane. In Split mode, you can add the content to the Design view and view the automatically updated markup content in the Source view pane.

Figure 13.11 displays the Split View feature in Visual Studio 2008.

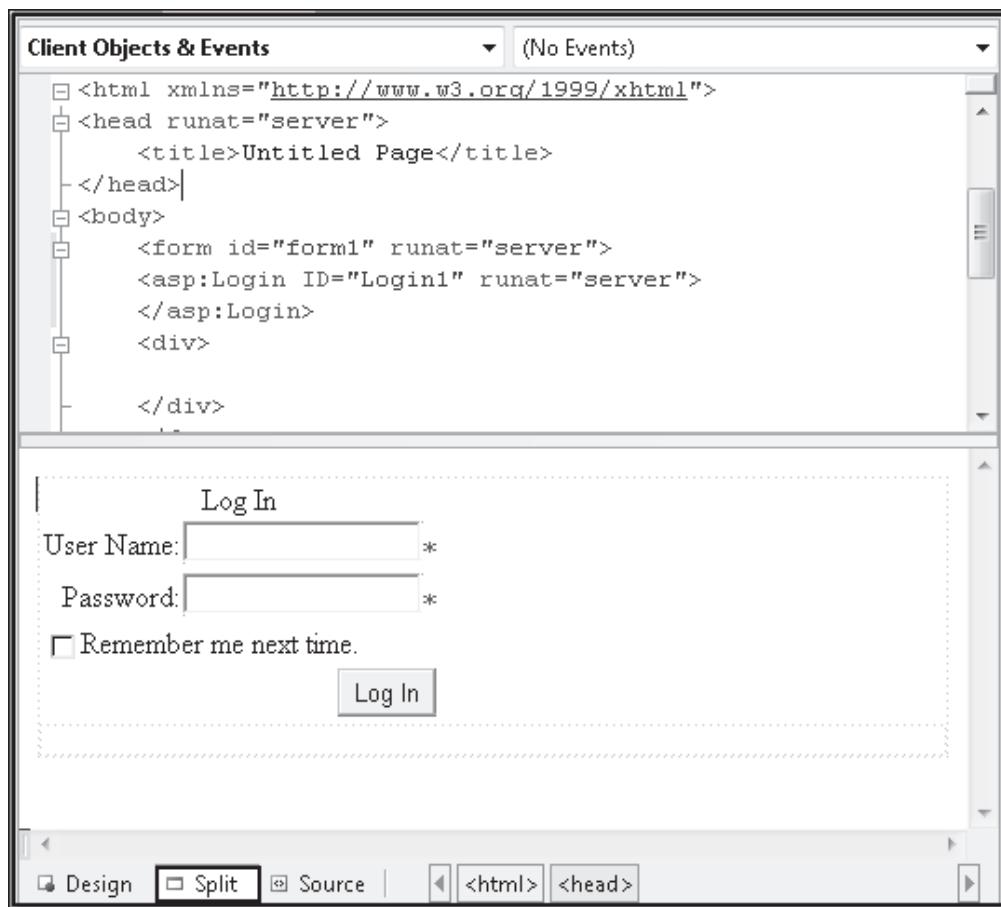


Figure 13.11: Split View feature in Visual Studio 2008

13.9.6 JavaScript IntelliSense Support

Visual Studio 2008 provides IntelliSense support for JavaScript. This feature helps developers to write client side validation in an easy and efficient manner. There is a difference in writing the JavaScript code in VS 2005 as compared to VS 2008 in that you can select the JavaScript keywords and language features from the IntelliSense. This feature also allows you to develop AJAX applications much more easily.

Figure 13.12 depicts an example of IntelliSense support for JavaScript code.

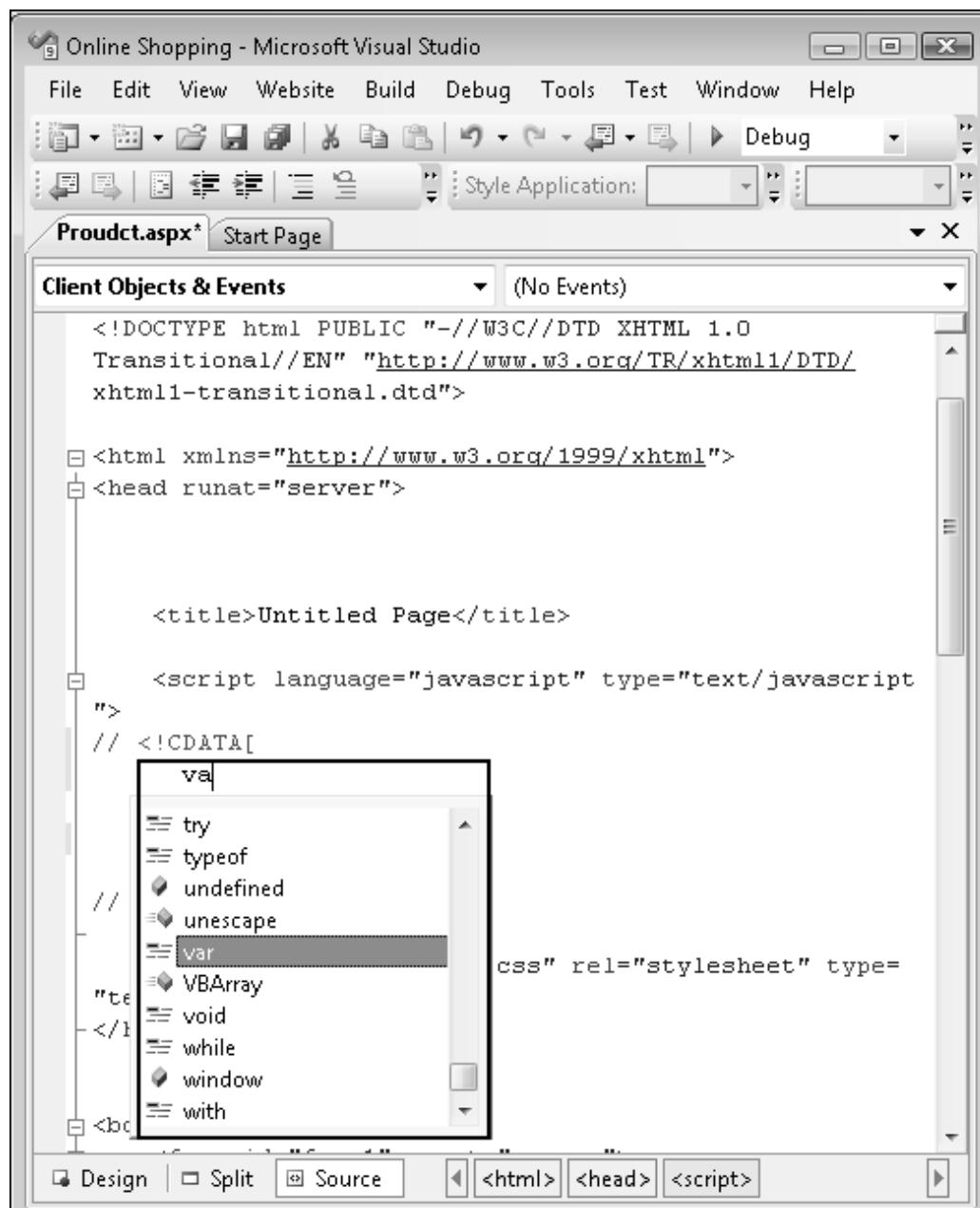


Figure 13.12: IntelliSense support for JavaScript

13.9.7 JavaScript Debugging

JavaScript debugging is a new feature introduced in ASP.NET 3.5. In the previous versions, if you had to debug a JavaScript code, then you had to first execute your ASP.NET application and then, set the breakpoints in the debugger.

Now in Visual Studio 2008, with JavaScript debugging, you can set your JavaScript code in ASP.NET pages, external code files, and even in master pages.

One of the new features added is the enhanced object inspection support that gives you a detailed object property and property type information.

13.9.8 CSS Properties Window

CSS Properties window is also one of the new features added to Visual Studio 2008. This window will display all CSS settings currently applied to any HTML or Web server control. A red line visible through the property name specifies that it has been overridden. You can click the Summary button to see the properties applied to the selected element. This new properties window allows you to keep track and change CSS properties easily and quickly. Figure 13.13 displays the CSS Properties window.

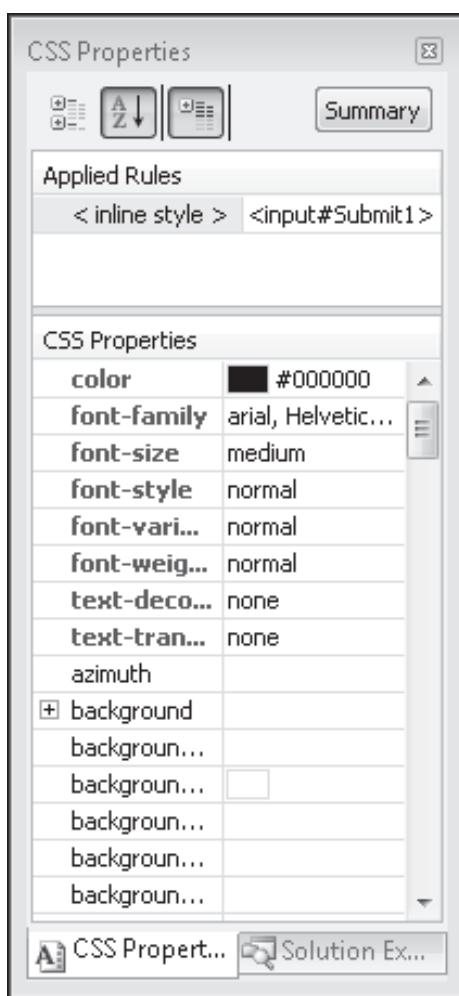


Figure 13.13: CSS Properties Window

Manage Styles window is a new added feature to Visual Studio 2008. It allows you to visualize and manage the CSS styles that you are presently working in both the design and the source view. If you see a circled item, it means that the rule is applied somewhere in the existing Web page. If you hover the mouse on a specific style, you can see the rule values applied to the control.

Figure 13.14 shows the Manage Styles window.



Figure 13.14: Manage Styles Window

As shown in figure 13.14, you can select a rule and see how it will be rendered in the bottom pane in the Selected Style Preview section of the Manage Styles window.

You can double-click a rule to open the style sheet and make changes in it. Another new feature, full IntelliSense support, is now available in your style sheets.

Figure 13.15 shows the IntelliSense support for style sheets.

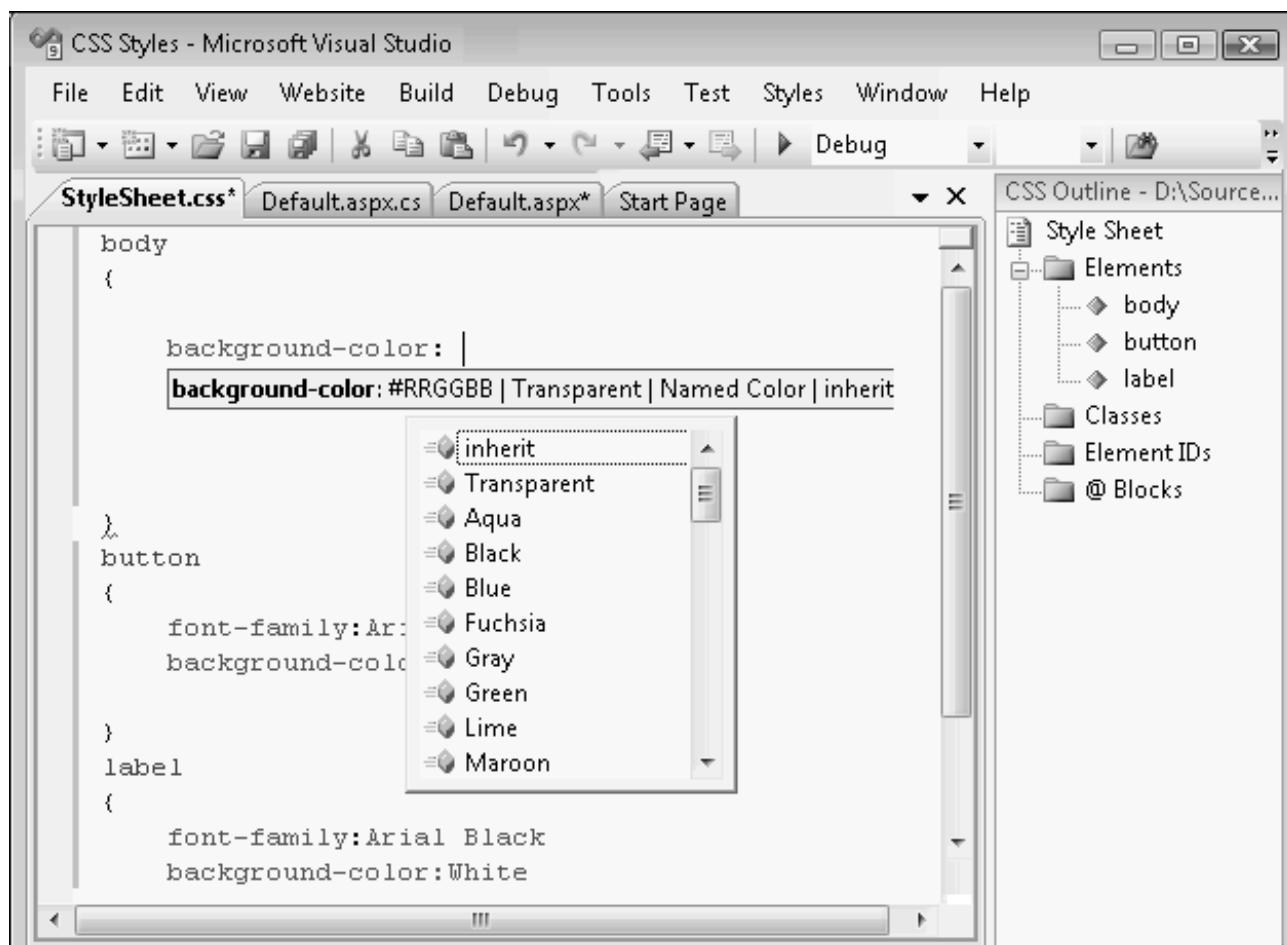


Figure 13.15: IntelliSense for Style Sheets

13.10 Check Your Progress

1. Which of the following statements about HyperLink control are true?

(A)	HyperLink control enables you to link a number of Web pages to one another in a Web site.
(B)	HyperLink control is used to create a sequence of forms to gather a wide range of data from users.
(C)	HyperLink control allows you to create a new step, add a new step, or reorder the steps in a Web site.
(D)	HyperLink control allows you to create a step by step procedure for collecting data from users.

(A)	a, b	(C)	a
(B)	c, d	(D)	a, d

2. Which property gets or sets the value of the current WizardStepBase object in the Wizard control?

(A)	ActiveStep	(C)	ActiveStepIndex
(B)	StepStyle	(D)	Index

3. Which of the following are the event procedures types?

(A)	Client-side event procedures
(B)	Page events
(C)	Control events
(D)	Server-side event procedures

(A)	b, c	(C)	a, b
(B)	c, d	(D)	a, d

4. Which of the following are the page events that are fired for a Web page?

(A)	InitComplete
(B)	LoadComplete
(C)	RenderComplete
(D)	SaveStateComplete

(A)	a, b, c	(C)	a, c, d
(B)	a, b, d	(D)	a, b, d

5. Which of the following data controls are introduced in ASP.NET 3.5?

(A)	DataList
(B)	DataPager
(C)	ListView
(D)	LinqDataSource

(A)	b, c, d	(C)	b, d
(B)	a, b	(D)	a, c

6. Identify the new assemblies are introduced by ASP.NET 3.5.

(A)	System.Xml
(B)	System.Data.Linq
(C)	System.Xml.Linq
(D)	System.Core

(A)	a, c, d	(C)	b, c, d
(B)	a, b, d	(D)	a, b, c

7. Which of these features allows you to visualize and manage the CSS styles in Visual Studio 2008 IDE?

(A)	JavaScript Debugging Window	(C)	Manage Styles Window
(B)	Style Sheet Window	(D)	CSS Properties Window

8. Which of the following features helps developers to write client side validation in an easy and efficient manner?

(A)	JavaScript IntelliSense Support	(C)	Nested master page support
(B)	Enhanced Web Design Experience	(D)	Multi-Framework Targeting

Module Summary

In this module, **Introduction to ASP.NET 3.5**, you learnt about:

→ **ASP.NET 3.5**

Microsoft ASP.NET 3.5 has introduced several new enhancements. New controls like ListView, Data-Pager, and LinqDataSource are introduced in ASP.NET 3.5. The Wizard control is used to create a step by step procedure for collecting user input in a Web application. The HyperLink control is used to navigate from one Web form to another Web form in a Web application. The two types of event procedures are client-side and server-side event procedures. Several new assemblies are introduced in ASP.NET 3.5 that support LINQ feature. New features in Visual Studio 2008 IDE for ASP.NET 3.5 include ASP.NET AJAX Extensions, JavaScript, and CSS IntelliSense support, Manage Styles window, and so forth. Nested Master pages support is a feature available now in ASP.NET 3.5.

Module - 14

Introduction to ASP.NET 3.5 (Lab)

Welcome to the module, **Introduction to ASP.NET 3.5 (Lab)**.

In this module, you will learn to:

- Create an ASP.NET 3.5 Web application
- Use HyperLink and Wizard controls
- Work with Wizard templates
- Use Page events in ASP.NET

Web Development

http://www



14.1 Part I – 90 Minutes

Exercise 1

Star Zone Private Limited is a company located in Los Angeles. The company deals with the development of online Web sites for businesses. They want to recruit new candidates as there are many upcoming projects to be completed. Hence, the company requires information about new candidates so that they can call them for interviews and select the candidates based on their skills. The company needs the details of the candidates such as name, contact details, and skills.

Assume that you are a Web developer for Star Zone and need to create a step by step procedure for collecting the details of the candidates. You need to perform the following tasks:

- Create a Welcome form that will navigate to the data entry page based on the click of a HyperLink control.
- Add a new form for data entry that will use a Wizard to accept user input.
- Add buttons for navigations like Previous and Next.
- Add validations for checking the data.
- Add a final completion step.
- Ensure that the details should be displayed at the last stage.
- You must use Visual Studio 2008 and ASP.NET 3.5 to create the Web-based application.

Solution:

1. Create a Web site named **CandidateSystem** as shown in figure 14.1.

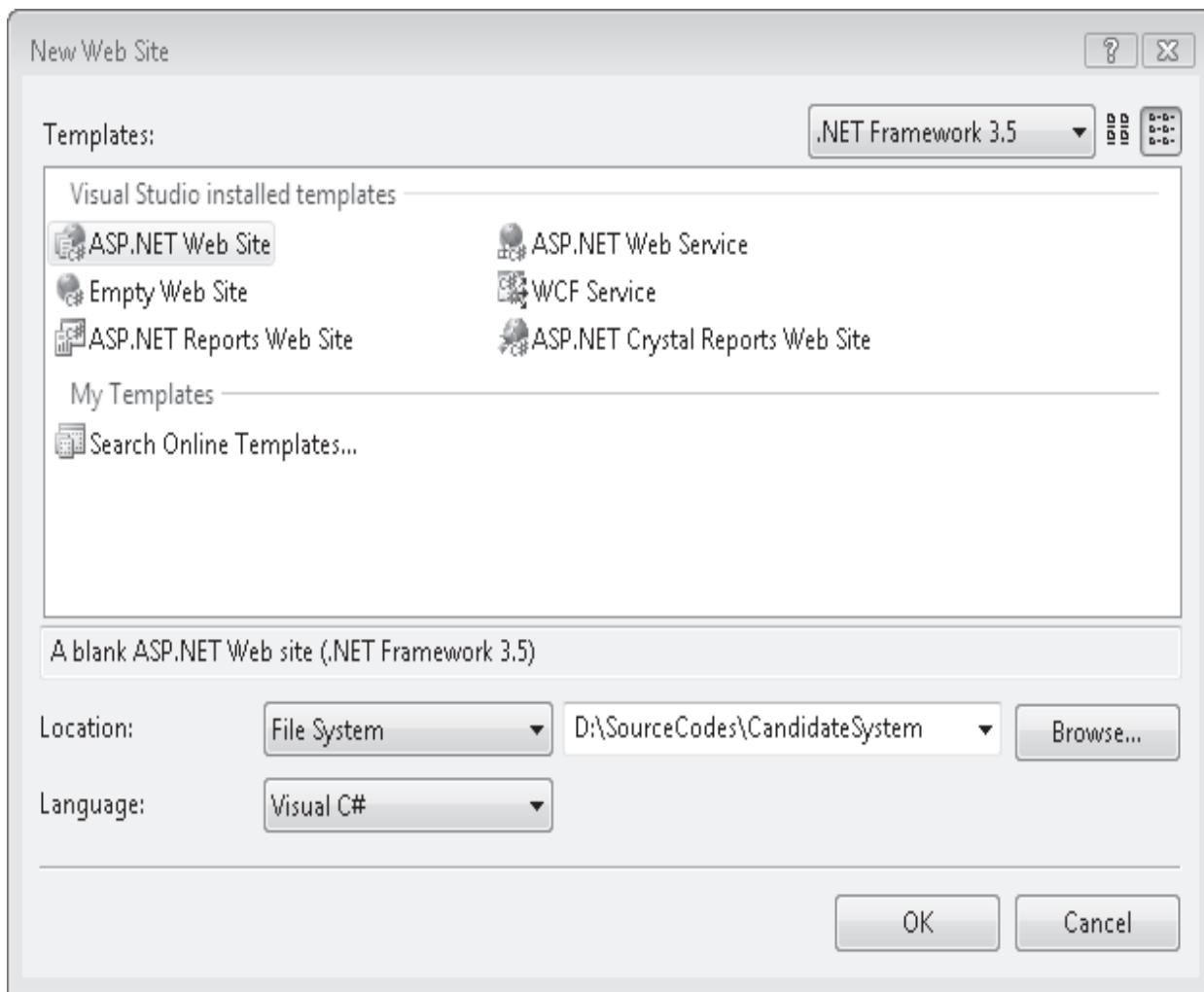


Figure 14.1: Creating a CandidateSystem Web Site

2. Rename the default Web Form, **Default.aspx**, as **Welcome.aspx**.
3. Add a new Web form to the Web site by either using **File** menu and selecting **New → File** sub-menu command or using **Web Site → New** command.
4. Rename the new Web Form as **CandidateDetails.aspx**.
5. Double-click the **Welcome.aspx** page in the **Solution Explorer**.
6. Add a **Heading 2** tag in the **Source View** under the **<form>** tag and type the text as “**Welcome to Candidate Entry System**”.
7. Change the **Body Bgcolor** to **#99ccff**.

8. Add a **HyperLink** control to the **Design View** from the **Standard** group of the **Toolbox** in the **Welcome.aspx** page. This **HyperLink** control will help for the navigation from **Welcome** page to the data entry page.
9. Select the **HyperLink** control and configure the properties as shown in table 14.1.

Control	Property	Value
HyperLink	Text	Candidate Details Entry Form
	ID	InkEnter
	NavigateUrl	~/CandidateDetails.aspx

Table 14.1: Properties of the HyperLink Control

The resulting interface will appear as shown in figure 14.2.

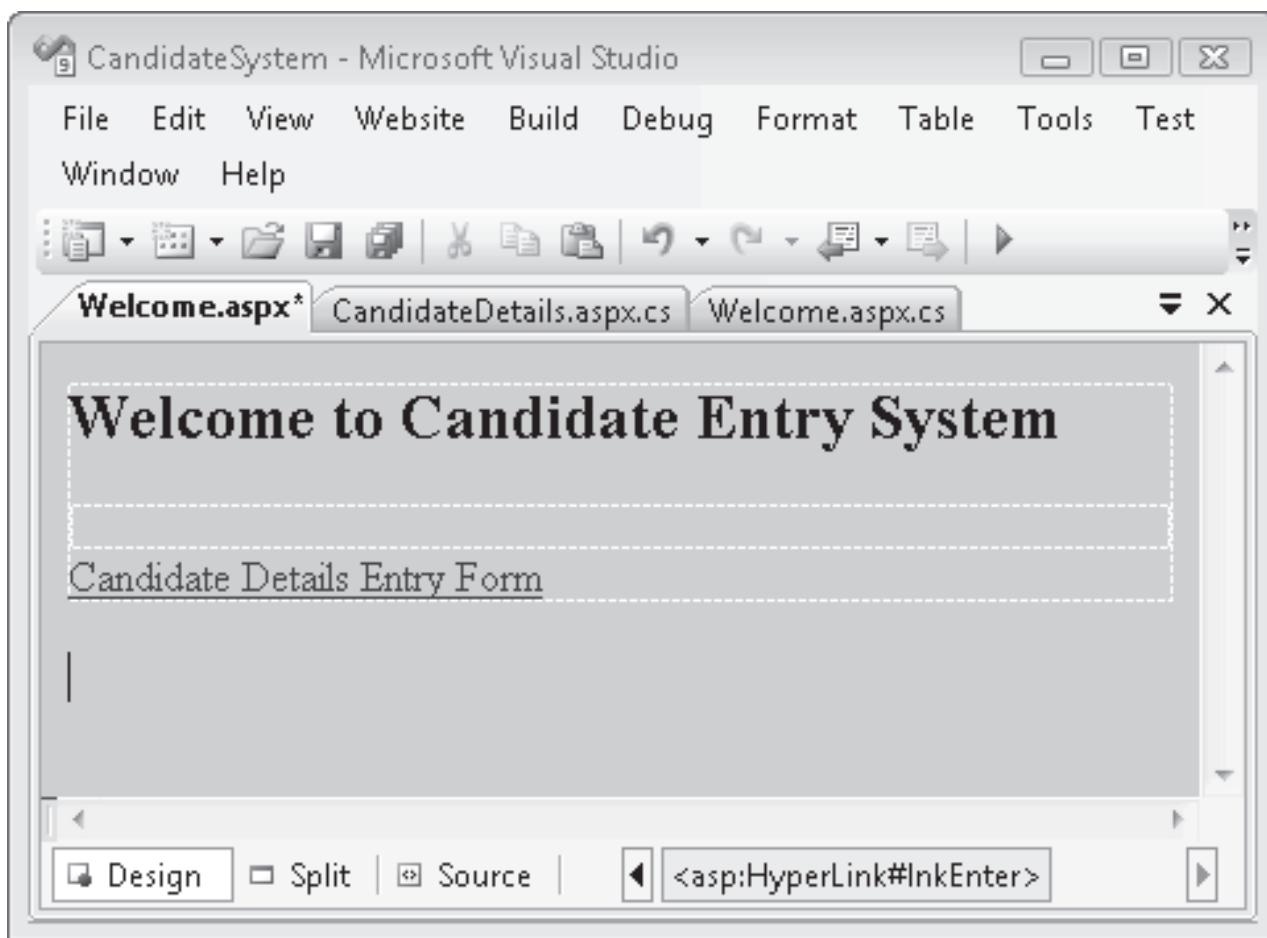


Figure 14.2: Welcome.aspx in Design Mode

10. Double-click the **CandidateDetails.aspx** page in the **Solution Explorer**.
11. Switch to the **Design view** of the **CandidateDetails.aspx** page.

12. Add a **Wizard** control from the **Standard** group of the **Toolbox** to the **CandidateDetails.aspx** page. This **Wizard** will consist of a step by step layout to accept user input. The number of steps, the arrangement of steps, and the completion can be customized as per requirements. Figure 14.3 shows the **Wizard** control added.

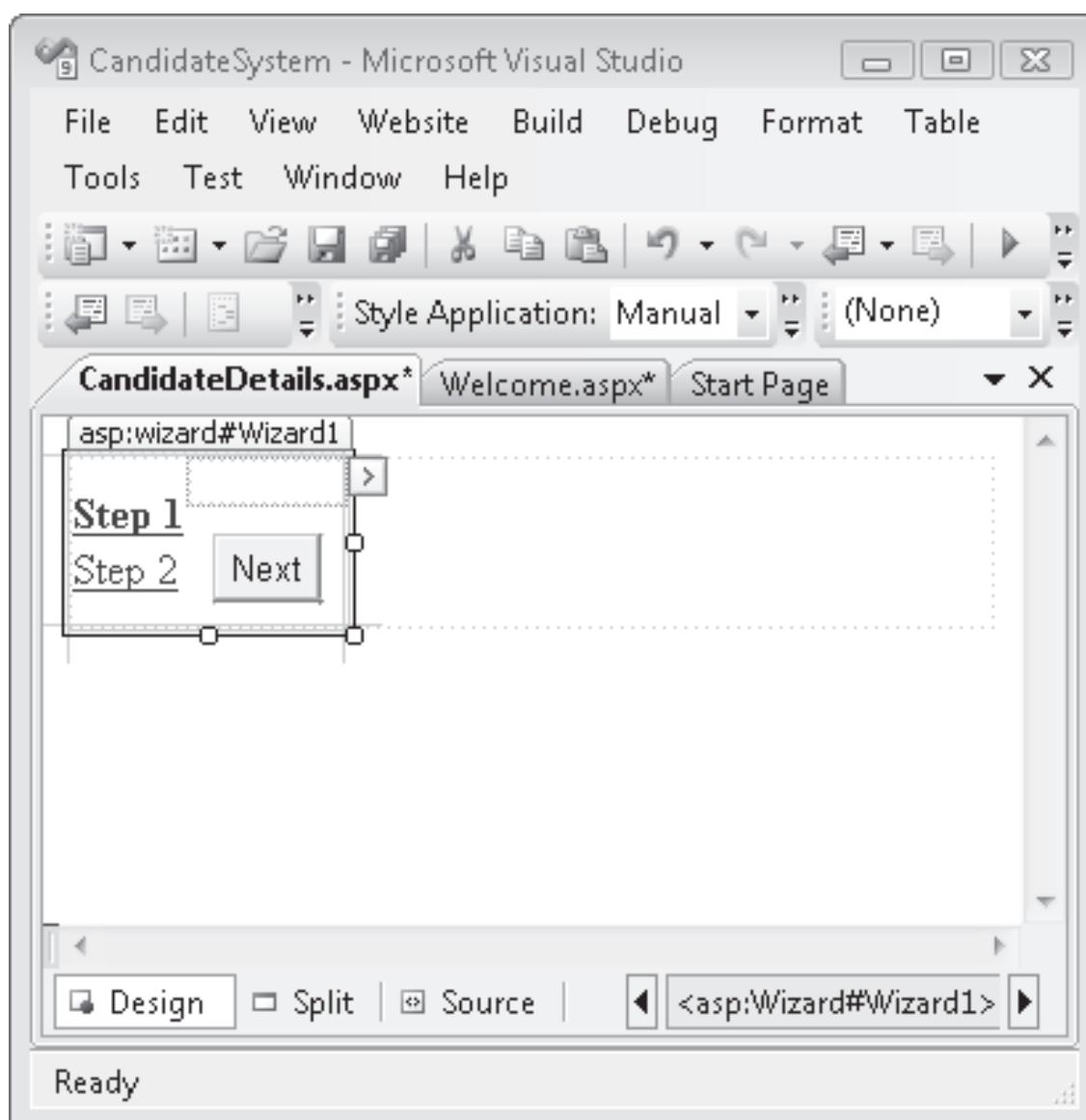


Figure 14.3: Adding a Wizard Control

13. Select the Wizard control and configure the properties as shown in table 14.2.

Control	Property	Value
Wizard	ID	wizControl
	Font-names	Verdana
	BackColor	Silver
	ForeColor	Navy
	ActiveStepIndex	0
	Width	494px
	Height	164

Table 14.2: Properties of the Wizard Control

14. Select the **StepStyle** property, expand it, and change the **BackColor** to **Gainsboro**. Then, set the **BorderWidth** to **1px** and **BorderStyle** as **Outset**.
15. Select the **WizardSteps** property of the **Wizard control** and click the collection ellipsis. The **WizardStep Collection Editor** will be displayed as shown in figure 14.4.

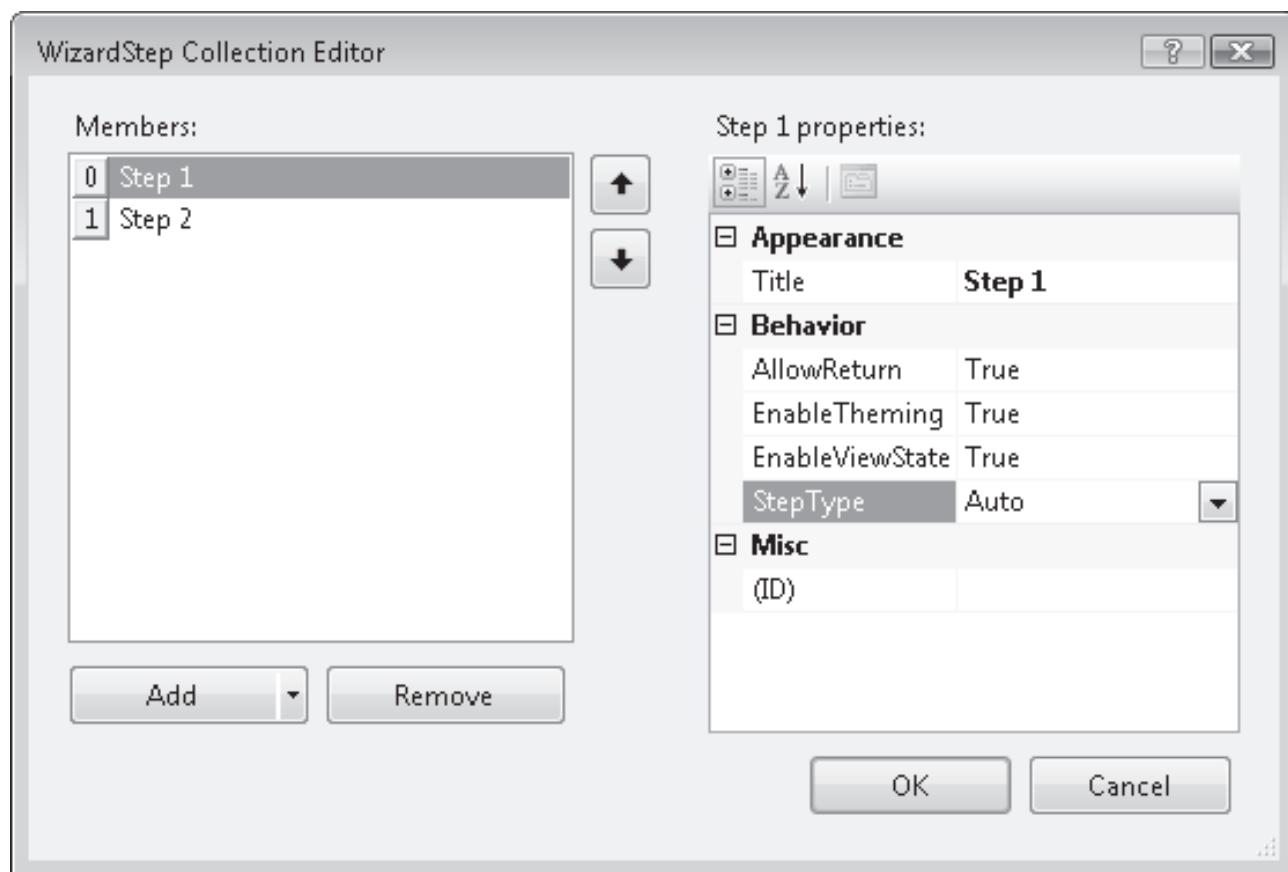


Figure 14.4: WizardStep Collection Editor

16. In the **WizardStep Collection Editor**, add four steps, and configure the properties as shown in table 14.3.

Step	Property	Value
Step 1	Title	Name
	ID	wizStepone
	StepType	Auto
Step 2	Title	Contact Information
	ID	wizSteptwo
	StepType	Auto
Step 3	Title	Skills
	ID	wizStepthree
	StepType	Auto
Step 4	ID	wizStepfour
	StepType	Complete

Table 14.3: Properties of the WizardStep Collection Editor

The results will be displayed as shown in figure 14.5.

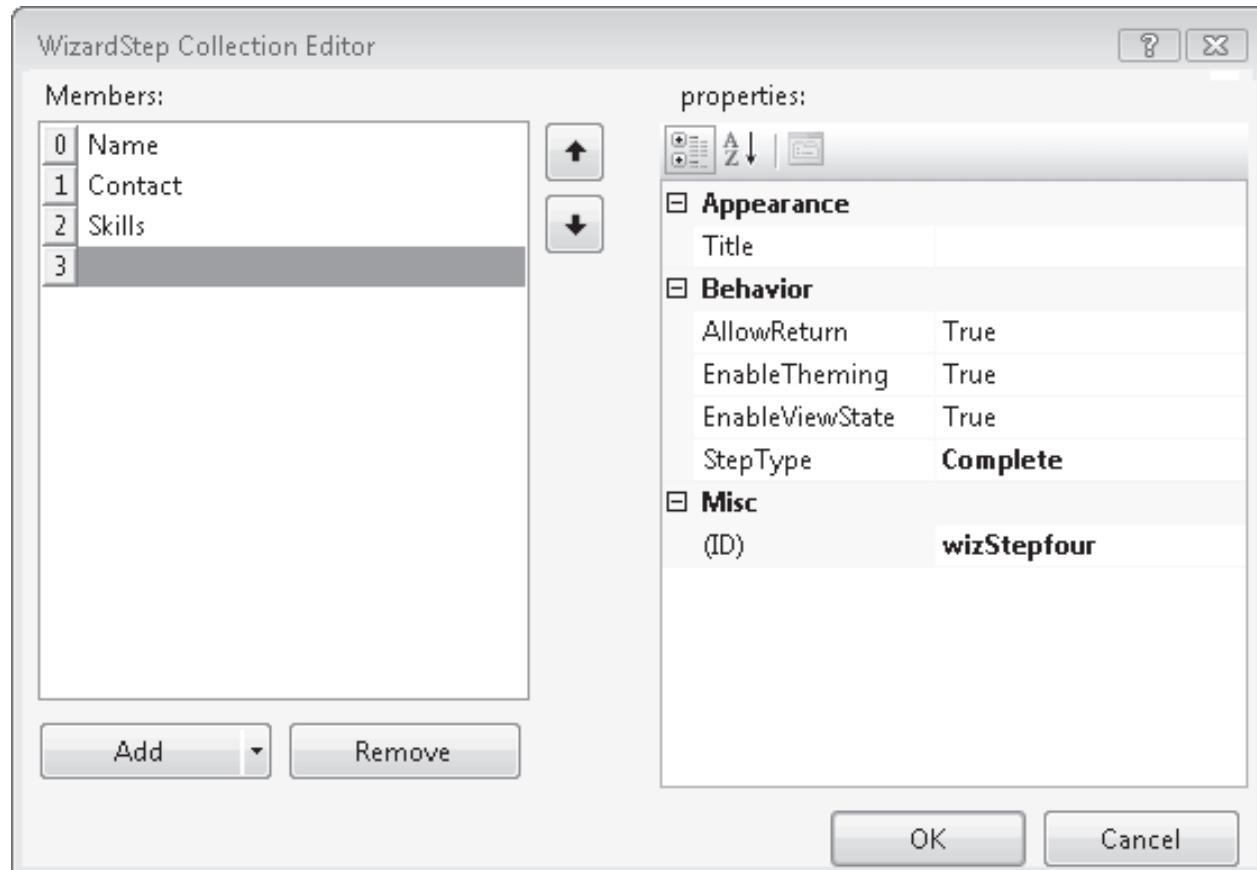


Figure 14.5: Steps in WizardStep Collection Editor

17. Select the **Wizard** Control, click the underlined text **Name Step**, and drag the **Table** control from the **HTML** group in the **Toolbox** into the step area. Type the text as “**Name Information**” in the first row of the table.
18. Select the second row of the table, add the text “**First Name:**”, drag a **TextBox** control, and a **RequiredFieldValidator** control beside the text.
19. Select the third row of the table, add the text “**Last Name:**”, drag a **TextBox** control, and a **RequiredFieldValidator** control beside the text.
20. Below the table, add a **ValidationSummary** control.
21. In the **Name Step** link, select the table and configure the properties of the text box controls, and **RequiredFieldValidator** controls as shown in table 14.4.

Control	Property	Value
Text box	ID	txtFname
RequiredFieldValidator	ID	reqvalFname
	Text	*
	ErrorMessage	Enter first name
	SetFocusOnError	True
	ControlToValidate	txtFnameID
Text box	ID	txtLname
RequiredFieldValidator	ID	reqvalLname
	ControlToValidate	txtLname
	ErrorMessage	Enter last name
	SetFocusOnError	True
	Text	*
Validation Summary	ID	valSummary
	DisplayMode	BulletList
	Height	132px
	Width	240px

Table 14.4: Properties of the Name Step in the Wizard Control

22. Click the underlined text “**Contact Information**” and drag an **HTML Table** in the step named as **Contact Information**.
23. Select the second row of the table, add the text “**Address:**”, drag a **TextBox** control, and a **RequiredFieldValidator** control beside the text.
24. Select the third row of the table, add the text as “**Phone Number:**”, drag a **TextBox** control, and a **RegularExpressionValidator** control beside the text.

25. In the **Contact Information Step** link, select the table, configure the properties of the text box controls, **RequiredFieldValidator**, and **RegularExpressionValidator** as shown in table 14.5.

Control	Property	Value
Text box	ID	txtAddress
	TextMode	MultiLine
RequiredFieldValidator	ID	valAddress
	ControlToValidate	txtAddress
	ErrorMessage	Enter address
	SetFocusOnError	True
	Text	*
Text box	ID	txtPhone
	MaxLength	10
Validation Summary	ID	valSummaryContact
	DisplayMode	BulletList
	Height	132px
	Width	240px
RegularExpressionValidator	ID	valPhone
	ControlToValidate	txtPhone
	ErrorMessage	Enter phone number
	SetFocusOnError	True
	Text	*
	ValidationExpression	\d{10}

Table 14.5: Properties of the Contact Step

26. Click the underlined text **Skills** and drag an **HTML Table** control in the step named as **Skill Information**.
27. Select the second row of the table, add the text as “**Qualification:**”, drag a **TextBox** control, and a **RequiredFieldValidator** control beside the text.
28. Select the third row of the table, add the data as “**Skills:**”, drag a **TextBox** control, and a **RegularExpressionValidator** control beside the text.
29. In the **Skills Step** link, select the table, configure the properties of the text box controls, and **RequiredFieldValidator** as shown in table 14.6.

Control	Property	Value
Text box	ID	txtQualification
RequiredFieldValidator	ID	valQualification
	Text	*
	ErrorMessage	Enter qualification

Control	Property	Value
	SetFocusOnError	True
	ControlToValidate	txtQualification
Text box	ID	txtSkills
RequiredFieldValidator	ID	reqvalSkills
	ControlToValidate	txtSkills
	ErrorMessage	Enter skills
	SetFocusOnError	True
	Text	*
Validation Summary	ID	valSummarySkills
	DisplayMode	BulletList
	Height	132px
	Width	240px

Table 14.6: Properties of the Skills Step

30. On the last step, type the text as “**Thank you for completing this survey**”. The resulting Wizard will be displayed in the Design view as shown in figure 14.6.

The screenshot shows the 'Wizard' control in design mode. On the left, there's a sidebar with three steps: 'Name', 'Contact', and 'Skills'. The 'Skills' step is currently selected. The main area has a title 'Name Information' and two text input fields: 'First Name:' and 'Last Name:', each with a corresponding text box. At the bottom right of the main area is a 'Next' button.

Figure 14.6: Wizard Control

31. Click the step **Skills**.
32. Double-click the **Finish** button on the **Wizard** control. The **CandidateDetails.aspx.cs** page will be opened.
33. Generate **Click** event for the form and add code as follows to the event handler.

```
/// <summary>
/// This code will display the Candidate details
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void wizControl_FinishButtonClick(object sender,
    WizardNavigationEventArgs e)
{
    Response.Write("<b>First Name:</b>" + txtFname.Text + "<br>") ;
    Response.Write("<b>Last Name:</b>" + txtLname.Text + "<br>") ;
    Response.Write("<b>Address:</b>" + txtAddress.Text + "<br>") ;
    Response.Write("<b>Phone Number:</b>" + txtPhone.Text + "</
        br>") ;
    Response.Write("<b>Qualification:</b>" + txtQualification.Text +
        "<br>") ;
    Response.Write("<b>Skills:</b>" + txtSkills.Text + "<br>") ;
}
```

The code will display the candidate details such as candidate name, contact information, and skills.

34. In the **Solution Explorer**, right-click the **Welcome.aspx** page, select **Set as Start Page**, and execute the application. Figure 14.7 shows the **Welcome.aspx** page.

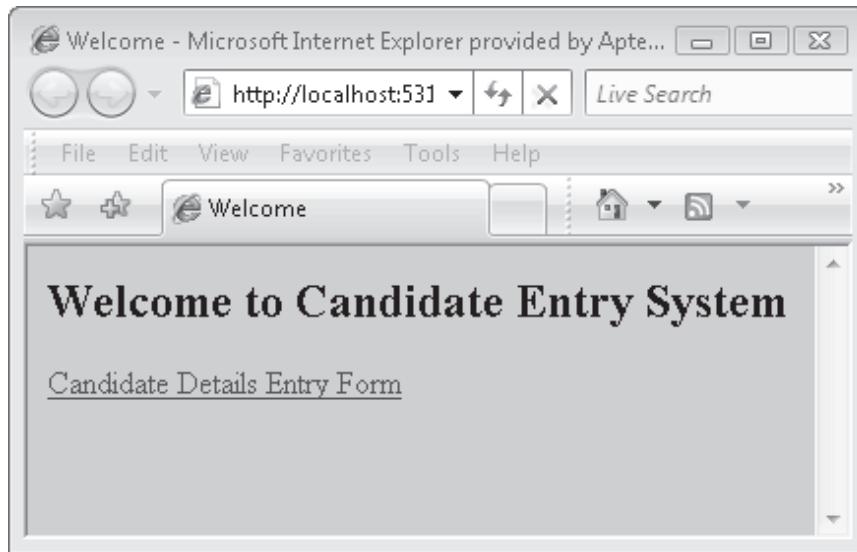


Figure 14.7: Welcome.aspx Page

35. Click the link **Candidate Details Entry Form**. When you click the link, **CandidateDetails.aspx** page will be displayed as shown in figure 14.8.

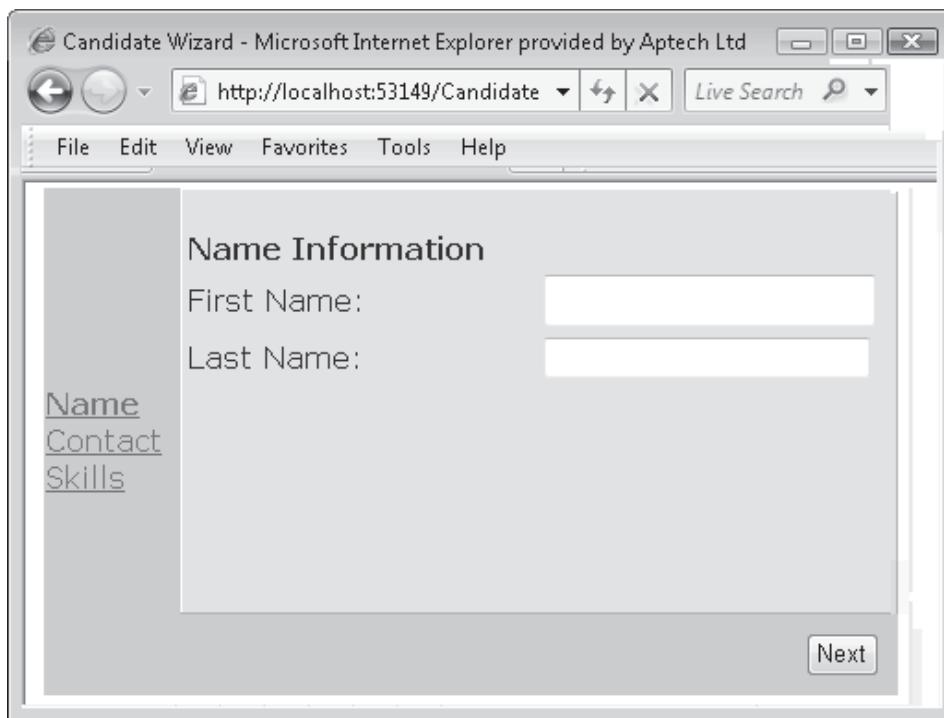


Figure 14.8: CandidateDetails.aspx Page

36. Click the **Next** button. The Web page will display error messages as shown in figure 14.9.

The screenshot shows a Microsoft Internet Explorer window with the title "Candidate Wizard - Microsoft Internet Explorer provided by Aptech Ltd". The address bar shows "http://localhost:53149/Candidate". The page content is titled "Name Information". It contains two text input fields: "First Name" and "Last Name", both of which have an asterisk (*) next to them, indicating they are required fields. To the right of these fields, there is a validation message: "• Enter first name" and "• Enter last name". On the left side of the form, there is a vertical navigation menu with three items: "Name", "Contact", and "Skills". At the bottom right of the form is a "Next" button.

Figure 14.9: Invalid Data and Error Message

37. Now, add the details with valid data in the **Wizard** control as shown in figure 14.10.

The screenshot shows a Microsoft Internet Explorer window with the title "Candidate Wizard - Microsoft Internet Explorer provided by Aptech Ltd". The address bar shows "http://localhost:53149/Candidate". The page content is titled "Name Information". It contains two text input fields: "First Name" with the value "Andy" and "Last Name" with the value "Williams". Both fields are no longer marked with an asterisk (*). On the left side of the form, there is a vertical navigation menu with three items: "Name", "Contact", and "Skills". At the bottom right of the form is a "Next" button.

Figure 14.10: Example of Valid Data

38. Click the **Contact** link or click the **Next** button. This will display the **Contact Step** section of the **Wizard**.
39. Add the Contact details as shown in figure 14.11.

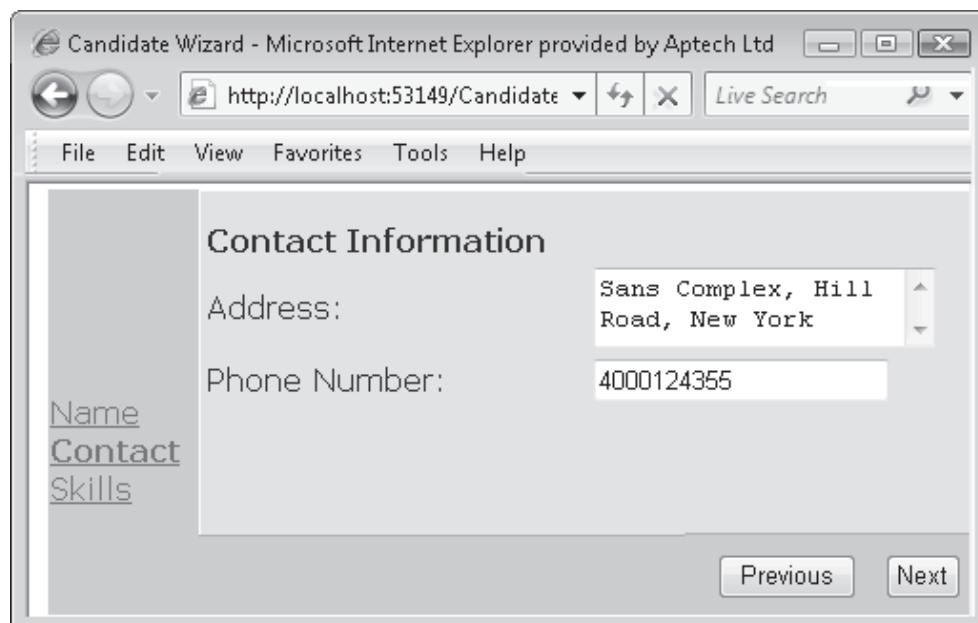


Figure 14.11: Adding Contact Details in the Wizard Control

40. Add the Skills details as shown in figure 14.12.

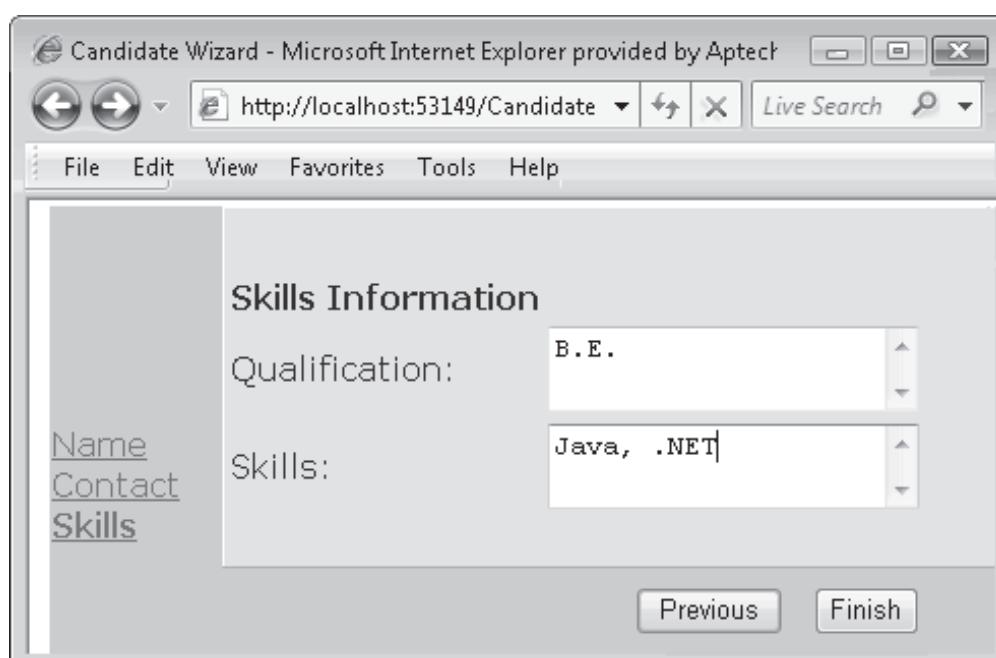


Figure 14.12: Adding Skills Details

41. Click the **Finish** button. The application will display the candidate details. The output will be as shown in figure 14.13.

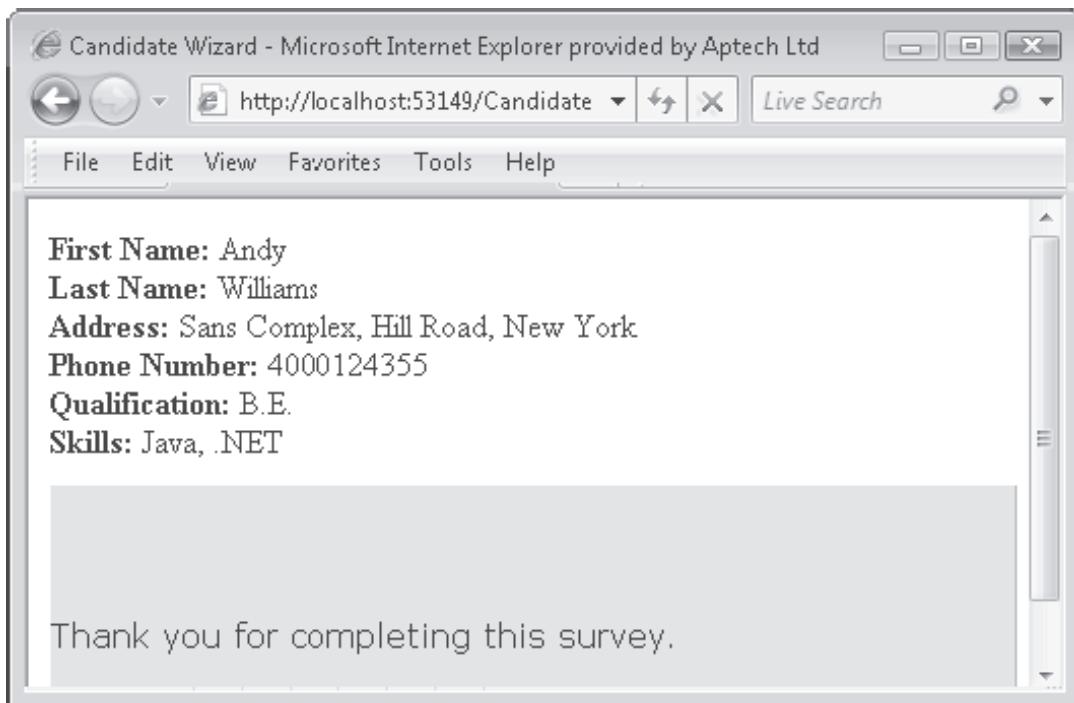


Figure 14.13: Displaying the Candidate Details

Exercise 2

Amazon Technologies is a leading company for developing products. The company has now decided to expand their branches all over the world. They have come to a conclusion to conduct a survey through which they can collect information about their current and prospective customers. Assume that you are a developer at Amazon Technologies and perform the following:

- Create a Customer form that will accept the details of the customers.
- Add a new form for data entry that will use a Wizard to accept user input.
- Add HeaderTemplate, StartNavigationTemplate, and FinishNavigationTemplate to the Wizard.

Solution:

1. Create a Web site named **Customer Management**.
2. Rename the **Default.aspx** page as **Customer.aspx**.
3. Drag a **Wizard** control from the **Toolbox** to the **Customer** page.
4. Click the smart tag next to the **Wizard** control and select the **AutoFormat** option.

5. Select the **Professional** option from the **AutoFormat** choices.
6. Add three steps as **Personal**, **Residential**, and **Official** information. In order to add steps in the Wizard, follow the procedure similar to Exercise 1.
7. Add validations as described in Exercise 1.
8. Click the smart tag of the **Wizard** control. It will display **Wizard tasks** as shown in figure 14.14.

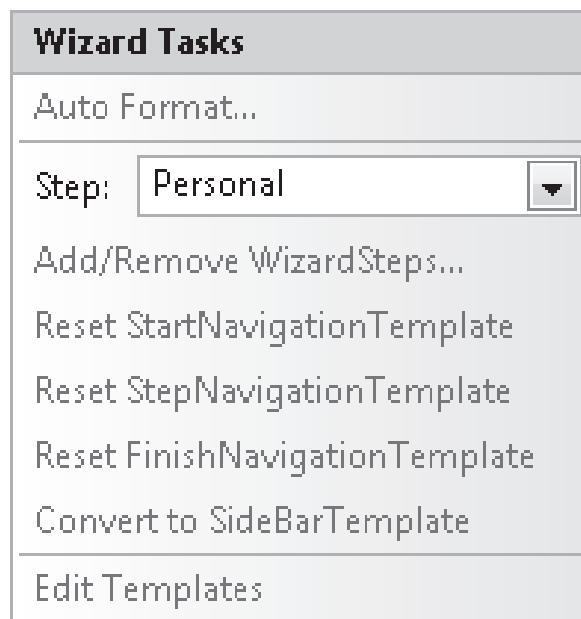


Figure 14.14: Wizard Tasks

9. Click **Edit Templates**. It will display the different types of templates.
10. Select the **Header Template** from the list of templates as shown in figure 14.15.

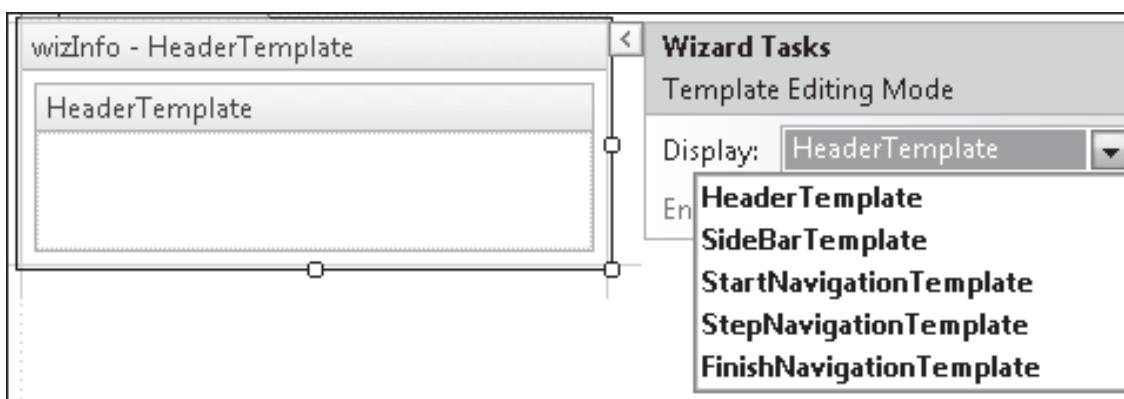


Figure 14.15: Wizard Tasks with Templates

11. Drag a Label control on the Header template and change the text as “Customer Information” as shown in figure 14.16.

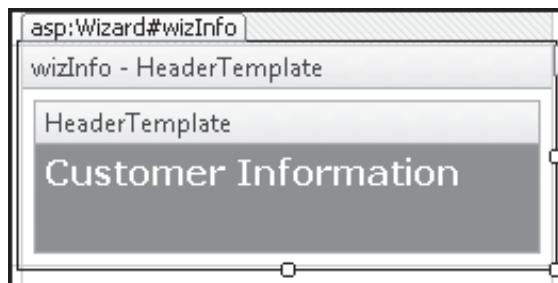


Figure 14.16: Header Template with Customer Information

12. Design the **Wizard** control and add three steps as **Personal**, **Residential**, and **Official**.
 13. In the **Personal Step**, add two labels, change the text as “**First Name:**” and “**Last Name:**” respectively, and add two text boxes as shown in figure 14.17.



Figure 14.17: Wizard Control showing Personal Step

14. In the **Personal Step**, select the **Wizard** tasks.
 15. Click the **Edit Templates** link and select the **StartNavigation** template.
 16. Add the Text as “**Customer Survey has started.**” in the **StartNavigation** template.
 17. Add a **Button** control and change the **Text** property as “**Continue...**” as shown in figure 14.18.

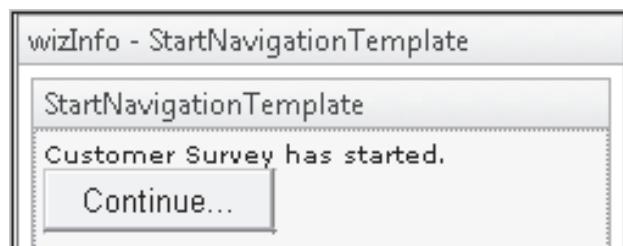


Figure 14.18: Adding the StartNavigation Template

18. Click the **End Template Editing** on the control. The **Wizard** control will appear as shown in figure 14.19 with **StartNavigation** template.

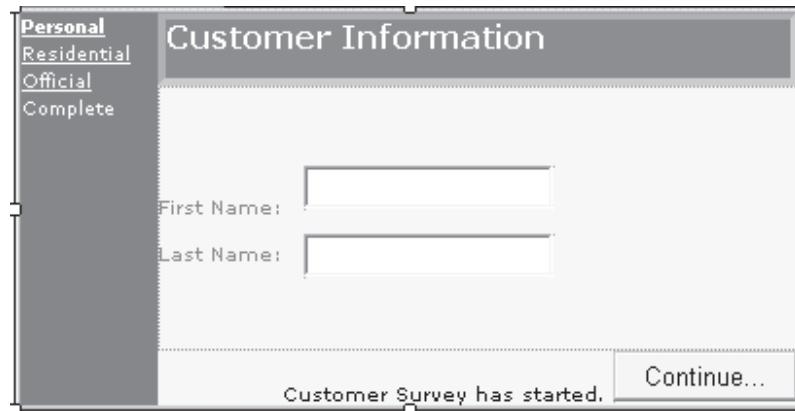


Figure 14.19: StartNavigation Template

19. Click the **End Template Editing** on the control.
 20. In the **Residential Step**, add two labels, change the text as “**Address:**” and “**Phone:**” respectively, and add two text boxes as shown in figure 14.20.

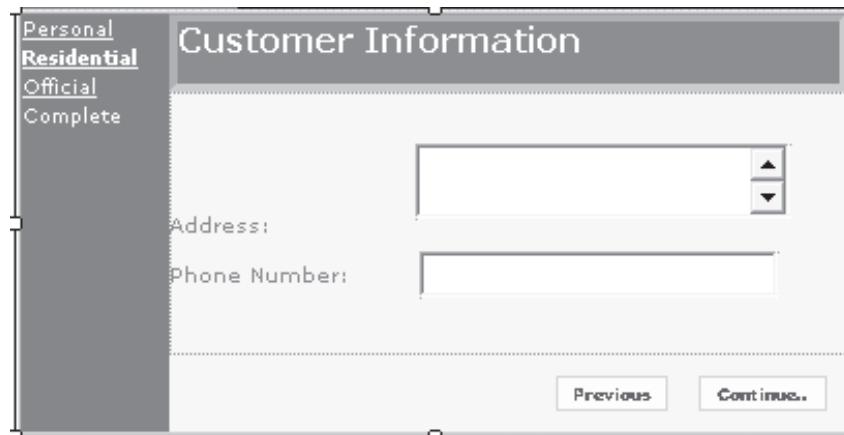


Figure 14.20: StartNavigation Template

21. Click the **Edit Templates** link and select the **FinishNavigation** template. The **FinishNavigation** template will be displayed as shown in figure 14.21.

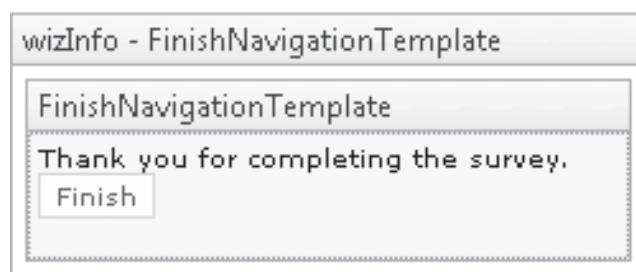


Figure 14.21: Adding FinishNavigation Template

22. In the **Official Step**, add two labels, change the text as “**Address:**” and “**Phone:**” respectively, and

add two text boxes.

23. Add the text as “**Thank you for completing the survey.**” in the **FinishNavigationTemplate** as shown in figure 14.22.

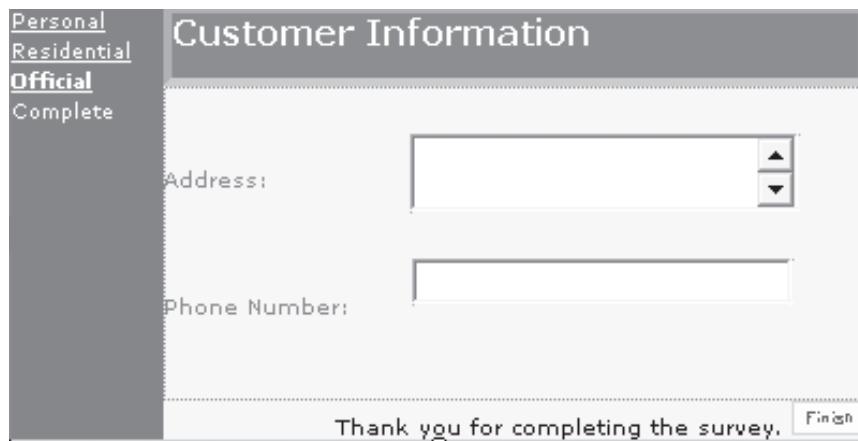


Figure 14.22: Wizard Control showing Official Step

24. Press **F5** and execute the application. Figure 14.23 shows the **StartNavigationTemplate** added.

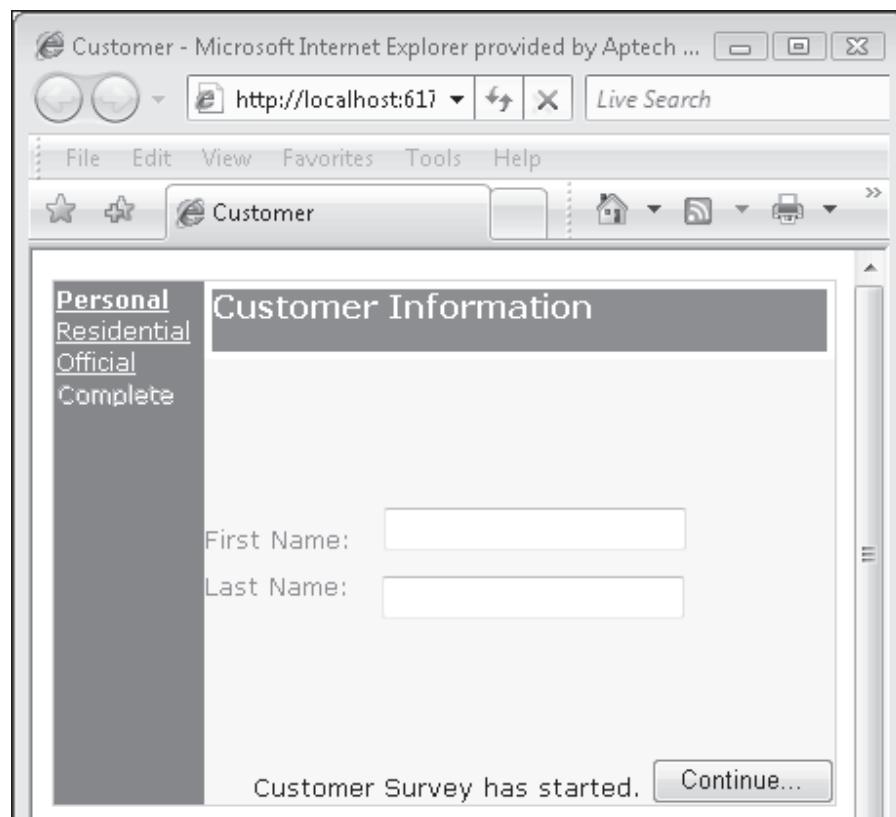


Figure 14.23: Customer.aspx Page with StartNavigationTemplate Added

25. Enter the details of the Customer such as first name and last name as shown in figure 14.24.

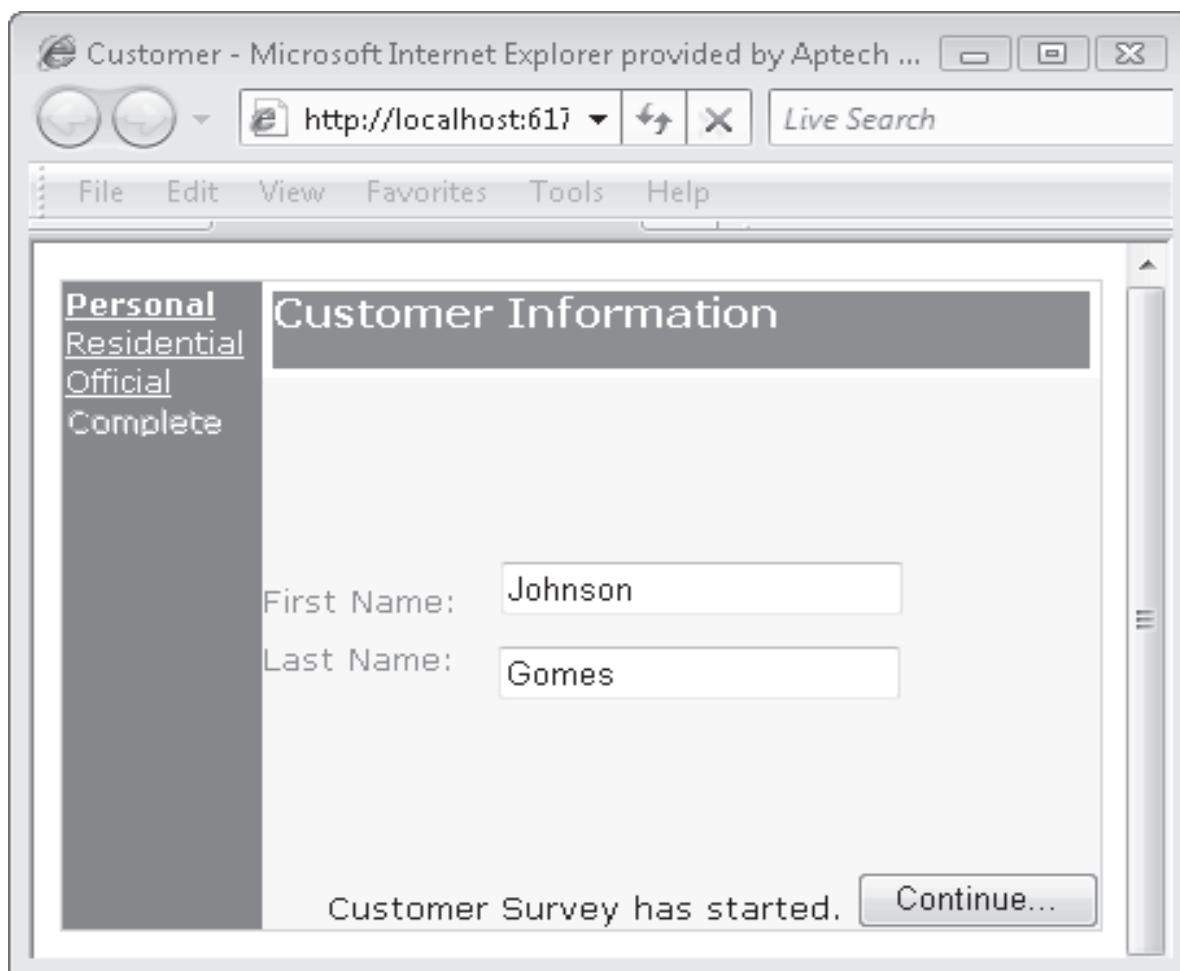


Figure 14.24: Personal Step in Customer.aspx Page

26. Enter the details for customer residential address and phone number as shown in figure 14.25.

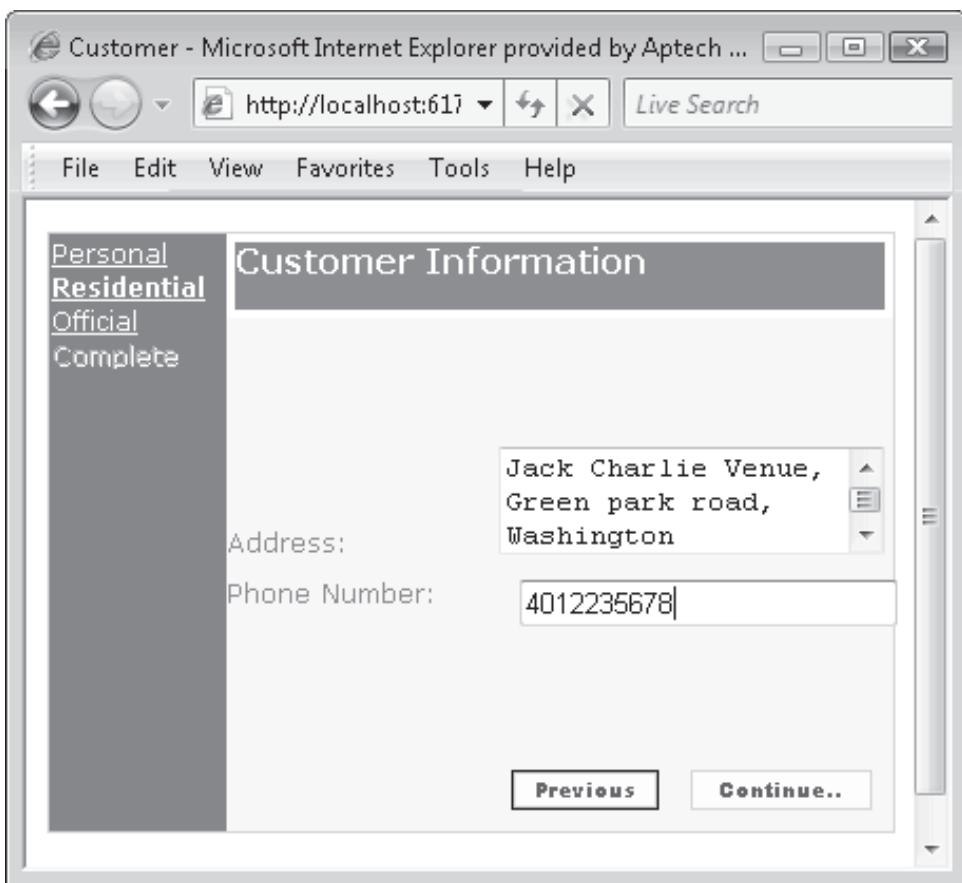


Figure 14.25: Residential Step in Customer.aspx Page

27. Similarly, add the details for customer official address and phone number as shown in figure 14.26. Figure 14.26 displays the FinishNavigation template.

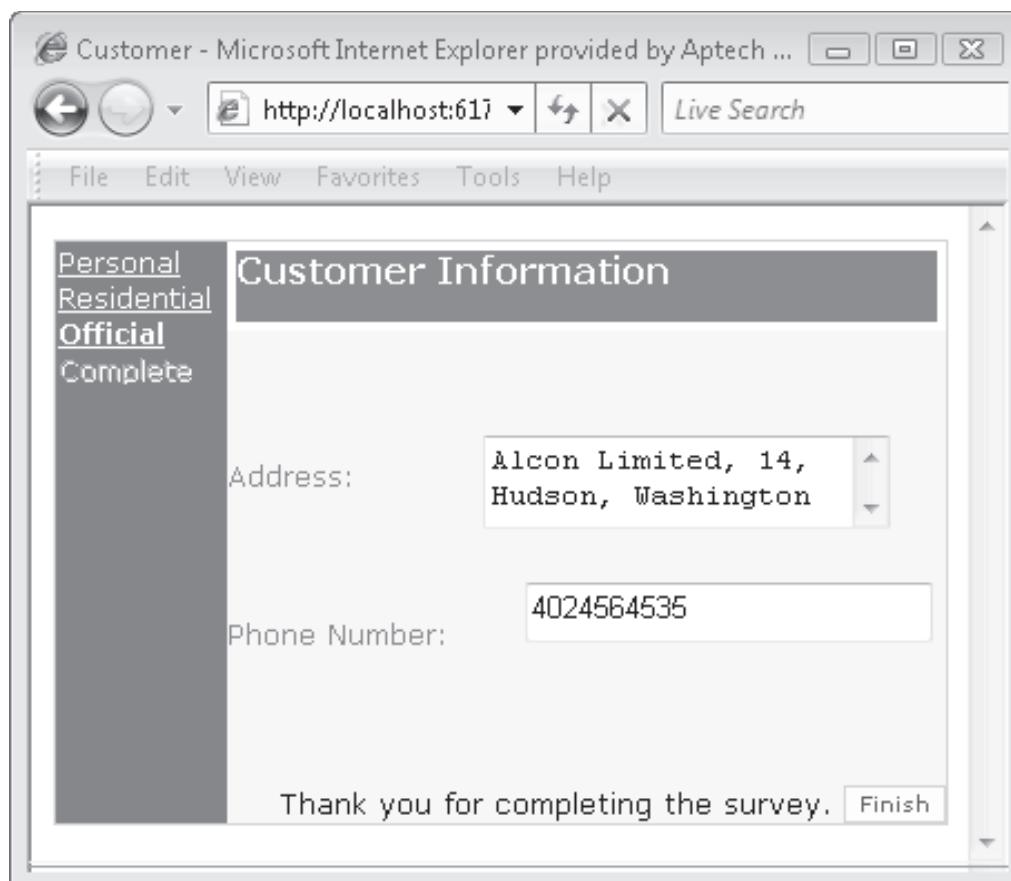


Figure 14.26: Official Step in Customer.aspx Page

14.2 Part II - 30 Minutes

Jovial Associates is a leading company in Web development. Assume that you are a developer working for them. They want you to check the different types of events generated in the life cycle of a Web page. You need to examine the sequence of events that are getting fired.

Hints:

Handle the following events in the code-behind file and display appropriate messages for each event:

- ➔ PreInit
- ➔ InitComplete
- ➔ PreLoad, Load
- ➔ LoadComplete
- ➔ SaveStateComplete

14.3 Do It Yourself

Opex Private Limited is a leading company for developing automobiles. The company has created several Web sites for their products using the earlier versions of ASP.NET 2.0. Now, they want to build their Web sites using ASP.NET 3.5 to implement new functionalities of Visual Studio 2008 and ASP.NET 3.5 in their projects. You need to create a Web site that will allow customers to register online. Use the new features of Visual Studio 2008 and ASP.NET 3.5.

Module - 15

Working with ASP.NET 3.5

Welcome to the module, **Working with ASP.NET 3.5**. This module covers cross-page posting, Server.Transfer method, and session management in ASP.NET. It also describes other essential features for working with ASP.NET.

In this module, you will learn to:

- Describe cross-page posting
- Explain the Server.Transfer() method
- Explain session state management in ASP.NET
- Explain the types of session state modes
- Describe the steps to set session timeout
- Explain the process of retrieving values from cookies
- Describe cookieless sessions

Web Development

http://www



15.1 Cross-Page Posting

On ASP.NET pages, when controls such as buttons that are capable of postback are clicked, they post the content back to the same page. Cross-page posting is the process of submitting a Web Form or posting Web Form content to a different page. Consider that you are collecting the flight information on one page and the output is shown on another page. Here, the first page data will be cross posted to the second page. Cross-page posting is generally needed when you are developing a multipage form to gather the information from the user on each page. When you are navigating from the source page to the destination page, the control values of the source page are accessible to the destination page.

Cross-page posting can be used with the `PostBackUrl` attribute to identify the page that you want to post. The following steps are used to perform cross-page posting:

1. Create an ASP.NET Web site called **PostingExample** with two pages, **First.aspx** and **Second.aspx** respectively.
2. Drag a **Button** control on the **First.aspx** page and change the **Text** property to **Click**. Name the **Button** as **btnClick**.
3. Set the property `PostBackUrl` of the **Button** to the target page, **Second.aspx**.

The resultant code for the Button will be as shown in the following Code Snippet:

Code Snippet:

```
<asp:Button ID="btnClick" runat="server" Text="Click"  
    PostBackUrl="~/Second.aspx">  
</asp:Button>
```

4. In the **Second.aspx** page, drag a **Label** and change the **Text** property to '**Cross-Page Posting**'. Rename the **Label** as **lblText**.

The resulting code will be as shown in the following Code Snippet:

Code Snippet:

```
<asp:Label ID="lblText" runat="server" Text="Cross-Page Posting">  
</asp:Label>
```

5. Set **First.aspx** as the Start page. Save, build, and execute the application. **First.aspx** page will be displayed.

When you click the **Button** on **First.aspx**, the **PostBackUrl** property will display the **Second.aspx** page with the **Label** text as shown in figure 15.1.

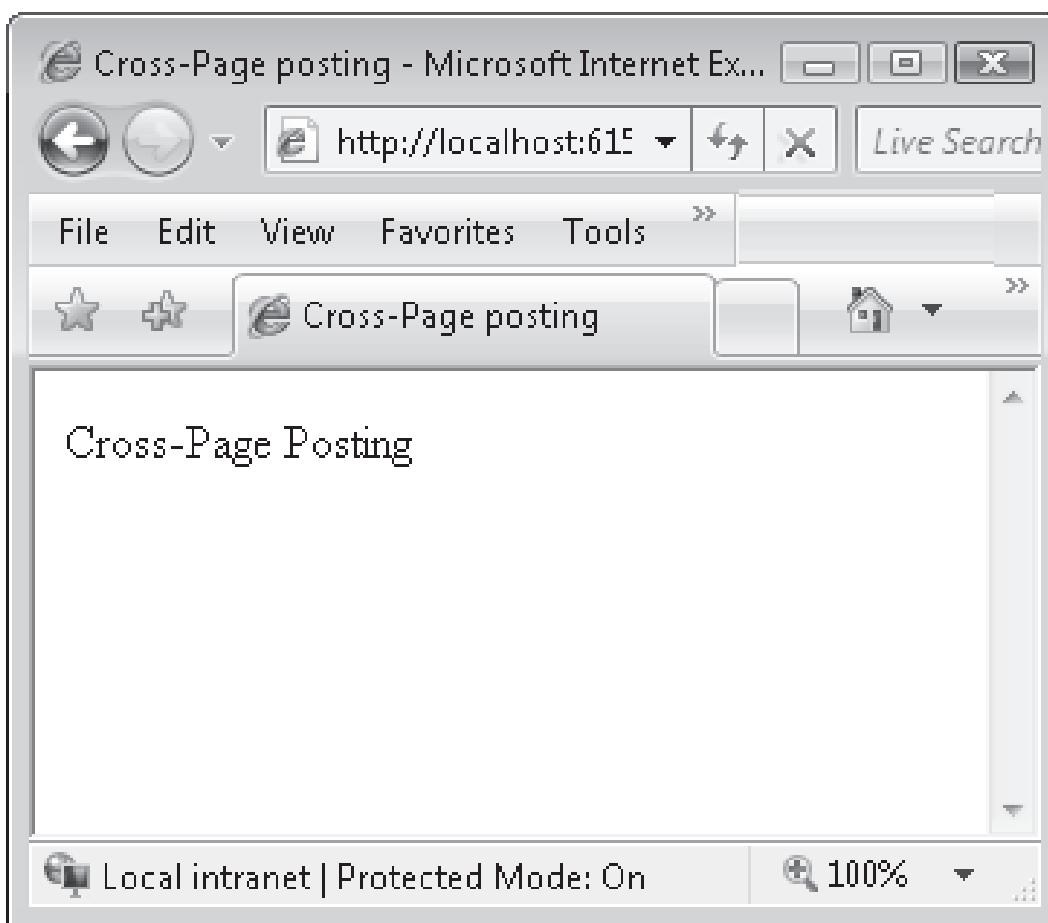


Figure 15.1: Second.aspx Page

15.2 Server.Transfer() Method

In addition to cross-page posting, there are also other ways in which ASP.NET enables you to send content of one page to another page. The **Server.Transfer()** method is one such approach.

The **Server.Transfer()** method sends the data of a Web page to another Web page. This method allows sharing the states between pages easily. You have to pass two parameters to the

Server.Transfer() method. The first parameter is the name of the Web Form to which the control has to be passed and the second is a Boolean value to maintain the state of the current Web Form.

The following steps will show how to use the **Server.Transfer()** method:

1. Create an ASP.NET Web site called **DataTransfer** and add two pages named **Home.aspx** and **Register.aspx** respectively.

2. Drag a **Button** control on **Home.aspx** page and set the properties as shown in the following Code Snippet:

Code Snippet:

```
<asp:Button ID="btnLogin" runat="server" Text="Login" BackColor="Silver">
</asp:Button>
```

3. Generate the Click event handler for **btnLogin**. Add the code as shown in the following Code Snippet:

Code Snippet:

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    Server.Transfer("Register.aspx", true);
}
```

The code shows the use of `Server.Transfer()` method in the event handler. The `Server.Transfer()` method is invoked with two parameters. The first parameter is a file name, **Register.aspx**, and the second parameter is a Boolean value, true. When you click the **Login** button, the `Server.Transfer()` method will transfer control to another page named **Register**. Since the second parameter is true, the state of the current Web Form will be maintained even after the control is transferred to the next page.

4. Drag two labels, text boxes, and a Button control on **Register.aspx** and change the properties as shown in the following Code Snippet:

Code Snippet:

```
<asp:Label ID="lblName" runat="server" Text="User name:>
</asp:Label>
<asp:TextBox ID="txtName" runat="server">
</asp:TextBox>
<asp:Label ID="lblEmail" runat="server" Text="Email Id:>
</asp:Label>
<asp:TextBox ID="txtEmail" runat="server">
</asp:TextBox>
<asp:Button ID="btnRegister" runat="server" Text="Register">
</asp:Button>
```

5. Set **Home.aspx** as the Start Page. Execute the application. Click the **Login** button. It will display the output as shown in figure 15.2.

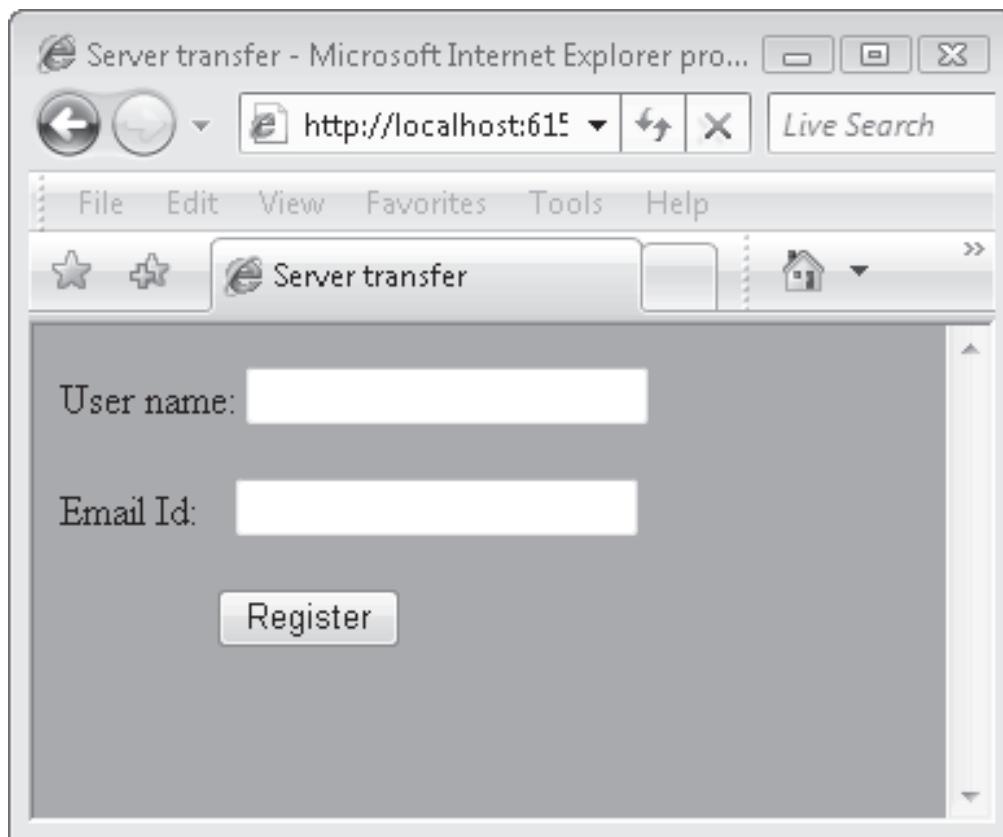


Figure 15.2: Register.aspx Page

15.3 Session State Management in ASP.NET

In ASP.NET 3.5, Web applications are mainly based on stateless protocol which does not hold any information related to the request made by the users. If the states of the pages are maintained, then the information provided by the users can be reused.

Accordingly, the users do not require re-entering the same data repeatedly each time the page is received or sent back to the server.

15.4 Types of Session State Management

ASP.NET preserves the state of a Web page for a number of round trips made to the server using two types of state management.

Figure 15.3 shows the architecture of state management in ASP.NET.

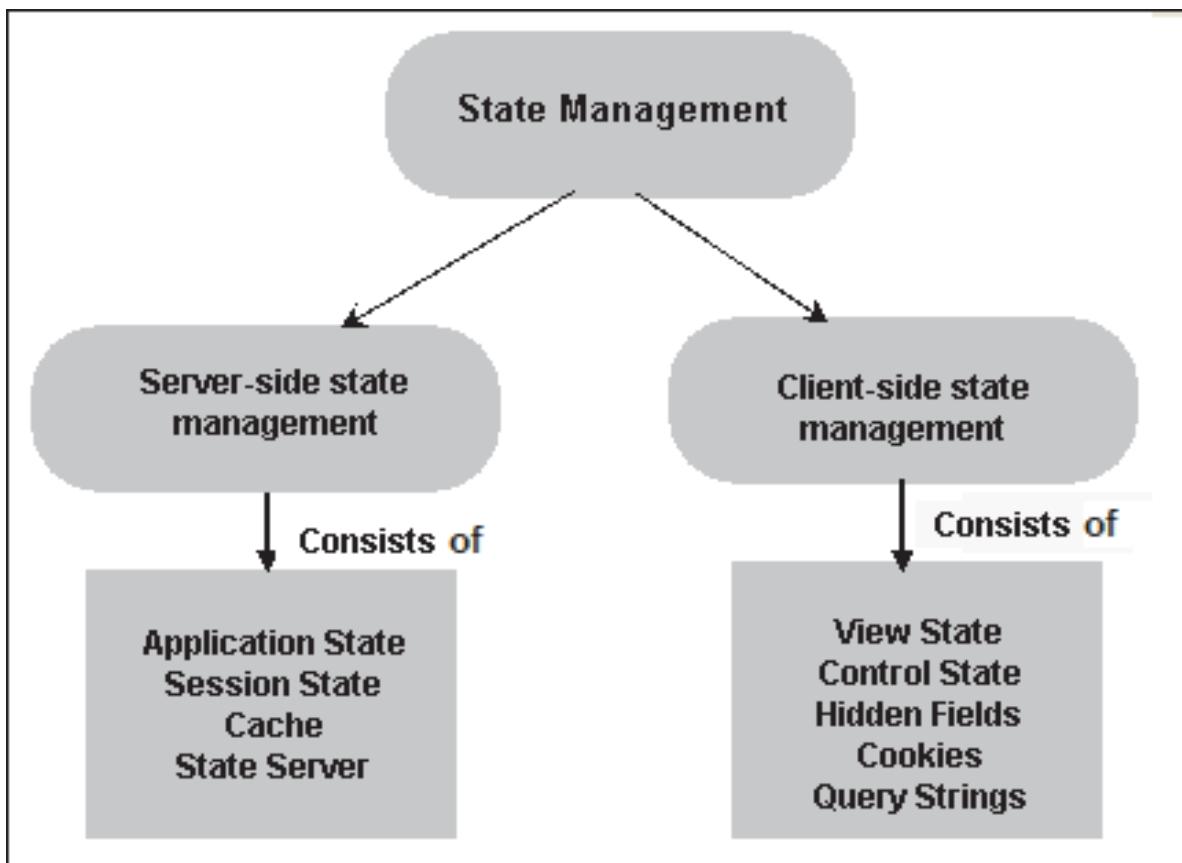


Figure 15.3: State Management Architecture

15.4.1 Server-side State Management

In server-side state management, data is stored by using server resources. Server-side state management has provided options that comprise the following:

- **Application State:** Data is made available to all users of the Web application. For example, if you want to keep track of all users visiting your Web site, you can store the total number of visitors who have accessed the Web site.
- **Session State:** Data is made available only to a particular user of a specific session in a Web application. For example, you can save the themes applied by the user.
- **StateServer or Microsoft SQL Server:** These states use database technology. A StateServer or a Microsoft SQL Server database can be used for storing large amount of information for a particular user. It is not necessary to have the database on the same server as your Web application.
- **Cache:** This object can be used to manage the state at the application level.

15.4.2 Client-side State Management

In client-side state management, data is not maintained by the server for storing the state information. This approach provides minimum security and faster performance. Client-side state management provides different options as compared to server-side state management to preserve the state of the application.

Some of the options in client-side state management include:

- **Cookies:** A cookie is a small plain text file that contains the data to be stored and maintained.
- **ViewState:** A ViewState retains the values between the multiple requests made for the same page. This property is maintained as a hidden field in the page and is enabled by default.
- **Query Strings:** A query string is the information appended at the end of a URL.
- **Hidden Fields:** A hidden field is used to store the values of a control that persists throughout many postbacks made to the server.
- **Control State:** A control state is used to preserve the information specific to a particular control.

15.5 Session State in ASP.NET

Session state provides various alternatives for storing the session data. Each alternative is described by a value in the `SessionStateMode` enumeration. The settings for session state are configured with the `web.config` file. Table 15.1 describes the members of the `SessionStateMode`.

Mode	Description
Off	Session state is disabled
InProc	Session state is stored in process with an ASP.NET worker process
StateServer	Session state is using the out-of-process ASP.NET state service to store state information
SQLServer	Session state is using an out-of-process SQL Server database to store state information
Custom	Session state is using a custom data store to store session-state information

Table 15.1: Members of `SessionStateMode` Enumeration

15.5.1 In-Process Mode

The in-process mode is the default mode in session state and is defined by using the `InProc` enumeration value of `SessionStateMode`. In this mode, the session state value and variables in memory are stored on the local Web server. This mode supports the `Session_OnEnd` event.

The following code snippet shows the configuration settings for the in-process mode. This code will be written in the `web.config` file:

Code Snippet:

```
<system.web>
    <sessionState mode="inproc" cookieless="false" timeout="20"/>
</system.web>
```

The code is used to configure the ASP.NET session state. It includes the following attributes:

- **mode:** This attribute states the type of mode used for session state.
- **cookieless:** This attribute accepts a Boolean value of true or false.
- **timeout:** This attribute sets the timeout period for a session. The session timeout indicates the duration of time for which a session is kept alive before it is timed out. Here, the session will expire in 20 minutes.

Figure 15.4 shows the flow of the in-process mode.

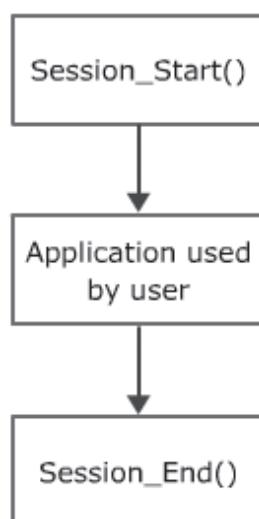


Figure 15.4: In-Process Mode

15.5.2 State Server Mode

The StateServer mode stores the state of a session in a process which is named as an ASP.NET state service. This is different from the ASP.NET worker process or IIS application pool. With this StateServer mode, you can ensure that the session state is saved even if your application restarts and it makes the session state easily available to the Web farm for multiple Web servers. A Web farm is a group of Web servers managed locally. If you are using StateServer mode, you have to check whether the ASP.NET state service is executing on the server used for storing the session. When you install the .NET Framework, the ASP.NET state service also is installed. You can also start the ASP.NET state service by performing the following steps:

1. Go to **Control Panel**.
2. Select **Administrative Tools → Services**.

Figure 15.5 displays the Services dialog box where the ASP.NET state service is selected.

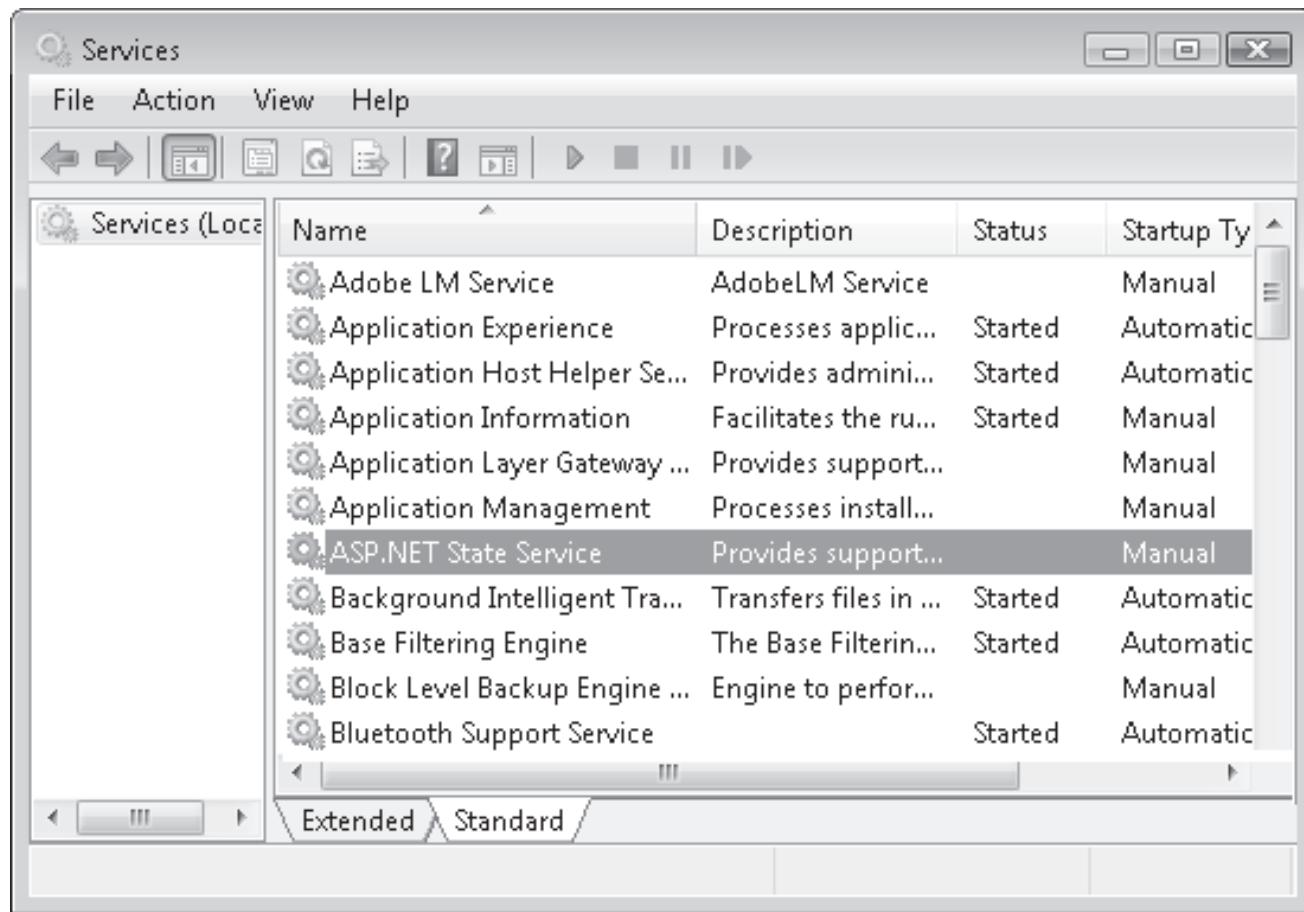


Figure 15.5: ASP.NET State Service

3. Start the service. The ASP.NET state service can be configured by using the StateServer mode in the **web.config** file.

The following code snippet shows configuration settings in the **web.config** file for StateServer mode where session state is stored on the remote system named as **StateServerExample**:

Code Snippet:

```
<system.web>
  <sessionState mode="StateServer" stateConnectionString="tcpip=
    StateServerExample:24242" cookieless="false" timeout="30"/>
</system.web>
```

15.5.3 SQL Server Mode

The SQL Server mode stores the state of a session in a SQL Server database. When this mode is used, it will ensure that the state of a session is saved even if the Web application restarts. The session state is made available to all Web servers in the Web farm. If you are using SQLServer mode, then you have to ensure that the ASP.NET session database is installed on SQL Server. The **aspnet_regsql.exe** tool is used to install the ASP.NET session state database. The ASP.NET application can be configured for using the SQLServer mode in file.

The following code snippet displays the configuration settings in the **web.config** file for SQLServer mode where the state of a session is stored on a SQL Server database named **SqlServerExample**:

Code Snippet:

```
<system.web>
  <sessionState mode="SQLServer"
    sqlConnectionString="Integrated Security=SSPI;data
    source=SqlServerExample;" />
</system.web>
```

You can run the **aspnet_regsql.exe** tool to install the session state database on SQL Server. This tool will create a database named **ASPState** that contains the stored procedures which supports the SQLServer mode. By default, the session data is stored in **tempdb** database. You can also change the storage location of session data by using the **-sstype** option.

Table 15.2 defines the possible values for **-sstype** option.

Options	Description
T	This is the default option that stores the session data in a SQL Server tempdb database. The session data will be lost if the SQL Server is restarted
P	This option stores the data in ASPState database
C	The session data is stored in the custom database

Table 15.2: **sstype** Command Options

The syntax to create a database on a SQL Server instance is as follows:

Syntax:

```
aspnet_regsql.exe -S <servername> -E <user> -U <userid>
-P <password> -ssadd -sstype p
```

where,

- S: specifies the name of the SQL Server object
- E: authenticates the current Windows user credentials
- U: specifies SQL Server user ID
- P: specifies the password
- ssadd: specifies support to add session state database
- sstype t | p | c: specifies the type of session state support

For example, if you want to create a database called **ASPState** on a SQL Server instance named SqlServerExample, you need to use the command shown in the following Code Snippet:

Code Snippet:

```
aspnet_regsql.exe -S SqlServerExample -E -ssadd -sstype p
```

15.5.4 Custom Mode

The **Custom** mode allows you to store session data by using a custom session state provider. If you want to change the mode to **Custom**, then you need to define the type of session state provider using the **sessionState** configuration element. In this element, you will create a child element, **providers**, and set the type of provider and the name attribute specifying the provider instance name.

The name of the custom provider object is then, assigned to the **customProvider** attribute of the **sessionState** element. If you want to configure ASP.NET session state, use the provider object for retrieving and storing session data.

The following code snippet shows an example of a custom session state provider in the **web.config** file. The custom session state provider uses the ODBC.NET Framework provider to manage session information using a Microsoft Access database.

Code Snippet:

```
<configuration>
    <connectionStrings>
        <add name="dbServices" connectionString="DSN=SessionStore;" />
    </connectionStrings>
    <system.web>
        <sessionState mode="Custom" customProvider="dbSessionProvider">
            <providers>
                <add name="dbSessionProvider"
                    type="Samples.AspNet.Session.OdbcSessionStateStore"
                    connectionStringName="dbServices"
                    writeExceptionsToEventLog="false" />
            </providers>
        </sessionState>
    </system.web>
</configuration>
```

15.5.5 Off Mode

The **Off** mode will disable the session for the application. You can configure the settings for this mode in the `web.config` file. The following code snippet shows the `Off` mode for the session state of an application:

Code Snippet:

```
<configuration>
    <system.web>
        <sessionState mode="Off" />
    </system.web>
</configuration>
```

15.6 Setting Session Timeout

The session timeout indicates the duration of time for which a session is kept alive before it is timed out. Once a session is timed out, the state is no longer retained. You can configure the timeout of a session in `web.config` file by setting the `timeout` attribute. The session event `Session.End()` is called when the session timeout period is over. The following code snippet shows how to set the session timeout period for 30 minutes in the `web.config` file:

Code Snippet:

```
<configuration>
  <system.web>
    <sessionState timeout="30"></sessionState>
  </system.web>
</configuration>
```

Another way of setting timeout value is through the `Session.Start` event. The following code snippet demonstrates this.

Code Snippet:

```
protected void Session_Start(Object sender, EventArgs e)
{
  Session.Timeout = 30;
}
```

15.7 Retrieving Values from Cookies

Cookies are small text files which are stored on the client machine. Cookies can be created by using the `Cookies` collection of the `Request` and `Response` objects. You can create two types of cookies, a temporary cookie and a persistent cookie. Temporary cookies are deleted from the memory when you close the browser. These cookies can be converted into persistent cookies by adding expiration time. The persistent cookies are deleted once the expiration time is over. The `Cookies` property specifies a group of cookies, each of which is an object of the `HttpCookieCollection` class. You can retrieve the value of a cookie by accessing the cookie collection of the `Request` object.

The following code snippet shows how to retrieve the values from a cookie:

Code Snippet:

```
HttpCookie mCookie = new HttpCookie("CookieName");

DateTime dt = DateTime.Now;

Response.Cookies.Add(mCookie);

mCookie.Values.Add("BackColor", "Gray");

mCookie.Values.Add("ForeColor", "Black");

mCookie.Values.Add("Time", dt.ToString());

lblTime.BackColor = System.Drawing.Color.FromName(mCookie.Values

    ["BackColor"]);

lblTime.ForeColor = System.Drawing.Color.FromName(mCookie.Values

    ["ForeColor"]);

lblTime.Text = mCookie.Values["Time"];
```

In the code, an `HttpCookie` object is created and the properties such as `BackColor`, `ForeColor`, and `Text` of the `Label` control are stored in the cookie by using key/value pairs. Later, the cookie values are retrieved and are assigned to the respective properties of the `Label` control.

Figure 15.6 displays the Label control whose properties have been set at runtime using cookies.

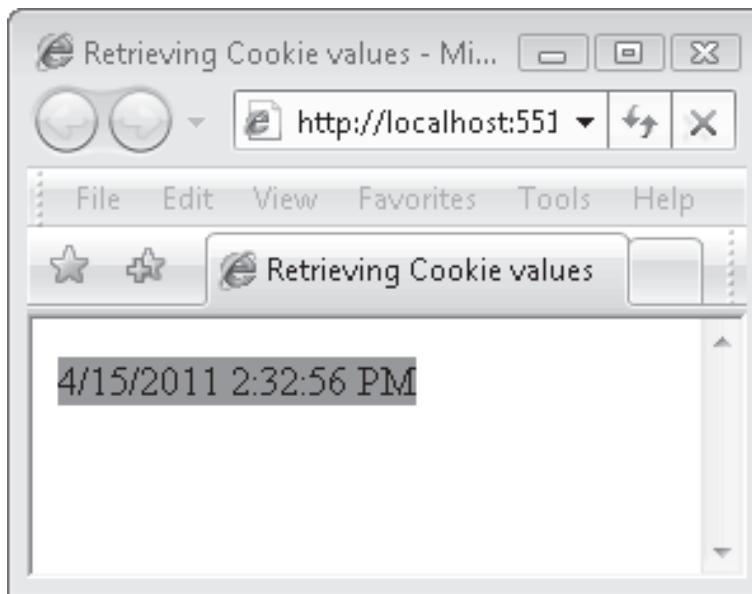


Figure 15.6: Retrieving Values from Cookies and Displaying in Label

15.8 Using Cookieless Sessions

In many scenarios, if important and critical data is accessed through cookies, it can cause harm to the local machine. There might also be situations where a browser may not support cookies or the client has turned off the cookies option from the settings of the application. The client-side applications will be affected if they are dependent on cookies. Often, cookies store session id at the client side and if your cookie is disabled, you cannot retrieve the data from that session.

For these reasons, ASP.NET provides support for cookieless sessions. When you select a cookieless session, the session id will be transported through the request URL. Each request of the Web application holds the session id embedded with its URL such that the application does not have to request the session from the cookies.

The following code snippet shows you how to enable a cookieless session:

Code Snippet:

```
<sessionState cookieless="true"/>
```

The code sets the `cookieless` attribute value to true in the `web.config` file and thus, enables a cookieless session.

15.9 Check Your Progress

1. Which of the following statements about Server.Transfer() method are true?

(A)	Server.Transfer() method transfers the data from one Web page to another
(B)	Server.Transfer() method does not allow sharing of states between pages
(C)	Server.Transfer() method takes two parameters
(D)	Server.Transfer() method uses a PostBackURL method

(A)	a, c	(C)	a, b
(B)	b, d	(D)	c, d

2. Which are the two types of state management in ASP.NET?

(A)	Server Management
(B)	SQL Server Management
(C)	Server-side Management
(D)	Client-side Management

(A)	a, c	(C)	a, b
(B)	b, d	(D)	c, d

3. Which of the following options belong to server-side state management?

(A)	Cookies
(B)	StateServer
(C)	Application State
(D)	Server State

(A)	b, c	(C)	b, c, d
(B)	c, d	(D)	a, d

4. Which of the following statements about the Custom mode are true?

(A)	Custom mode allows you to store session data by using a custom session state provider
(B)	Custom mode stores the session state values and variables on the local Web server
(C)	Custom mode does not store the data in SQL Server database

(D)	Custom mode sets the provider and attribute name that specifies the provider instance name		
-----	--	--	--

(A)	a, b	(C)	b, c
(B)	a, d	(D)	b, d

5. Which of the statements about the SQL Server mode are true?

(A)	SQL Server mode stores the state of a session in a SQL Server database		
(B)	SQL Server ensures that the state of a session is saved even if the Web application restarts		
(C)	SQL Server sets the attribute name and not the provider name		
(D)	SQL Server mode is accessible to all the users		

(A)	a, c	(C)	a, b
(B)	b, d	(D)	c, d

6. Which of the following are the session events in ASP.NET?

(A)	Session.Start()		
(B)	Sesion.End()		
(C)	Session.Begin()		
(D)	Sesion.Stop()		

(A)	a, c	(C)	c, d
(B)	b, d	(D)	a, b

Module Summary

In this module, **Working with ASP.NET 3.5**, you learnt about:

→ **Working with ASP.NET**

Cross-page posting is an approach used for submitting the contents of a Web page to another page. Server.Transfer() method transfers control from one Web page to another page and shares the states between the Web pages. State management in ASP.NET enables you to preserve the states of a session. In-process, StateServer, SQLServer, Custom, and Off Session are the session state modes in ASP.NET. The timeout attribute is used to configure the session timeout in the web.config file. ASP.NET supports cookieless sessions for scenarios where a cookie should not be or cannot be used.

Module - 16

Working with ASP.NET 3.5 (Lab)

Welcome to the module, **Working with ASP.NET 3.5 (Lab)**.

In this module, you will learn to:

- Use Cross-page posting
- Work with storing and retrieving values from cookies
- Work with State Server mode in Session state

16.1 Part I – 60 Minutes

Exercise 1

Gallant Technologies Private Limited is a software company located in Switzerland. The company is the market leader in .NET technologies such as ASP.NET and ADO.NET. They are developing both Windows and Web-based applications. The company wants to create a profile of all their employees. The employee details have to be filled in the Employee Form. This form is available to all the employees on the company's Web site so that the information can be collected easily.

Consider that you are a developer in the software team and are assigned the following tasks:

- ➔ Create an Employee form to accept data from employees
- ➔ Ensure that each employee should enter the name and designation. These fields are mandatory
- ➔ Add Appropriate validations to validate the details of the employees
- ➔ Ensure that the employee details should be stored on the local machine using cookies
- ➔ Retrieve the employee details using cookies
- ➔ You must create the Web application using Visual Studio 2008 IDE and ASP.NET 3.5.

Solution:

The details of each employee will be stored and retrieved using cookies.

To create application Web site named Gallant Technologies, perform the following steps:

1. Create a Web site named **Gallant Technologies** as shown in figure 16.1.

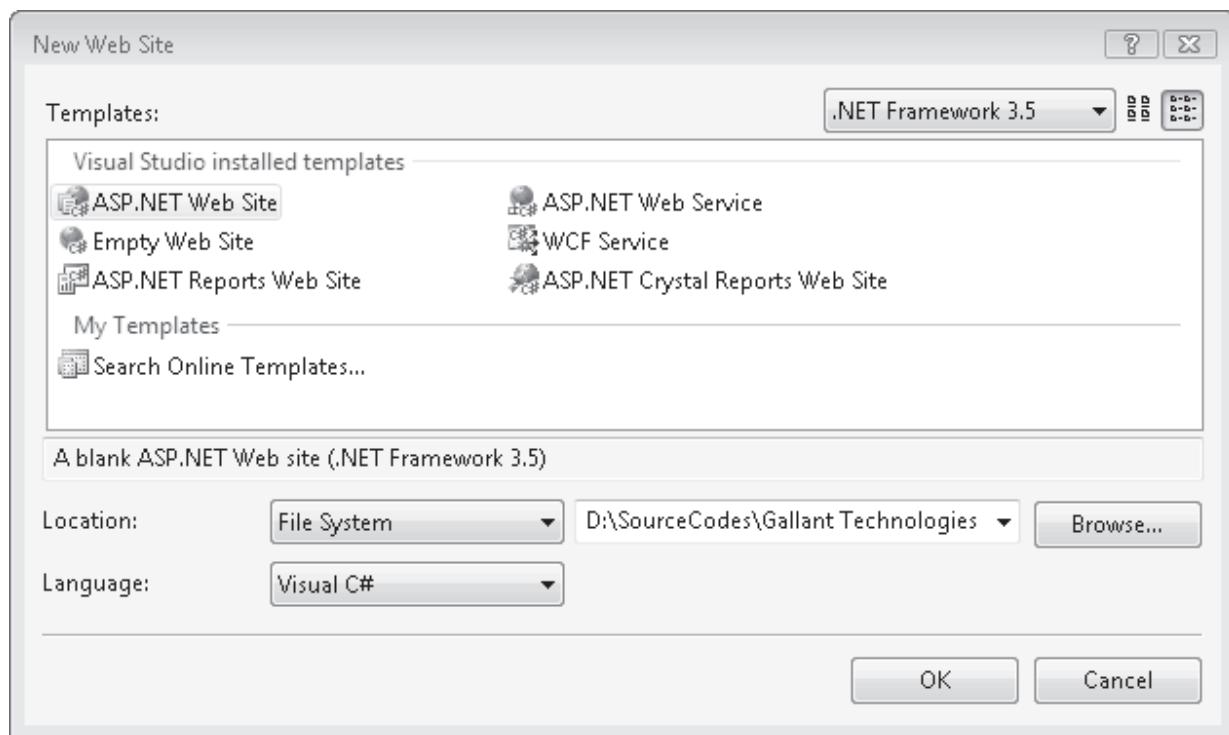


Figure 16.1: Creating a Web Site

2. Rename **Default.aspx** as **Employee.aspx**. Also, rename all instances of **Default** to **Employee**.

3. Design the user interface as shown in figure 16.2.

The screenshot shows a web page titled "Gallant Technologies". At the top, there is a text input field labeled "Employee Name:" with a placeholder "Employee Name". Below it is another text input field labeled "Employee Designation:" with a placeholder "Employee Designation". A "Submit" button is located next to the designation input field. At the bottom of the page, there are three separate lines of text: "Enter employee name", "Enter the designation", and "Display the employee details".

Figure 16.2: User Interface of Employee Page

4. Select the text boxes, button, and hyperlink controls and configure the properties as shown in table 16.1.

Control	Property	Value
Text box	ID	txtName
RequiredFieldValidator	ID	valName
	SetFocusOnError	true
	ErrorMessage	Enter employee name
	ControlToValidate	txtName
Text box	ID	txtDesignation
RequiredFieldValidator	ID	valDesignation
	SetFocusOnError	true
	ErrorMessage	Enter the designation
Button	ID	btnSubmit
	Text	Submit

Control	Property	Value
	BackColor	#FFFFCC
	ForeColor	Black
HyperLink	ID	InkDisplay
	NavigateUrl	EmployeeDetails.aspx
	Visible	false
	Text	Display the employee details

Table 16.1: Properties of the Employee Page Controls

5. Add the following code in **Page_Load** event of the **Employee.aspx** page.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        lnkDisplay.Visible = true;
        lnkDisplay.NavigateUrl = "EmployeeDetails.aspx?cookie=" +
            txtName.Text.ToString();
        btnSubmit.Enabled = false;
    }
}
```

Initially, the page will not display the link in the browser. After the details are entered by the employee and the Submit button is clicked, a postback occurs. Then, the link is visible. Also, the destination URL of the hyperlink is set to EmployeeDetails.aspx followed by cookie and the value of the text box, Name. This value will be the employee name. The button, Submit, is disabled because details are already entered.

6. Generate the Click event of the button, **btnSubmit**. Add the following code to the event handler.

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    //Create a new cookie, passing the name into the constructor
    HttpCookie objHttpCookie = new HttpCookie(txtName.Text);
```

```

//Set the cookies value
objHttpCookie.Value = txtDesignation.Text;
//Set the cookie to expire in 2 minutes
DateTime dt = DateTime.Now;
TimeSpan ts = new TimeSpan(0, 0, 2, 0);
objHttpCookie.Expires = dt + ts;

//Add the cookie
Response.Cookies.Add(objHttpCookie);
Response.Write("Employee Details saved. <br><hr>");
}

```

In this code, a cookie is created to store the employee details for two minutes. The employee details will be retrieved from the Web Form, stored into the cookies, and a message will be displayed indicating that employee details are saved.

7. Click **File** menu and select **New → File** sub-menu command.
8. Add a new Web Form and name it as **EmployeeDetails.aspx**.
9. Switch to **Design View** of **EmployeeDetails** page.
10. Add a **Label** and a **Button** control in the **EmployeeDetails** page.
11. Specify the **Button ID** as **btnDetails** and **Text** as **Back**.
12. Specify the **Label ID** as **lblDetails** and delete the default text.

The resulting user interface is displayed as shown in figure 16.3.



Figure 16.3: User Interface of EmployeeDetails Page

13. Add the following code to the **Page_Load** event handler.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Request.QueryString["cookie"] != null)
    {
        ReadCookie();
    }
}
```

This code uses the `QueryString` to receive the cookie and check if the cookie is not null. If the cookie is not null, it will call the `ReadCookie()` method.

14. Create a method named **ReadCookie()** and add the following code:

```
protected void ReadCookie()
{
    // Retrieve the name entered by the user from the cookie
    String strCookieName = Request.QueryString["cookie"].ToString();

    // Hold the cookie value
    HttpCookie objCookieValue = Request.Cookies[strCookieName];
    // Check to make sure the cookie object exists
    if (objCookieValue == null)
    {
        lblDetails.Text = "Cookie not found. <br>";
    }
    else
    {
        // Write the cookie value
        String strCookieValue = objCookieValue.Value.ToString();
        lblDetails.Text = "<center><b>The Details of the Employee are:</b>" + "<hr>" +
        "<br>" + "Employee Name:" + strCookieName + "<br>" + "Designation is:" +
        strCookieValue + "</b></center><br><hr>";
    }
}
```

The code will retrieve the name entered by the user from the cookie.

The code creates an `HttpCookie` to hold the retrieved cookie value. Then, it checks if the cookie is null. If the cookie is null, then it will display the message **Cookie not found**. Otherwise, it will

display the employee name and designation based on the cookie.

15. Generate the Click event of the button, btnDetails. Add the following code to the event handler.

```
protected void btnDetails_Click(object sender, EventArgs e)
{
    if (PreviousPage != null && PreviousPage.IsCrossPostBack)
    {
        TextBox txtEmpname = (TextBox) PreviousPage.FindControl("txtName");
        TextBox txtEmpdesignation = (TextBox) PreviousPage.FindControl
            ("txtDesignation");
        lblDetails.Text = "<b>Employee Name:</b> " + txtEmpname.Text
        + "<br>" + "<b>Designation:</b> " + txtEmpdesignation.Text;
    }
    else
    {
        Response.Redirect("Employee.aspx");
    }
}
```

This code checks if the previous page is not null and CrossPostBack has been performed. If these conditions are satisfied, the details will be fetched from the text boxes by using the FindControl method of **PreviousPage**. The details are assigned to new text box instances.

If the initial conditions are not satisfied, the user is redirected to **Employee** page.

16. Set the **Employee.aspx** as the start page.
17. Save, build, and execute the application. The **Employee** page is displayed.

18. Add the name and designation details of the employee as shown in figure 16.4.



Figure 16.4: Employee Page

19. Click **Submit**. The form will display a link and the message '**Employee Details saved**' as shown in figure 16.5.

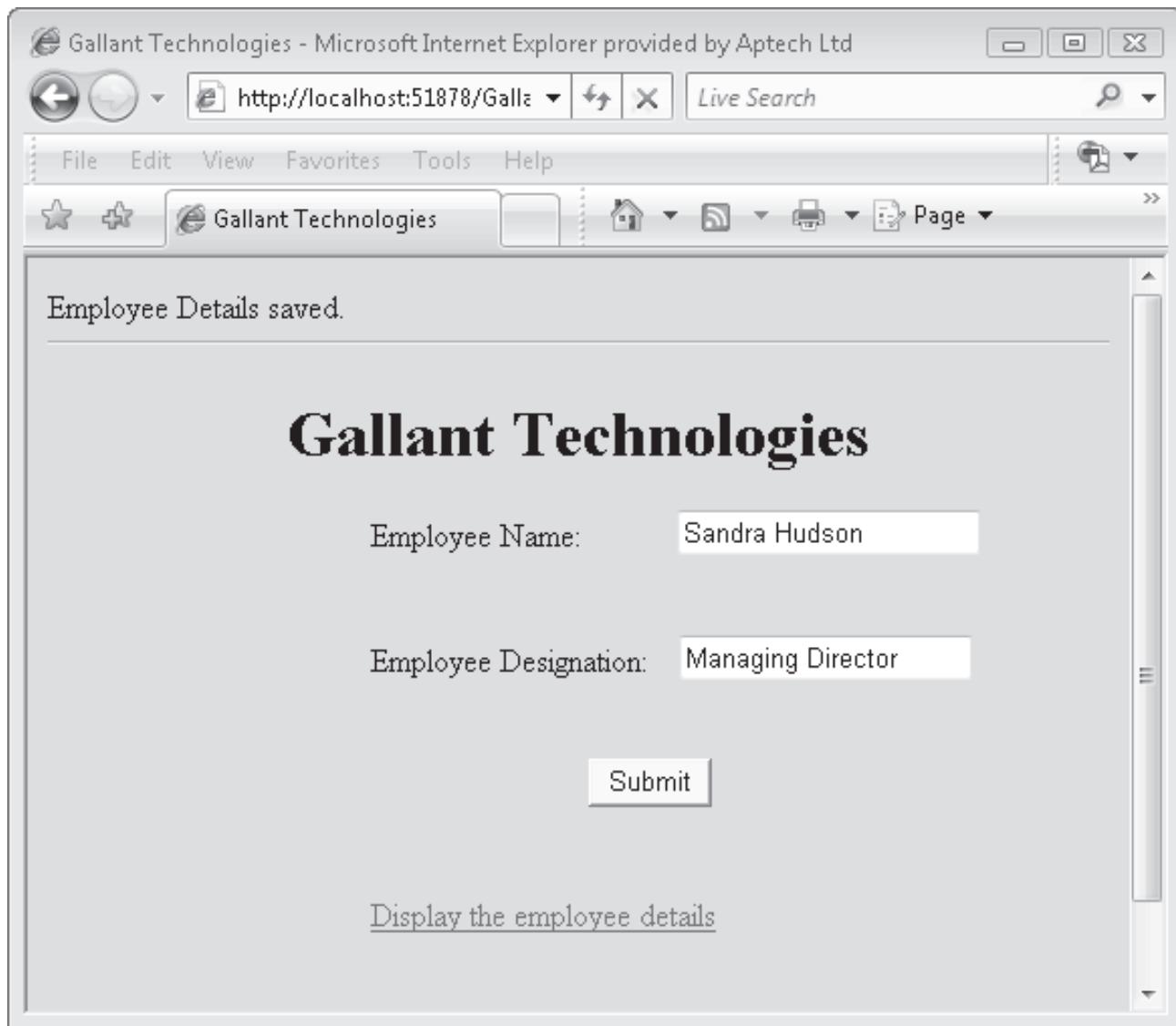


Figure 16.5: Displaying message in the Employee.aspx Page

20. Click 'Display the employee details'. The resulting output will be as shown in figure 16.6.



Figure 16.6: EmployeeDetails Page

The **EmployeeDetails** page will display the employee name and designation.

Exercise 2

OrpiaSolutions Company is a software development company located in Germany. The company has a good reputation in the global market for developing excellent projects. They have a wide range of expertise and experience in .NET technologies. The company has decided to collect the information about their customers. The Customer Web page should accept the details of the customers and store the session of every customer who has logged in to the Web site.

Consider that you are one of the programmers in the development team and are assigned the following tasks:

- Create a Customer form to accept the data from the customer
- Create a CustomerDetails class to define properties for each customer
- Ensure that the customer details will be restored after they are stored
- Use the State Server mode
- Ensure that the ASP.NET state service is enabled
- You must create the Web application using Visual Studio 2008 IDE and ASP.NET 3.5 with administrator rights.

Solution:

1. Create a Hypertext Transfer Protocol (HTTP) Web site named OrpiaSolutions as shown in figure 16.7.

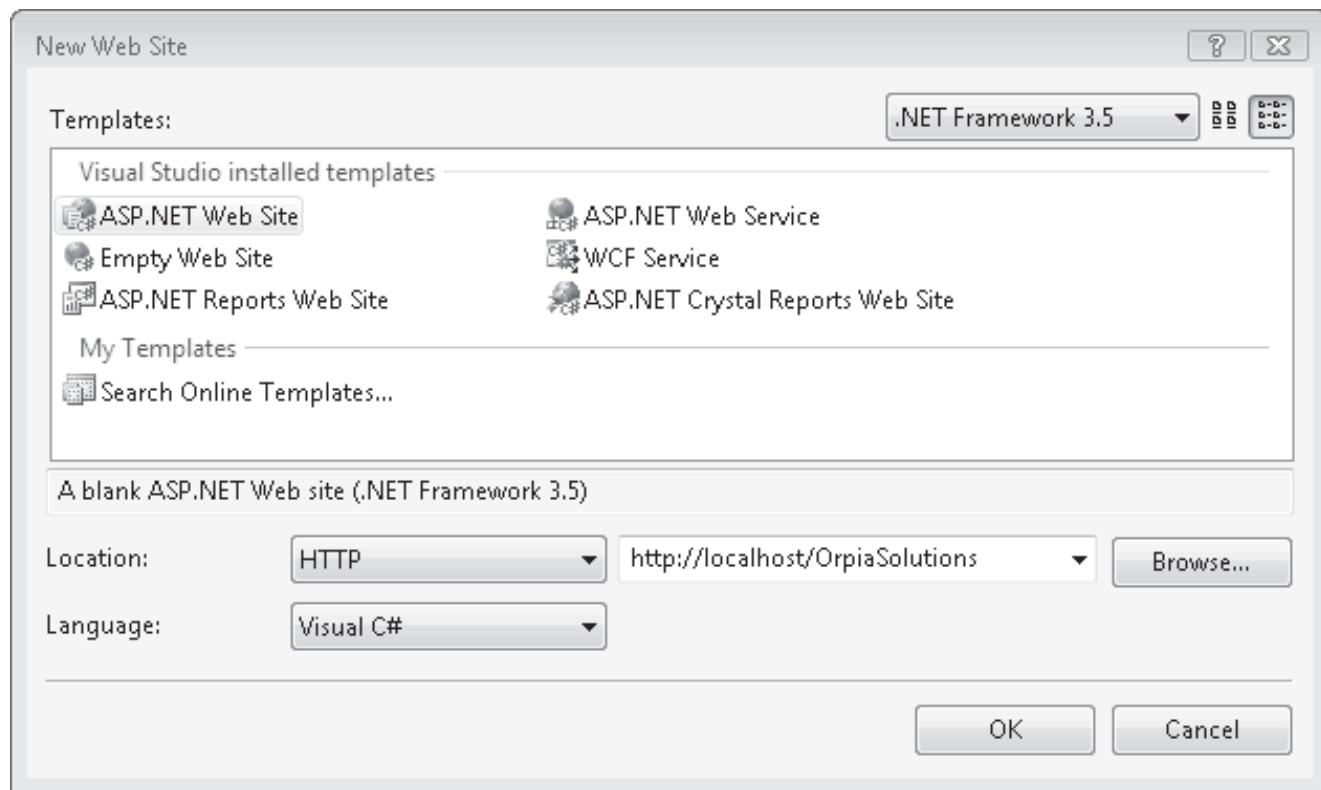


Figure 16.7: Creating an HTTP Web Site

2. Rename **Default.aspx** to **Customer.aspx**. Also, rename all instances of **Default** to **Customer**.
3. Create a new class named **CustomerDetails** with the following code:

Rename Default.aspx to Customer.aspx. Also, rename all instances of Default to Customer.

Create a new class named CustomerDetails with the following code.

```
/// <summary>
/// Summary description for CustomerDetails
/// </summary>
[Serializable]
public class CustomerDetails
{
    // Constructor
    public CustomerDetails ()
    {
```

```
// Constructor  
  
public CustomerDetails ()  
{  
    // TODO: Add constructor logic here  
}  
  
/// <summary>  
/// Create properties of CustomerDetails Class  
/// </summary>  
  
/// <param name="intId">Int Id</param>  
/// <param name="custName">String Name</param>  
  
public CustomerDetails (int intId, string custName)  
{  
    this.Id = intId;  
    this.custName = custName;  
}  
  
private int intId;  
private string custName;  
  
public int Id  
{  
    get;  
    set;  
}  
  
public string Name  
{  
    get;  
    set;  
}
```

This code will create a serializable class named **CustomerDetails**. This class creates two variables for the id and name of the customer. The `get` and `set` accessors are used to fetch and assign the values of the properties.

4. Open **Customer.aspx**.
5. Design the user interface as shown in figure 16.8.

The screenshot shows a web page titled "OrpiaSolutions". It contains two text input fields: one labeled "Customer Id:" and another labeled "Customer Name:". Below these fields are two buttons: "Submit" and "Restore". The entire form is enclosed in a dashed border.

Figure 16.8: User Interface of Customer Page

6. Select the labels, text boxes, and buttons and configure the properties as shown in table 16.2.

Control	Property	Value
Text box	ID	txtId
Label	ID	lblCustomerID
	Text	Customer Id:
Text box	ID	txtName
Label	ID	lblName
	Text	Customer Name:
Button	ID	btnSubmit
	Text	Submit
Button	ID	btnRestore
	Text	Restore

Table 16.2: Customer Page Controls

7. Generate the Click event of **btnSubmit** and add the following code:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    CustomerDetails objCustomer = new CustomerDetails(Int32.Parse
        (txtId.Text), txtName.Text);
    Session["objCustomer"] = objCustomer;
```

```
Response.Write("<b>Customer details:</b> " + "<br>");
Response.Write("Customer ID: " + txtId.Text + "<br>");
Response.Write("Customer Name: " + txtName.Text + "<br>");
txtId.Text = "";
txtName.Text = "";
}
```

This code will create an instance of **CustomerDetails** and store the values in the session object and the details will be displayed when the user or customer clicks the **Submit** button.

8. Generate the Click event of **btnRestore** and add the following code:

```
protected void btnRestore_Click(object sender, EventArgs e)
{
    CustomerDetails objCustomer = (CustomerDetails)Session["objCustomer"];
    txtId.Text = objCustomer.Id.ToString();
}
```

This code will restore the values of the customer id in the text box on the **Customer** page.

9. Add the following code in **web.config** file under **<system.web>** element:

```
<sessionState mode="StateServer" stateConnectionString="
tcpip=127.0.0.1:42424"/>
```

This statement will set the session state mode to **StateServer**. This means that the session state will use the **out-of-process** ASP.NET State service to store state information.

10. Open the Services window and select the **ASP.NET State Service** option as shown in figure 16.9.

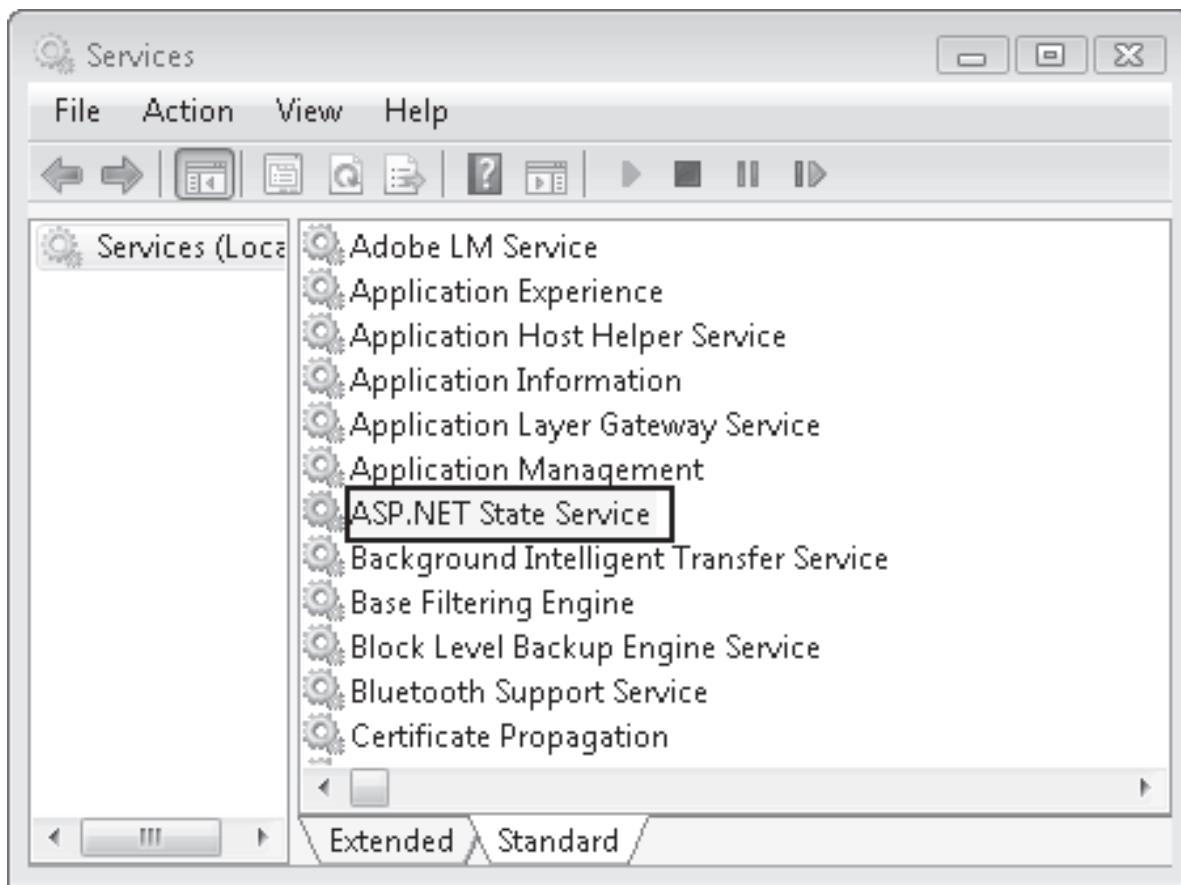


Figure 16.9: Services Dialog Box

11. Right-click the **ASP.NET State Service** option and select the **Start** option.

You have to ensure that **ASP.NET State Service** is running. In case it is not started, you have to manually start the service.

12. Execute the application. The resulting output will be as shown in figure 16.10.

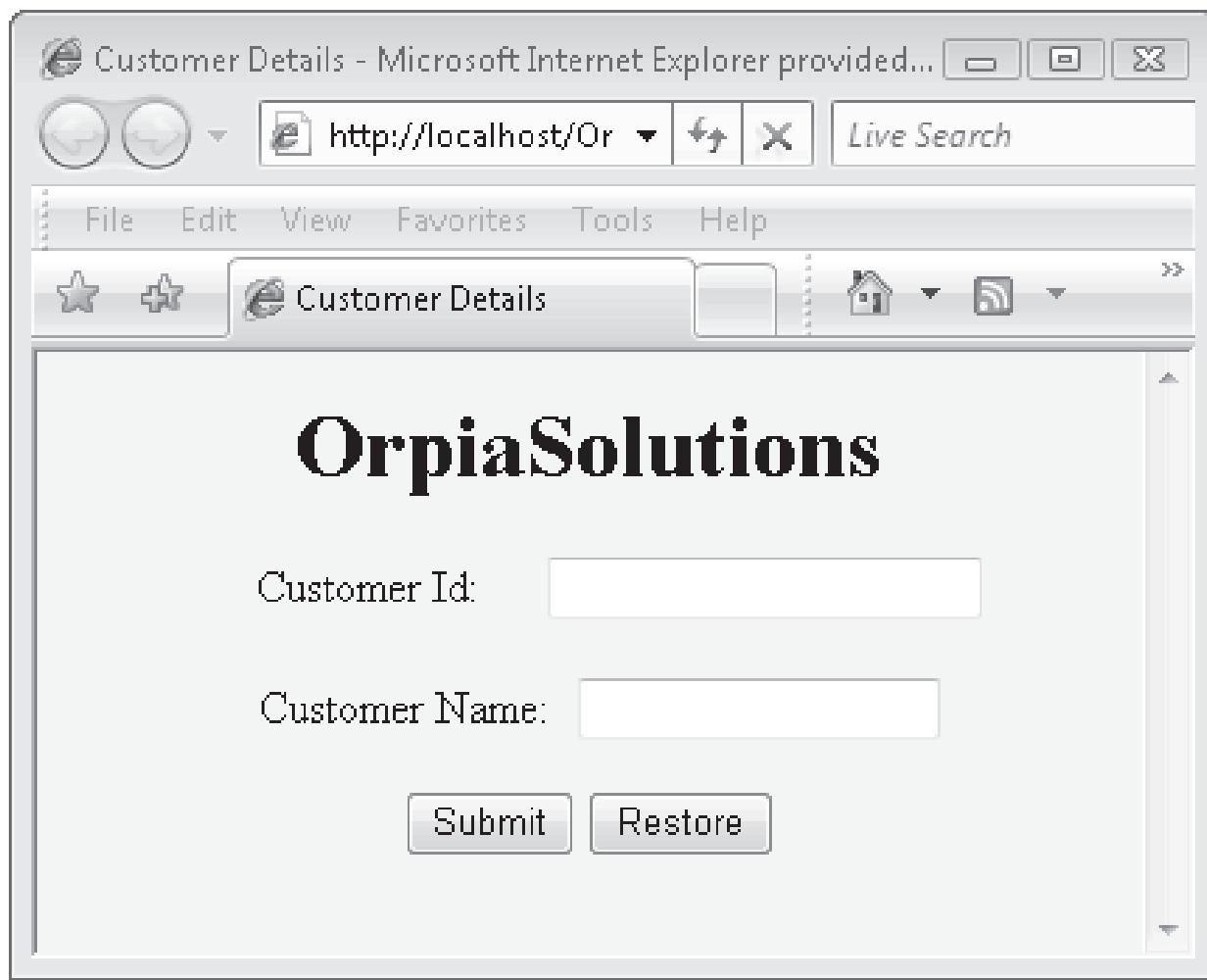


Figure 16.10: Customer Page

13. Enter the customer id and name in the respective text boxes.

14. Click **Submit**. The resulting output is displayed as shown in figure 16.11.

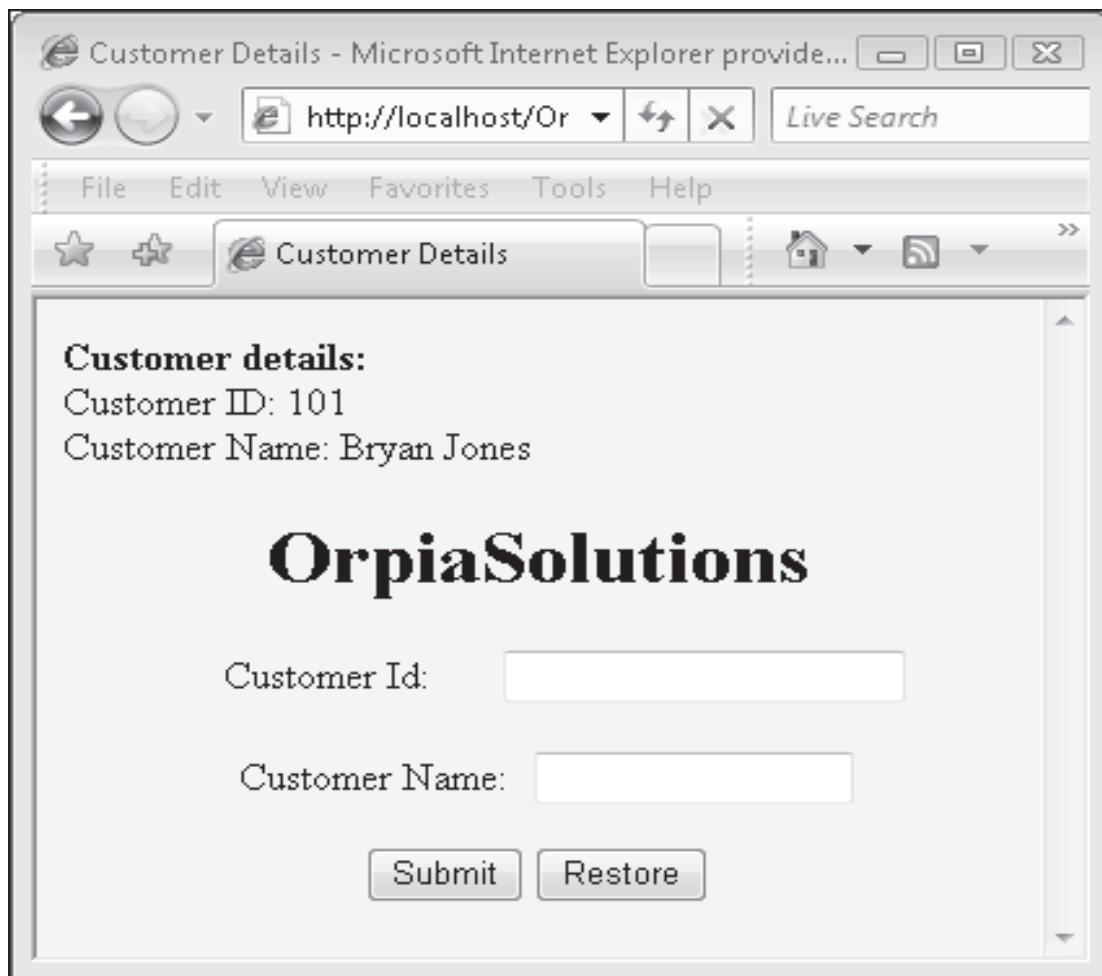


Figure 16.11: Displaying Customer Details

15. Click **Restore**.

- 16.

When you click the Restore button, the output will be as shown in figure 16.12. It restores the value of the customer id in the text box on the **Customer** page.

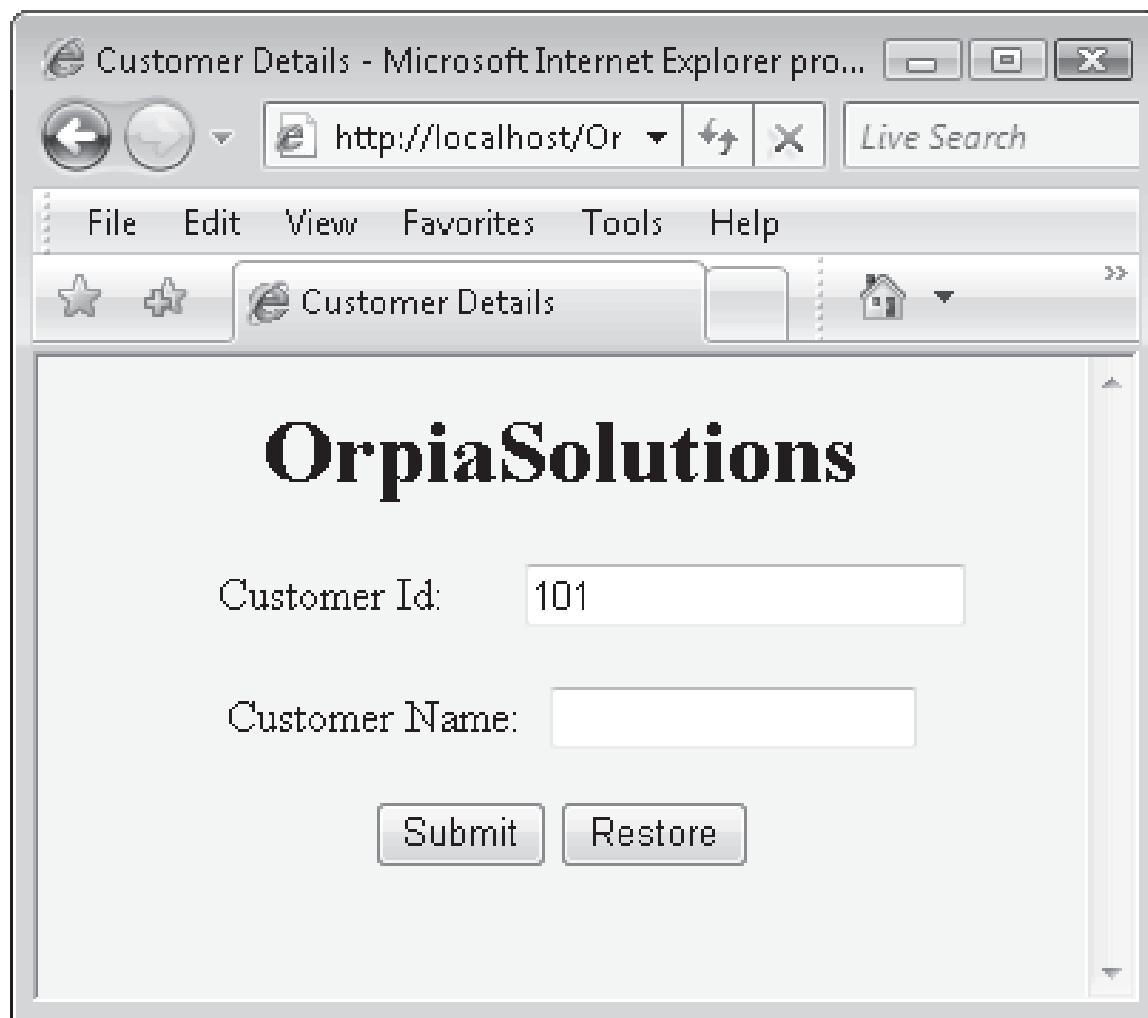


Figure 16.12: Customer Details Restored on the Form

It restores the value of the customer id in the text box on the **Customer** page.

16.2 Part II - 60 Minutes

Caroline Technologies is a leading company for developing Web-based applications. The company has got a new project to create an online shopping Web site for toys. The Web site should display the toys and price details. If customers want to purchase toys, they have to enter their name and contact details in the Registration form. The customer details must be temporarily stored on the local machine. Ensure that the correct data is submitted by the customers so that the toys can be delivered at their location.

Hints:

Create a Web site name Toy Shoppe

Create a registration Form for the customer

Use cookies for storing and retrieving values

16.3 Do It Yourself

Create a HTTP Web-based application for accepting details of books online. The book details must be restored on the book details form. Implement the StateServer mode of Session Management.

Module - 17

Authentication and Authorization

Welcome to the module, **Authentication and Authorization**. This module explores the security features in an ASP.NET application.

In this module, you will learn to:

- Describe Web application security
- Compare authentication and authorization
- List the ASP.NET authentication methods
- Explain Windows-based authentication in ASP.NET
- Explain Forms-based authentication in ASP.NET
- Explain IIS authentication mechanisms in ASP.NET

Web Development

http://www



17.1 Introduction

Security is an essential aspect to any application and ASP.NET provides various security options for Web site developers.

17.2 Security in Web Applications

Security is an important factor that you must consider when developing or maintaining a Web application. Each Web site has different security requirements. For example, consider a search engine, an online library, and an online shopping application. A typical search engine does not require any security features. An online library with user registration stores user names, passwords, and personal information. Although the information is not highly sensitive, you require some security to protect user details. However, for the shopping application, you require strong security to protect sensitive information, such as user's credit card details.

Web sites that are vulnerable to attacks require some security mechanism to allow only authorized users to access important information. ASP.NET provides security features that help you resolve this as well to enforce other security measures.

17.3 Security Features in ASP.NET Applications

The three fundamental security features for an ASP.NET application are as follows: authentication, authorization, and impersonation.

- **Authentication**
- Authentication is used to check the identity of a user before allowing or denying a request. For example, in an e-mail application, a user name and password are validated against a database of registered users. After verification, no further authentication is required to send and receive messages unless the user logs off from the application.
- **Authorization**
- Using authorization, only users with a valid identity can access specific resources in an application. For example, a student is not allowed to view examination records that a teacher or a Web administrator can access.

Figure 17.1 shows the steps for authentication and authorization.

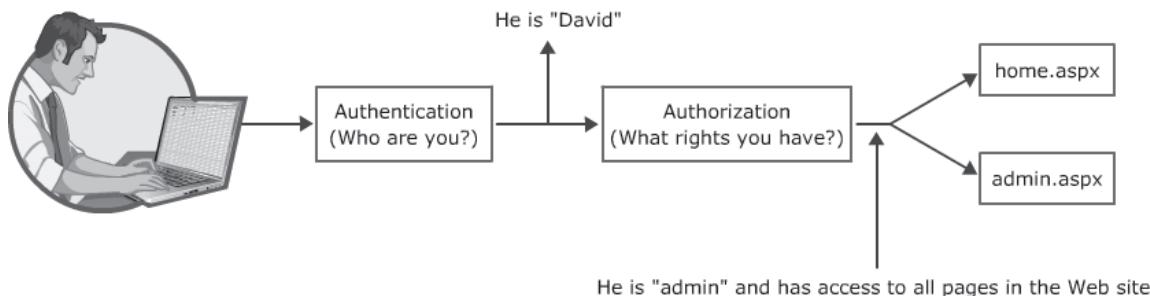


Figure 17.1: Authentication and Authorization

→ Impersonation

In impersonation, the ASP.NET application process acts on behalf of a user whose identity is authenticated using IIS. IIS passes an authentication token to the ASP.NET application. ASP.NET then uses the token and operates under the identity of the authenticated user.

17.4 ASP.NET Authentication Methods

ASP.NET implements authentication by using methods containing code to authenticate the credentials of the user.

Three types of authentication methods are supported in ASP.NET. They are as follows:

→ Windows-Based Authentication

Windows authentication is the default authentication method in ASP.NET. This type of authentication is based on a user's Windows accounts. Windows authentication uses IIS, which can be configured to allow only users on a Windows domain to log on to the application.

→ Forms-Based Authentication

Forms-based authentication uses the Forms authentication provider. In forms-based authentication, HTML forms are used to collect authentication information, such as user names and passwords. The application needs to have the code to verify the supplied credentials against a database. The credentials of an authenticated user can be stored in a cookie to be used during a session.

→ Microsoft Passport Authentication

In passport authentication, users are authenticated using the Passport Service provided by Microsoft. However, to use this type of authentication, you must be registered with Microsoft's Passport Service. The Passport server uses encrypted cookies to identify and validate users.

17.5 Authorization in ASP.NET

Authorization specifies whether an identity can be granted access to a specific resource. The two types of authorization available in ASP.NET are as follows:

→ File Authorization

This type of authorization uses NT File System (NTFS) permissions to check the access rights of the user account that the ASP.NET application is using. For example, if a user wants to open a particular file, the user account that is used to access the ASP.NET application must have read permission to that file.

→ URL Authorization

In the `web.config` file, the authorization rules for various folders or files of an application can be specified. Using the `<authorization>` element, you can specify the names of users who are allowed or denied access.

The syntax for the URL authorization is as follows:

Syntax:

```
<authorization>
  <[allow|deny] users roles verbs />
</authorization>
```

Here the `allow` or `deny` element is specified. The `allow` element grants the access and the `deny` element revokes the access. The `users` or the `roles` attributes need to be specified. You can include both or omit both as they are optional. The `verbs` attribute in the syntax is optional.

The following code snippet grants access to the `John` identity and members of the `Admin` roles, and denies access to the `David` identity (not in `Admin` role) and to all anonymous users.

Code Snippet:

```
<authorization>
  <allow users="John"/>
  <allow roles="Admin"/>
  <deny users="David"/>
  <deny users="?"/>
</authorization>
```

The users and roles can be created by using the **ASP.NET Web Site Administration Tool**.

Figure 17.2 shows the screen in **Web Site Administration Tool** for creating a user.

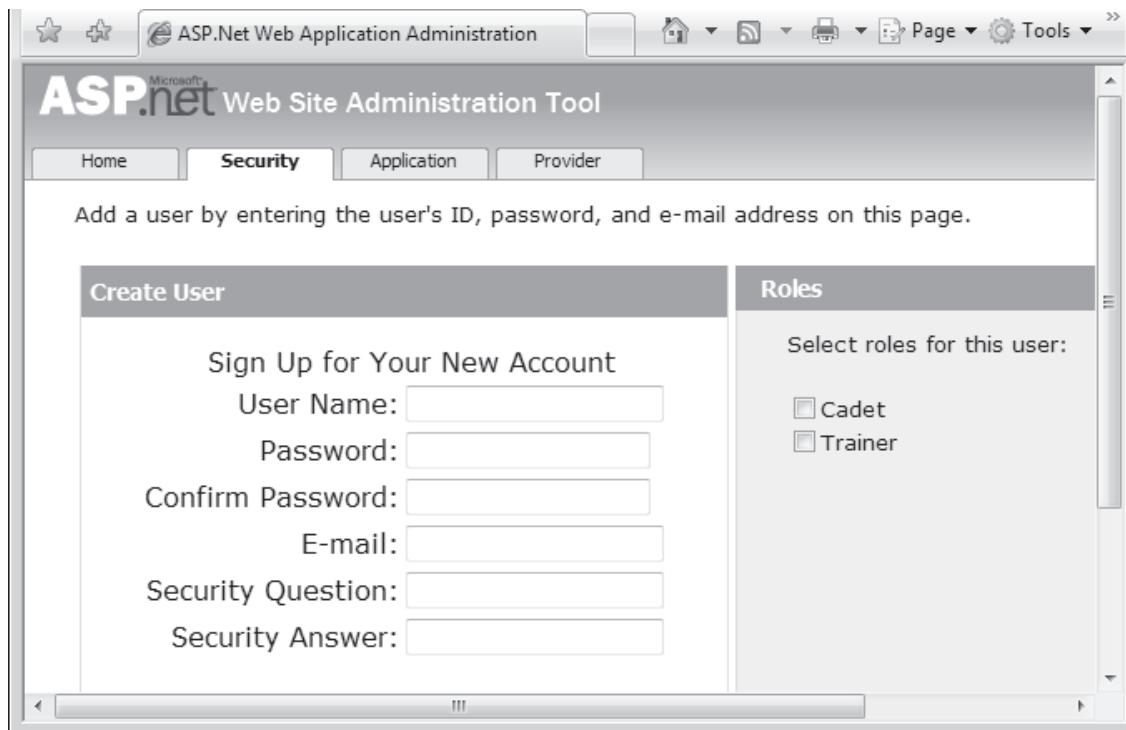


Figure 17.2: Web Site Administration Tool

17.6 ASP.NET Authentication Methods - Comparison

Each of the three authentication methods that ASP.NET supports (Windows-based, Forms-based, and Microsoft Passport) are best suited to specific situations. Each method has significant advantages and disadvantages.

Table 17.1 shows the advantages and disadvantages of Windows-based and Forms-based authentication.

Mode	Advantages	Disadvantages
Windows-based Authentication	The existing Windows infrastructure is used. Controls access to sensitive information.	Not suitable for most Internet applications.
Forms-based Authentication	Best-suited for Internet applications. Supports all client types.	Based on cookies.

Table 17.1: Windows-based and Forms-based Authentication - Advantages and Disadvantages

→ Windows-Based Authentication

Windows-based authentication uses the existing Windows infrastructure. Therefore, it is best suited to situations in which you have a fixed number of users with existing Windows user accounts. Two

example situations are as follows:

- **Case 1:** Developing an intranet for your organization. For example, your organization may already have Windows user accounts configured for each employee.
- **Case 2:** Controlling access to sensitive information. For example, you may want users in the Human Resources group to have access to directories that contain employee resumes and salary details. You can use Windows-based authentication to prevent employees in other Windows groups such as the Developers group from accessing these sensitive documents.

The disadvantage of Windows-based authentication is that it is not suitable for most Internet applications. For example, if you build a public user registration and password system, Windows-based authentication is not a good authentication option. With Windows-based authentication, a valid Windows user account must be configured for each user who accesses a restricted page. You cannot easily automate the process of adding new user accounts.

→ **Forms-Based Authentication**

Forms-based authentication is an appropriate solution if you want to set up a custom user registration system for your Web site. The advantage of this type of authentication is that it enables you to store user names and passwords in whatever storage mechanism that you want. For example, you can store credentials in the `web.config` file, an XML file, or a database table.

Forms-based authentication relies on cookies to determine the identity of the user. After Forms-based authentication is enabled, the user cannot access the requested page unless a specific cookie is found on the client. If this cookie is not found, or if the cookie is invalid, ASP.NET rejects the request and returns to the logon page.

→ **Microsoft Passport Authentication**

Microsoft Passport authentication includes several advantages:

- You can use the same user name and password to sign in to many Web sites; users are therefore, less likely to forget their passwords. For example, both Microsoft Hotmail and Microsoft MSN use Microsoft Passport to authenticate users.
- You do not have to set up and maintain a database to store user registration information. Microsoft performs all of this maintenance for you.
- You can customize the appearance of the registration and sign-in pages by supplying templates.

There are two disadvantages of Microsoft Passport authentication. First, there is a subscription fee to use the Microsoft Passport service. Second, Microsoft Passport authentication is based on cookies.

17.7 Secure Sockets Layer

When you develop Web applications, certain parts of the application require extra security. For example, Web pages that send confidential data, such as login credentials or financial transaction details, require strong security. You can use Secure Sockets Layer (SSL) to add security for such pages.

SSL provides the following features:

- SSL is supported by most Web servers and browsers
- Only trusted digital certificates are needed to protect Web applications through SSL
- In client-server operations, the SSL protocol uses a third party, a Certificate Authority (CA), to identify one end, or both ends of the communication

SSL encrypts data transmission and incorporates a mechanism to detect any change in data transmission. This helps prevent eavesdropping or tampering with sensitive data during transmission.

17.7.1 SSL with Client Browser and Server

SSL uses a public key and a private key to encrypt data transmission between a client and a Web server. The public key is known to everyone, and the private key is known only to the recipient of the message. A typical communication process between a client and Web server is shown in figure 17.3.

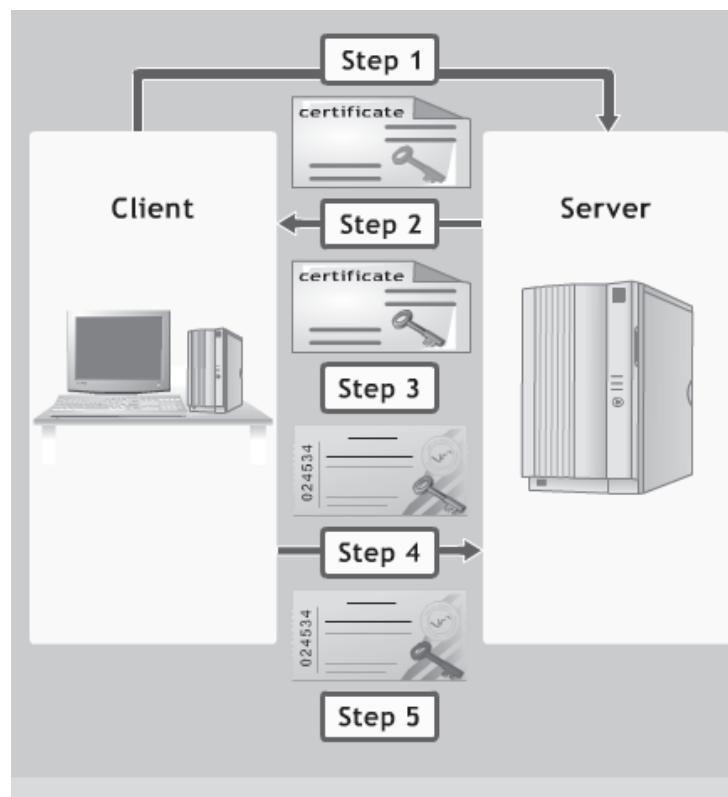


Figure 17.3: Communication Process

Each step in the communication process is explained as follows:

→ **Step 1**

The client browser contacts the Web server.

→ **Step 2**

The server sends back its certificate, encrypted with a trusted third-party private key.

→ **Step 3**

The browser decrypts the certificate with a trusted third-party public key.

→ **Step 4**

The browser uses the trusted third-party public key to encrypt a session token. The token is sent back to the server.

→ **Step 5**

The Web server receives the request and decrypts the session ticket with its private key. The server and the browser use the same session ticket for further encryption in transmission.

17.7.2 Configuring SSL in ASP.NET Pages

After configuring the server to use SSL, any page can be requested from the Web site by using a secure connection. SSL uses Hypertext Transfer Protocol Secure (HTTPS) to retrieve a Web page. For example, the secured page can be accessed by the address with the format <https://www.mysite.com/login.aspx>.

Note - The `Request.IsSecureConnection` property can be used to check whether you are on a secure HTTPS connection.

17.7.3 Obtaining an SSL Certificate

Consider a Web application in which you want to implement SSL for a login page. To use SSL, you need to obtain a certificate. To get an SSL certificate, a Certificate Signing Request (CSR) has to be submitted. CSR is a data file that holds details of the requesting party to a CA.

You can create a CSR using the Certificate wizard in IIS. The cert.txt file that is created needs to be submitted to a certificate-issuing authority. You save the certificate that is issued. Then, using the Certificate wizard in IIS, you process the pending request and install the certificate on the server.

After you install the SSL certificate, a user requesting the Home page is redirected to the SSL-secured login page. A padlock icon appears in the lower-right corner of the status bar to indicate the use of SSL. You can view the certificate by clicking the padlock icon.

The following code snippet demonstrates how to redirect users from the Home page to a login page that uses SSL.

Code Snippet:

```
<script runat="server">
protected void Page_Load(object sender, EventArgs e)
```

```
{
    stringurl = "https://localhost/SSLexample/login.aspx";
    Response.Redirect(url);
}

</script>
```

In the code, the user is redirected from Home.aspx to login.aspx by the `Response` object's `Redirect` method. The application has to explicitly switch to SSL when it is redirecting to an SSL-secured resource. This is done using an absolute URL such as `https://server/application/page.aspx` because relative URLs such as `~/page.aspx` will not work.

Figure 17.4 shows the content of the `cert.txt` file.

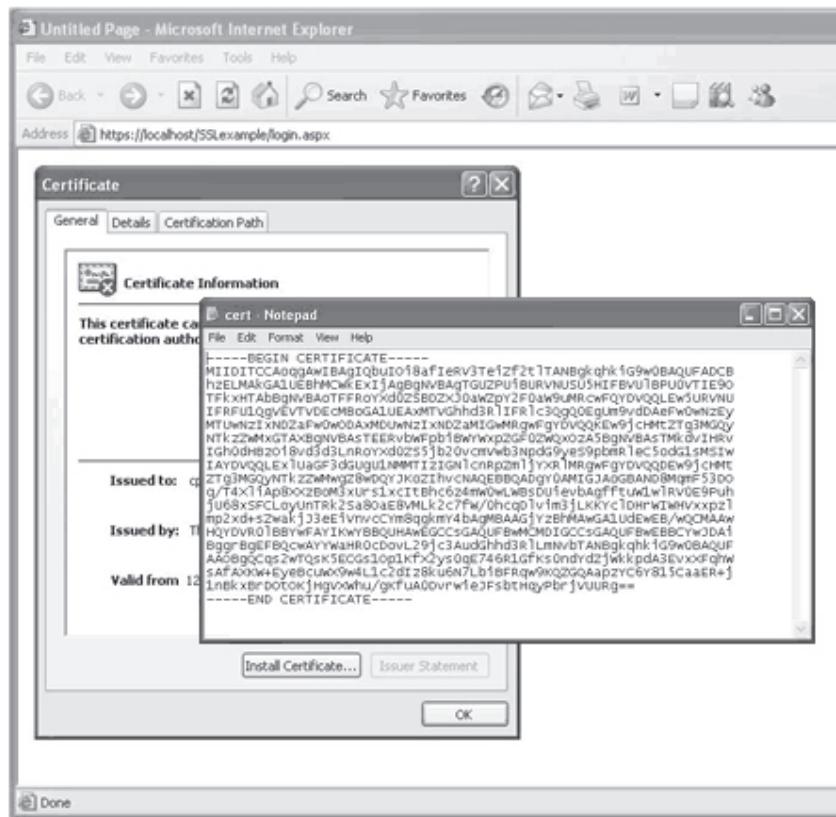


Figure 17.4: Client Certificate Details

17.8 Windows-Based Authentication

You can use Windows-based authentication to secure Web applications when you know which users access your Web site.

You can secure Web applications using Windows-based authentication using a four-step process:

1. Configure IIS.

2. Set up authentication in **web.config**.
3. Set up authorization in **web.config**.
4. Request of logon information from the users by IIS.

The details of each step are as follows:

→ **Configure IIS**

In Windows-based authentication, for securing Web application, you must configure IIS to use one or more authentication mechanisms:

- Basic Authentication
- Digest Authentication
- Windows Integrated Security

→ **Set up authentication in web.config**

The second step is to set ASP.NET security to Windows-based authentication in `web.config`. The `<authentication>`, `<authorization>`, and `<identity>` sections in **web.config** can be used for the security settings.

The following code snippet sets the authentication method for the application to Windows by using the `<authentication>` subsection in `web.config` file.

Code Snippet:

```
<system.web>
  <authentication mode="Windows" />
</system.web>
```

→ **Set up authorization in web.config**

You can secure specific pages in a Web application by using the `<location>` section in the `<configuration>` section with `<system.web>` and `<authorization>` subsections.

The following code snippet demonstrates securing a page named **LibraryRegister.aspx** by denying access to all anonymous users.

Code Snippet:

```
<location path="LibraryRegister.aspx">
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>
```

```
</system.web>
</location>
```

Note - A Web Form or a folder can be specified in the <location> section. If you specify a folder name, all of the subfolders under it are secure. You can secure multiple Web Forms or folders by using multiple <location> sections.

The following code snippet secures a Web application by creating an <authorization> section in the <system.web> section.

Code Snippet:

```
<system.web>
  <authorization>
    <deny users="?" />
  </authorization>
</system.web>
```

→ Request of logon information from the users by IIS

The last step of enabling Windows-based authentication is when users attempt to access a Web Form from your Web application and IIS requests logon information from the user. The user must provide his or her user name and password. If the user's credentials are approved, IIS grants the user access to the secure Web page.

17.8.1 User Information

After completion the Windows-based authentication, the Web server can read the user identity from any Web page of the Web application. The User.Identity.Name is used to read the identity of the user. The Web server can also use User.Identity.AuthenticationType to identify the IIS authentication mechanism that is used to authenticate the user. Additionally, Web server can test if the user is authenticated by using User.Identity.IsAuthenticated.

The following code snippet shows the code that allows the Web server to read the user identity:

Code Snippet:

```
userIdentity.Text = User.Identity.Name;
userTypeIdentity.Text = User.Identity.AuthenticationType;
userAuthenticatedIdentity.Text = User.Identity.IsAuthenticated;
```

17.9 Forms-Based Authentication

The most common method to secure a Web application is Forms-based authentication.

Figure 17.5 shows the sequence of Forms-based authentication.

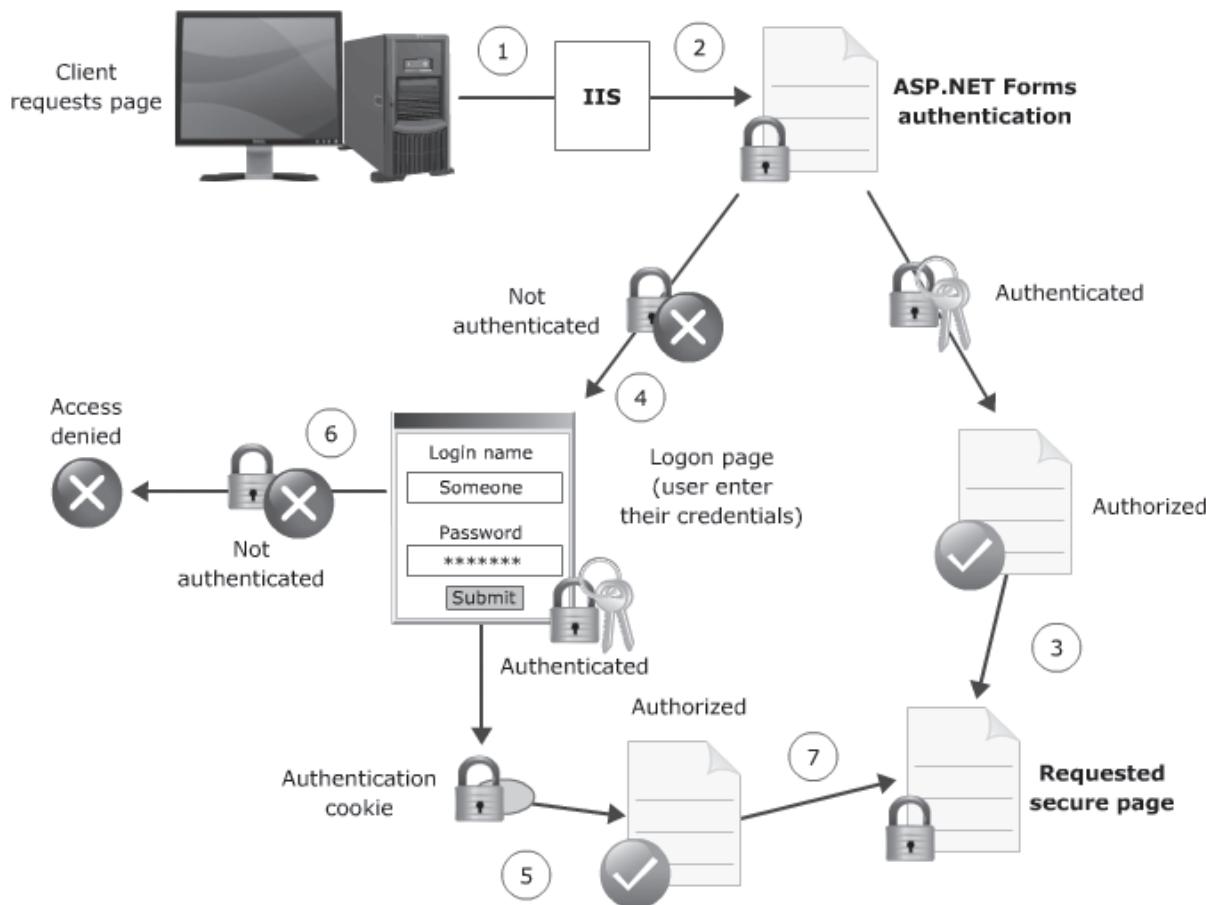


Figure 17.5: Forms-based Authentication

The Forms-based authentication provides a customized means of authentication without having to use cookies to manage sessions. When a user requests restricted resources in a Web application, user authentication is first performed by IIS. If anonymous access is enabled in IIS or on successful authentication, the request is forwarded to the ASP.NET application. ASP.NET examines the request for a valid authentication cookie and then, performs the authorization check. If the user clears the authorization check, access to the resources is granted. Otherwise, access is denied.

If the user request is without an authentication cookie, ASP.NET redirects the user to the login page. On the login page, the user credentials are resubmitted for authentication by the application code. On authentication, ASP.NET attaches a cookie and redirects the user to the requested resources. The same cookie is then used to allow the user to revisit restricted resources during the session.

17.9.1 Enabling Forms-Based Authentication

The following four steps are required to enable Forms-based authentication:

1. Configure IIS to use **Anonymous** authentication.
2. Configure authentication in **web.config**.

3. Configure authorization in **web.config**.
4. Create the login page.

The details of each step are as follows:

→ **Configure IIS to use Anonymous authentication**

The first step for Forms-based authentication is to configure IIS to use anonymous authentication so that the user is authenticated by ASP.NET and not by IIS.

Configure authentication in `web.config`

The second step is to set the authentication method to Forms-based for the application in `web.config` file.

The following code snippet demonstrates the Forms-based authentication in `web.config` file by using the `<authentication>` sub-section of `<system.web>`.

Code Snippet:

```
<system.web>
  <authentication mode="Forms">
    <forms name=".ASPxAUTH" loginUrl="login.aspx" />
  </authentication>
</system.web>
```

In the code snippet, the `name` attribute specifies the HTTP cookie to use for authentication. The default value is `.ASPxAUTH`. The `loginUrl` indicates the URL to redirect the user to if a valid authentication cookie is not found.

If the authentication mode is Forms, the `<forms>` element must be added to the `<authentication>` section.

The settings of the cookie can be configured in the `<forms>` section. You can set the `name` attribute to the suffix to be used for the cookies and the `loginUrl` attribute to the URL of the page to which unauthenticated requests are redirected.

→ **Configure authorization in `web.config`**

The next step is to set the `<authorization>` section in `web.config`. In this section you can allow or deny access to users in the Web application.

→ **Create the login page**

The final step is to create a logon Web Form. The page can be created by using the ASP.NET login controls. The user has to enter the user name and password in the logon page to establish authentication and to access the Web application.

17.9.2 Creating a Logon Page

Whenever a user visits a Web portal with facilities such as online shopping or money transactions, security of the account or data from other users is one of the most important requirements. For example, if a user has an account with Paypal.com, the data of the user needs to be secured from other users, who may use the same account. To enable this kind of functionality, there is a need to authenticate the user before he or she is allowed to access their online account.

To address this issue, ASP.NET provides a set of server controls that offer a complete login solution for Web applications. These controls provide users with an option to type and validate their login credentials. You can drag and drop the relevant login controls from the Toolbox and then, customize the properties of the added controls.

You can use the login controls in ASP.NET to authenticate a user. These controls do not require any additional programming. Table 17.2 lists the ASP.NET login controls.

Control	Description
Login	Provides all pre-built user interface elements that are required for user authentication.
LoginView	Customizes the information displayed to anonymous and logged-in users for a Web site.
LoginStatus	Provides a login link for the users who are not authenticated and a logout link for authenticated users for a Web site.
LoginName	Displays the name of authenticated users of a Web site who are logged on.
PasswordRecovery	Enables a user to recover a forgotten password. The password will be sent to the e-mail address that was used when the account was created.
CreateUserWizard	Creates a new user account and adds it to the ASP.NET membership system.
ChangePassword	Enables users to change their passwords.

Table 17.2: ASP.NET Login Controls

17.10 IIS Authentication Mechanism

IIS needs to be configured before you can use Windows-based authentication. When the user requests a page that requires authorization, the user is authenticated by IIS.

IIS uses several mechanisms that you can use to establish authentication. The four options available in IIS are as follows:

→ Anonymous Access

This mechanism allows any user to access the ASP .NET application. When a request from an anonymous user is received, IIS in turn makes the request to Windows by using the default `IUSR_machinename` account.

→ Basic Authentication

This authentication requires the use of a Windows user name and a password to connect to the application. However, the password is transmitted in plain text, making this type of authentication insecure and it is very easy to crack. The data is passed by using Base64 encoded format. Figure 17.6 shows the data transmission.

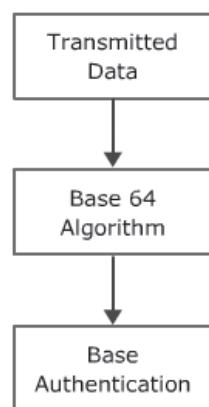


Figure 17.6: Data Transmission

Figure 17.7 shows the authentication methods with the Basic Authentication enabled in Internet Information Services (IIS) Manager.

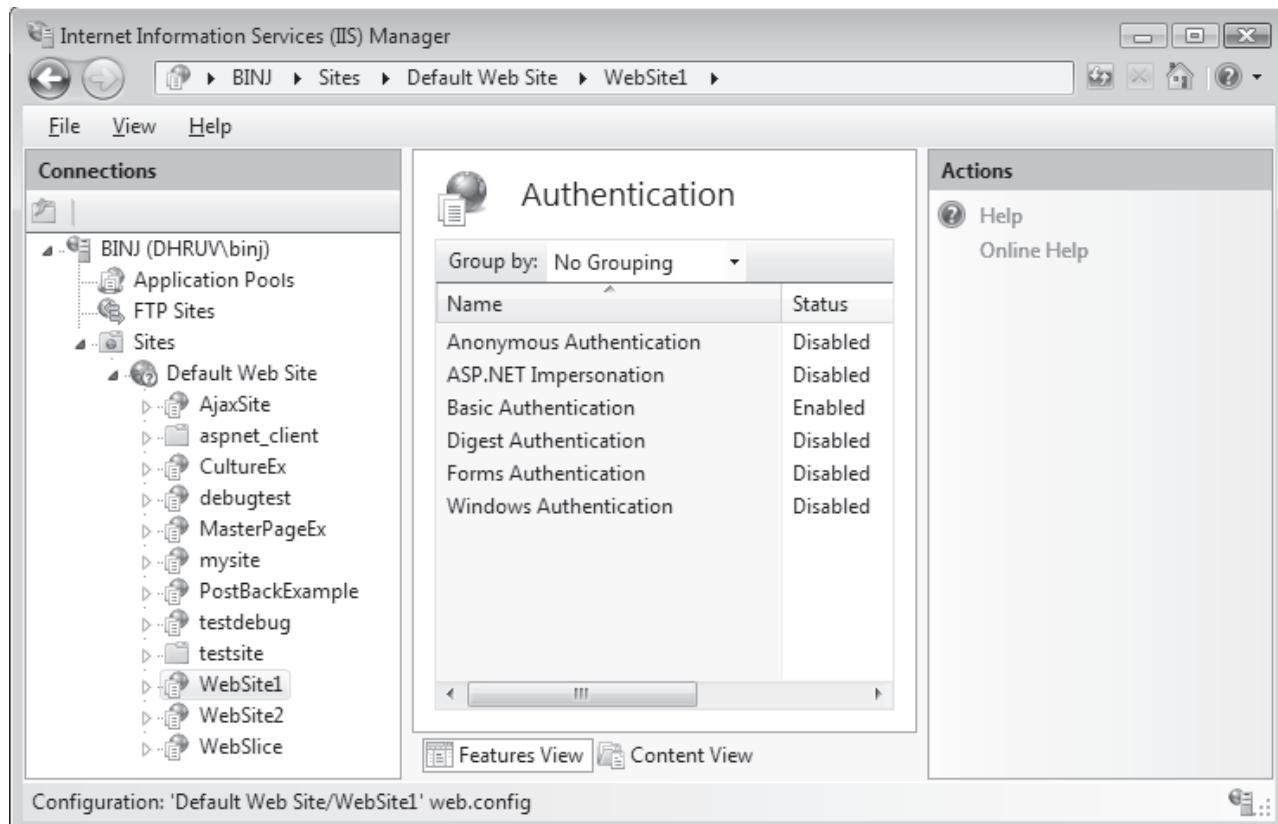


Figure 17.7: Authentication Methods in Internet Information Services (IIS) Manager

→ Digest Authentication

This authentication is similar to Basic Authentication. However, the user information is encrypted and transmitted to the server. If Anonymous access is disabled, users are prompted for their credentials (logon information). The browser combines this logon information with the other information that is stored on the client and then, sends an encoded hash called an MDS hash (also known as Message Digest) to the server. The server already has a copy of this information; it recreates the original details from its own hash and authenticates the user. This mechanism works only with Microsoft Internet Explorer 5 or more recent browsers, but it does pass through firewalls and proxy servers and also over the Internet. Figure 17.8 shows the data transmission in digest authentication.

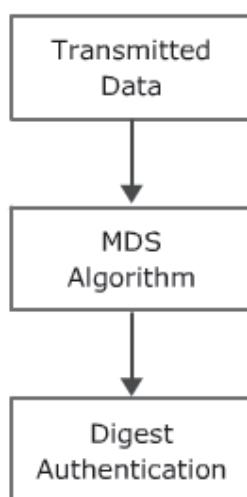


Figure 17.8: Data Transmission

→ Integrated Windows Security

The Windows logon credentials are used here to authenticate users. In a Windows-based network, if the user has already been authenticated, IIS can pass on the user's credentials when they request access to a resource. The user name and password are not included in the credentials, only an encrypted token that indicates the user's security status.

However, Integrated Windows security is not practical in Web applications that confront firewalls. Therefore, it is best suited to a corporate intranet scenario.

17.11 Check Your Progress

1. Which of the following statements about the security features in ASP.NET are true?
 - a. For Forms-based authentication, you must provide the code to verify user credentials.
 - b. Impersonation does not work with anonymous user access.
 - c. Authentication is required before authorization.
 - d. File authorization makes use of authorization rules from the web.config file.
 - e. Impersonation requires the use of IIS to authenticate users.

(A)	a, b, c	(C)	a, b, d
(B)	a, c, e	(D)	b, c, d

2. Which of the following statements about Forms-based authentication for an ASP.NET application are true?
 - a. Forms-based authentication requires the use of a Web page for user authentication.
 - b. ASP.NET provides an authenticated cookie for a valid user.
 - c. IIS performs the authorization check for users.
 - d. User credentials can be stored in a database, an XML file, or the web.config file.
 - e. The `<authorization mode>` attribute in the web.config file is set to Forms.

(A)	b, c, d	(C)	a, c, d
(B)	a, b ,d	(D)	a, b ,c

3. Which of the following options refer to securing Web sites?
 - a. Restrict specific domain names
 - b. Authorize only authenticated users
 - c. SSL encrypts trusted certificates
 - d. ASP.NET encrypts data transmission
 - e. SSL prevents data tampering
 - f. SSL protocol uses CA
 - g. Install certificates using IIS

(A)	a, b, c	(C)	c, f, g
(B)	b, d, e	(D)	a, b, e, g

4. Match the following controls with their corresponding description?

	Control		Description
(a)	LoginView	(1)	Provides a login link for the users who are not authenticated and a logout link for authenticated users for a Web site
(b)	LoginName	(2)	Provides all pre-built user interface elements that are required for user authentication
(c)	LoginStatus	(3)	Creates a new user account and adds it to the ASP.NET membership system
(d)	Login	(4)	Displays the name of authenticated users of a Web site who are logged on
(e)	CreateUserWizard	(5)	Customizes the information displayed to anonymous and logged-in users for a Web site

(A)	a-2, b-3, c-4, d-5, e-1	(C)	a-5, b-4, c-1, d-2, e-3
(B)	a-4, b-5, c1, d-2, e-3	(D)	a-3, b-4, c-5, d-1, e-2

5. Which one of the following codes allows access to the Blake identity and deny access to all other users?

(A)	<pre><authorization> <allow users="Blake"/> <deny users="*"/> </authorization></pre>	(C)	<pre><authorization> <allow users="Blake"/> <deny users="all"/> </authorization></pre>
(B)	<pre><authorization> <accept users="Blake"/> <reject users="?"/> </authorization></pre>	(D)	<pre><authorization> <allow = "Blake"/> <deny="?"/> </authorization></pre>

Module Summary

In this module, **Authentication and Authorization**, you learnt about:

→ **Authentication**

Authorization, authentication, and impersonation are the security mechanisms in ASP.NET. Authentication is used to verify the identity of a user before allowing or denying a request. Authorization enables only users with a valid identity to access specific resources in an application. Authentication providers help you provide Windows-based, Forms-based, or Microsoft Passport authentication.

→ **Authorization**

SSL secured pages help you protect parts of your Web site that process confidential information. IIS uses mechanisms such as anonymous access, basic authentication, digest authentication, and integrated Windows security to establish authentication.

Module - 18

Authentication and Authorization (Lab)

Welcome to the module, **Authentication and Authorization (Lab)**.

In this module, you will learn to:

- Create an application that uses Web application security
- Use Forms-based and Windows-based authentication in ASP.NET

Web Development

http://www



18.1 Part I – 60 Minutes

Exercise 1

Zion Motors is a multinational automaker based in New York, USA. The company was founded in 1980 and has now become the world's fourth largest automaker. With its global headquarters in New York, Zion employs 50,000 people in every major region of the world and does business in around 97 countries. Zion Motors manufactures buses and trucks in 19 countries.

The management of the company wants to develop a Web-based application to maintain their Sales and Technician details. The application is intended to be available for each employee of Sales and Technician departments. However, some information is highly secure and must be made accessible only to the director and some other senior staff.

Consider that you are one of the programmers in the development team and are assigned the following tasks:

- Create roles
- Create users and assign them roles
- Create a Login form to authenticate users of the application
- Create access rules for users belonging to certain roles
- You must use Visual Studio 2008 and ASP.NET 3.5 to create the application.

Solution:

The solution is to develop a Web-based application that enables Forms-based authentication. You can create roles for Sales and Technician departments and specify access to only those roles.

Setting Forms-based Authentication

1. Create a Web site named **ZionMotors**.
2. Change the **id** of the form to '**frmHome**' and **title** to **Zion Motors**.
3. Add the heading of the page as **ZION Motors** and apply the **blue** color to it.
4. Copy the image file, **bus.jpg**, into the application.
5. To display the image on the page, create a new style named **body** in the **Source** section as follows:

```
body
{
    background-image: url('bus.jpg');
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-position: top center;
}
```

Now, the Web Page will look as shown in figure 18.1.

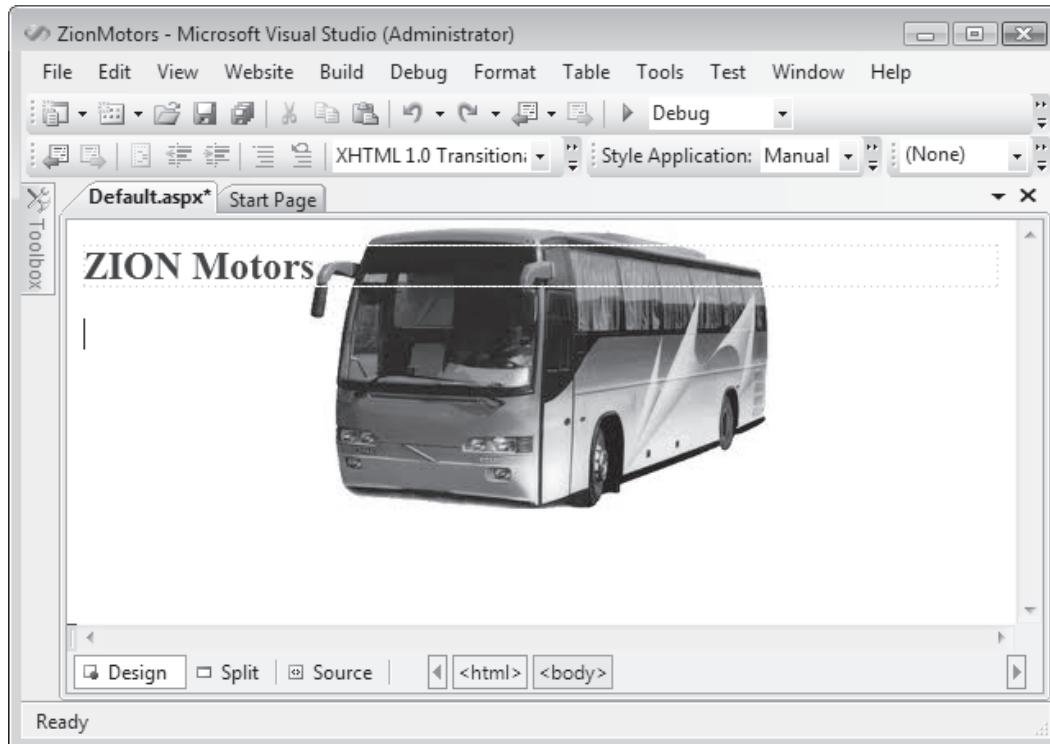


Figure 18.1: Design View of Home Page

6. To open the Microsoft **ASP.NET Web Site Administration Tool**, on the **Website** menu, click **ASP.NET Configuration**.

Figure 18.2 shows the ASP.NET Web Site Administration Tool.

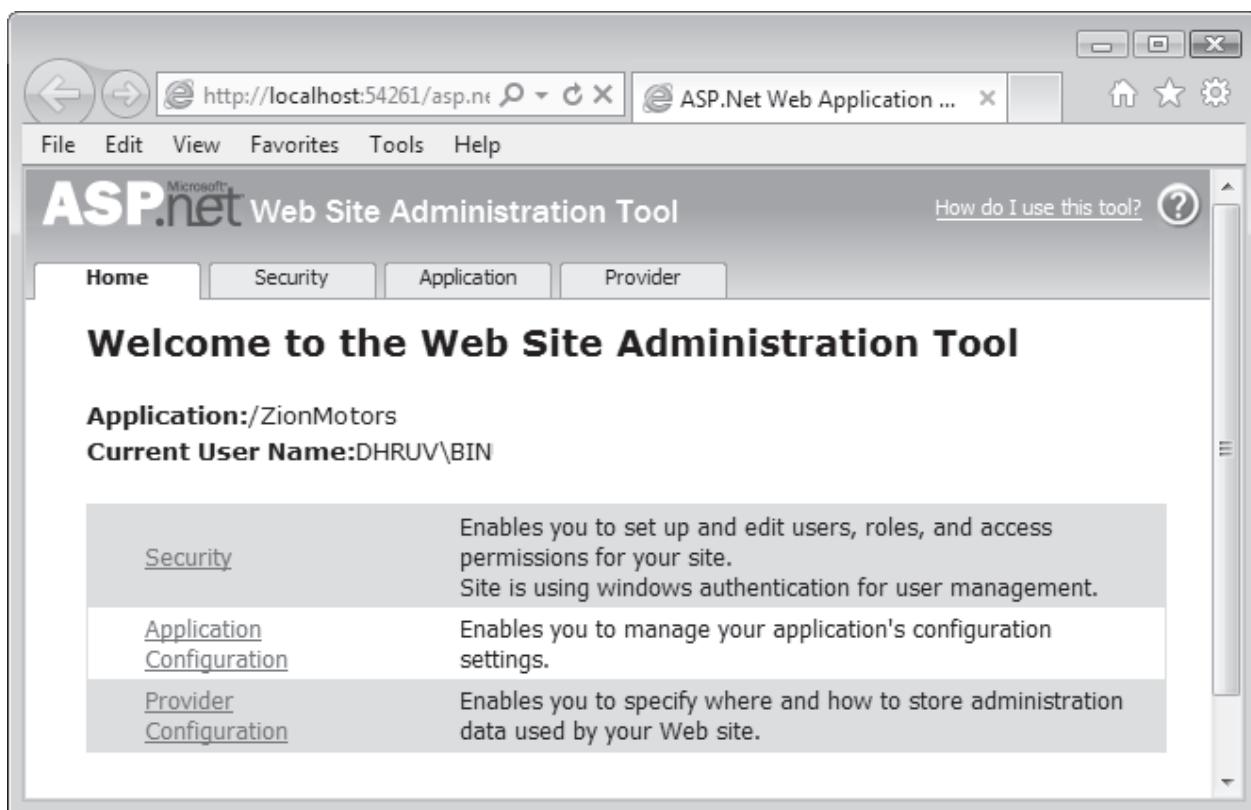


Figure 18.2: ASP.NET Web Site Administration Tool

You can use the **ASP.NET Web Site Administration Tool** to configure the security settings for the **ZionMotors** application.

7. To open the **Security** page, on the **ASP.NET Web Site Administration Tool**, click **Security**.

Figure 18.3 displays the Security page.

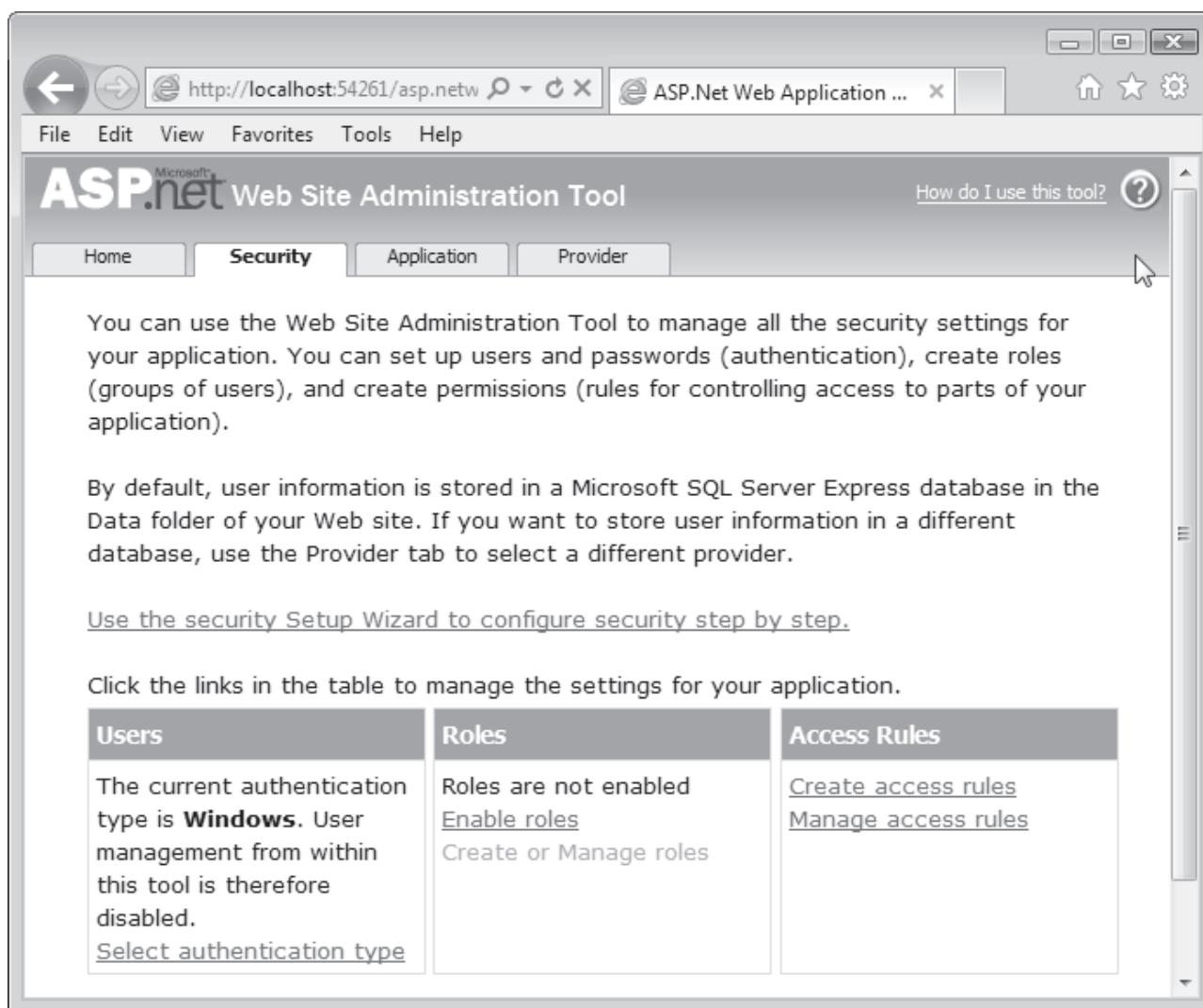


Figure 18.3: Security Page

You can define the settings for the authentication process in this page.

8. To define the authentication type and corresponding values, in the **Users** section on the **Security** page, click **Select authentication type**.
- The '**How will users access your site?**' section of the security page is displayed. In this section you can choose the authentication type as Forms-based or Windows-based.

9. To enable Forms-based validation, on the ‘How will users access your site?’ section, click **From the Internet** as shown in figure 18.4.

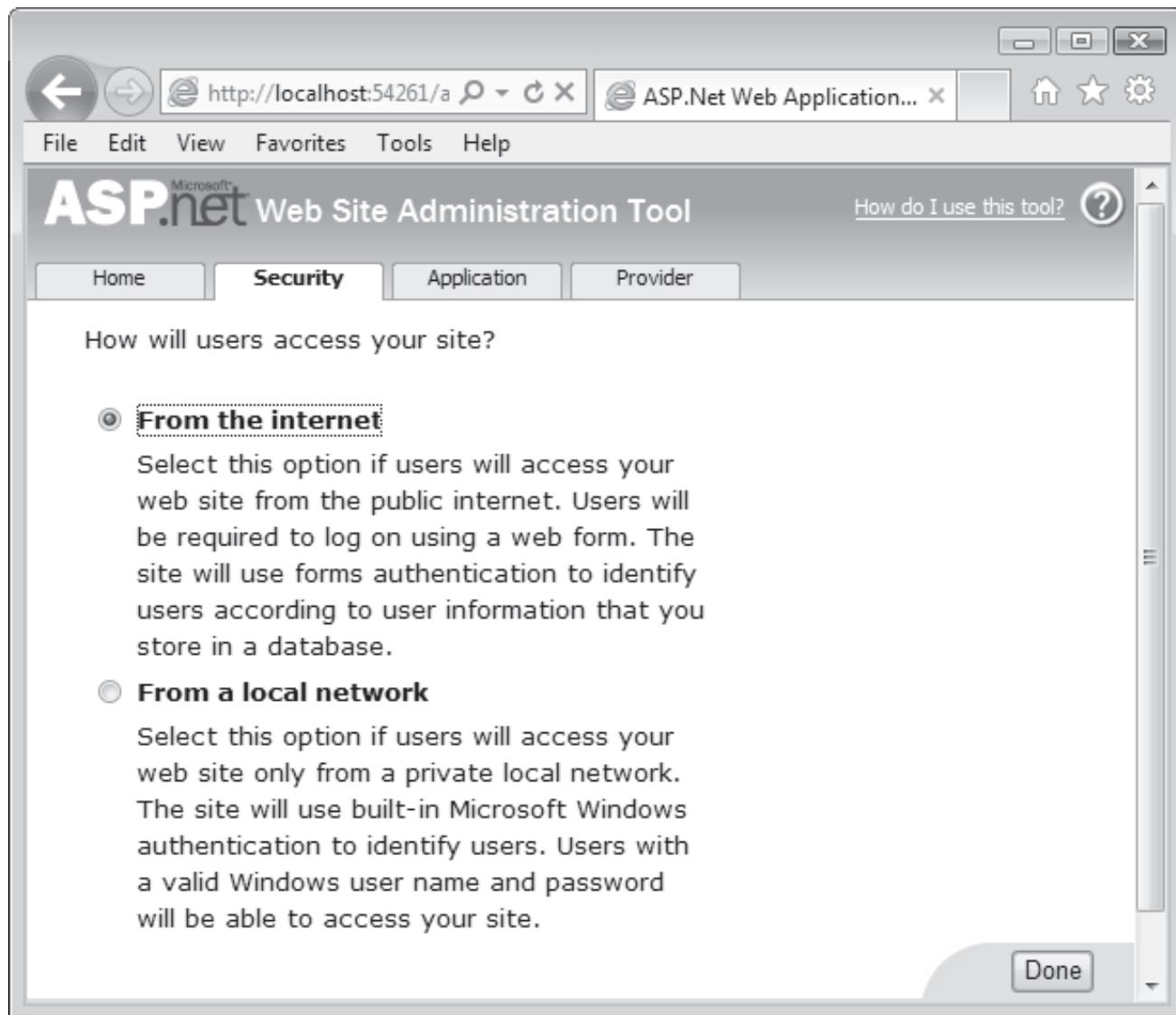


Figure 18.4: Selecting Forms-based Authentication

Forms-based authentication validates users based on the login credentials information stored in the `web.config` file or in a database. The current application will use the `aspnetdb` database to store the login credentials.

10. To save the selected authentication type, on the **Security** page, click **Done**.

This process creates a new database named **ASPNETDB.MDF** in the application.

Also, the **web.config** file is updated with the authentication element as follows:

```
<authentication mode="Forms" />
```

11. To complete the configuration, in the **web.config** file, type the following code between the

<authentication> and </authentication> elements:

```
<forms name="ZionMotors" loginUrl="Login.aspx" timeout="30" path="/" slidingExpiration="true">
</forms>
```

The code defines the cookie name as ZionMotors and the login page is set to login.aspx. The timeout attribute specifies that the cookie will expire after 30 minutes. The cookie will be saved at the root because the path attribute is assigned the value '/'. Further, the cookie timeout will reset on the subsequent visit of a user because the slidingExpiration attribute is set to true.

12. To restrict anonymous or unauthenticated users from accessing the application, in the **web.config** file, after </authentication> element, type the following code:

```
<authorization>
  <deny users="?">
  </deny>
</authorization>
```

The <deny> child element is used to deny access to specified users. The '?' value for users indicates anonymous or unauthenticated users.

Configuring Access to Anonymous or Unauthenticated Users

In the section, you will create a login page in which anonymous or unauthenticated users will be redirected to logon to the Web application.

1. Create a new Web Form named **Login.aspx**.
2. Change the **title** to **Login Page** and **form id** to **frmLogin**.
3. Create the user interface of the Login page as shown in figure 18.5.

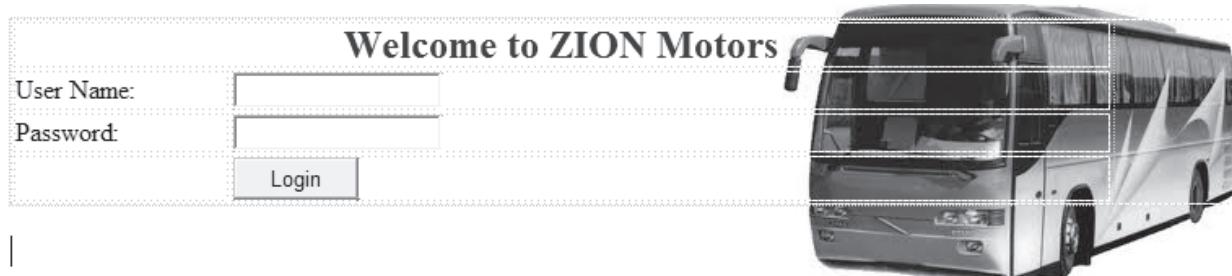


Figure 18.5: User Interface of Login.aspx

The code to create the user interface is shown as follows:

```
<table>
    <tr>
        <td colspan="2" style="text-align: center">
            <h2 class="style3">
                Welcome to ZION Motors
            </h2>
        </td>
    </tr>
    <tr>
        <td class="style2">
            User Name:
        </td>
        <td>
            <asp:TextBox ID="txtUserName" runat="server"></asp:TextBox>
        </td>
    </tr>
    <tr>
        <td>
            Password:
        </td>
        <td>
            <asp:TextBox ID="txtPassword" runat="server" TextMode="Password">
            </asp:TextBox>
        </td>
    </tr>
    <tr>
        <td>
            &nbsp;
        </td>
    </tr>
```

```
<td>
<asp:Button ID="btnLogin" runat="server" Text="Login" Width="77px" />
</td>
</tr>
</table>
```

4. Generate the Click event of the button, **btnLogin**, and add code as follows:

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    if (Membership.ValidateUser(txtUserName.Text, txtPassword.Text))
        FormsAuthentication.RedirectFromLoginPage(txtUserName.Text, false);
    else
        Response.Write("Invalid Credentials");
}
```

The `ValidateUser()` method of the `Membership` class is used to authenticate the user by verifying the login credentials provided by the user against the credentials stored in the membership system.

Enabling Role Management in the Application

In this section, you will enable role management for the ZionMotors application.

1. Open the **ASP.NET Web Site Administration Tool**.
2. Select the **Security** page.
3. To enable role management in the application, in the **Roles** section in the **Security** page, click **Enable Roles**.

Role management is enabled for the application as shown in figure 18.6.

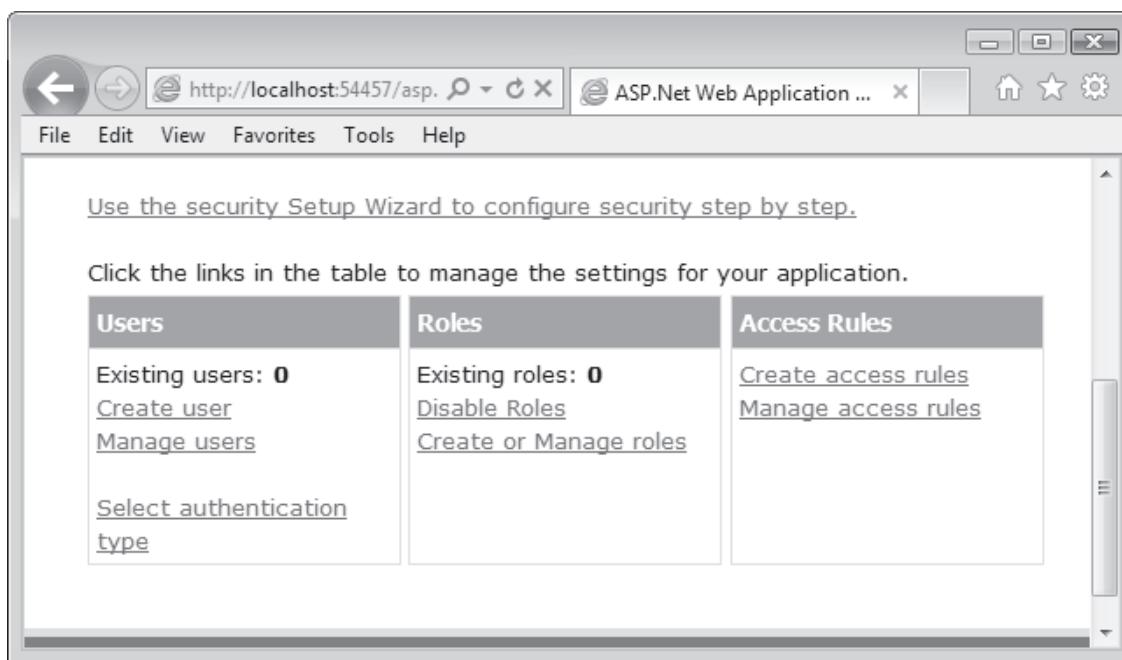


Figure 18.6: Enabling Role Management

4. Click **Yes** to save the changes.

The following code is automatically added in the **web.config** file when you enable role management.

```
<roleManager enabled="true" />
```

Creating roles for the application

The next step is to create roles for the application. You can perform the following steps to create the roles:

1. Open the **ASP.NET Web Site Administration Tool**.
2. Select the **Security** page.
3. To create roles in the application, in the **Roles** section on the **Second** page, click **Create or Manage roles**.

The page where new roles can be created is displayed as shown in figure 18.7.

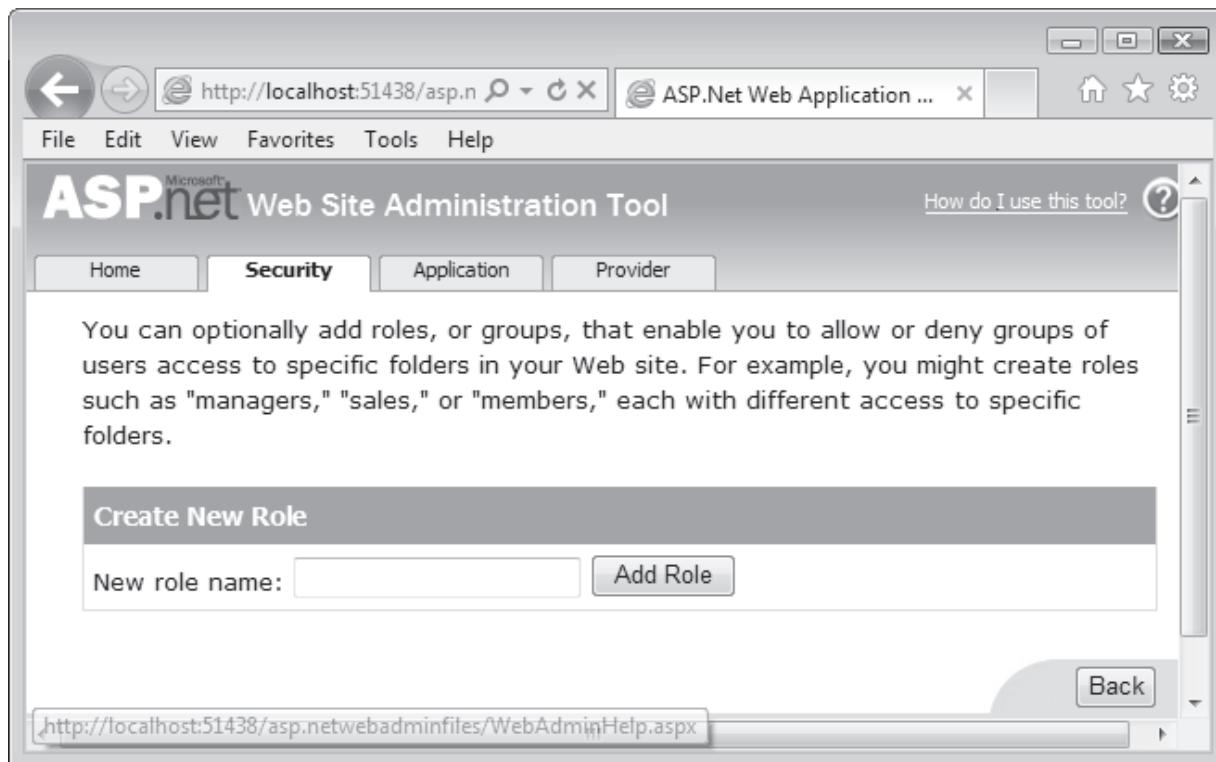


Figure 18.7: Adding Roles

4. To specify a new role name, in the **New role name** box, type **Sales** and click **Add Role**. The new role named **Sales** is added to the application.
5. To create another role, in the **New role name** box, type **Technician** and click **Add Role**. An additional role named **Technician** is added.

Creating Users and Assigning Roles

The next step is to create users for the ZionMotors application and assign appropriate roles to them.

1. Open the **ASP.NET Web Site Administration Tool**.
2. Select the **Security** page.
3. To display the page for creating users, in the **Users** section on the **Security** page, click **Create user**. The page where new users can be created is displayed.

4. To create a user named **davidblake** with password **sales _ 123**, type the following details as shown in figure 18.8.

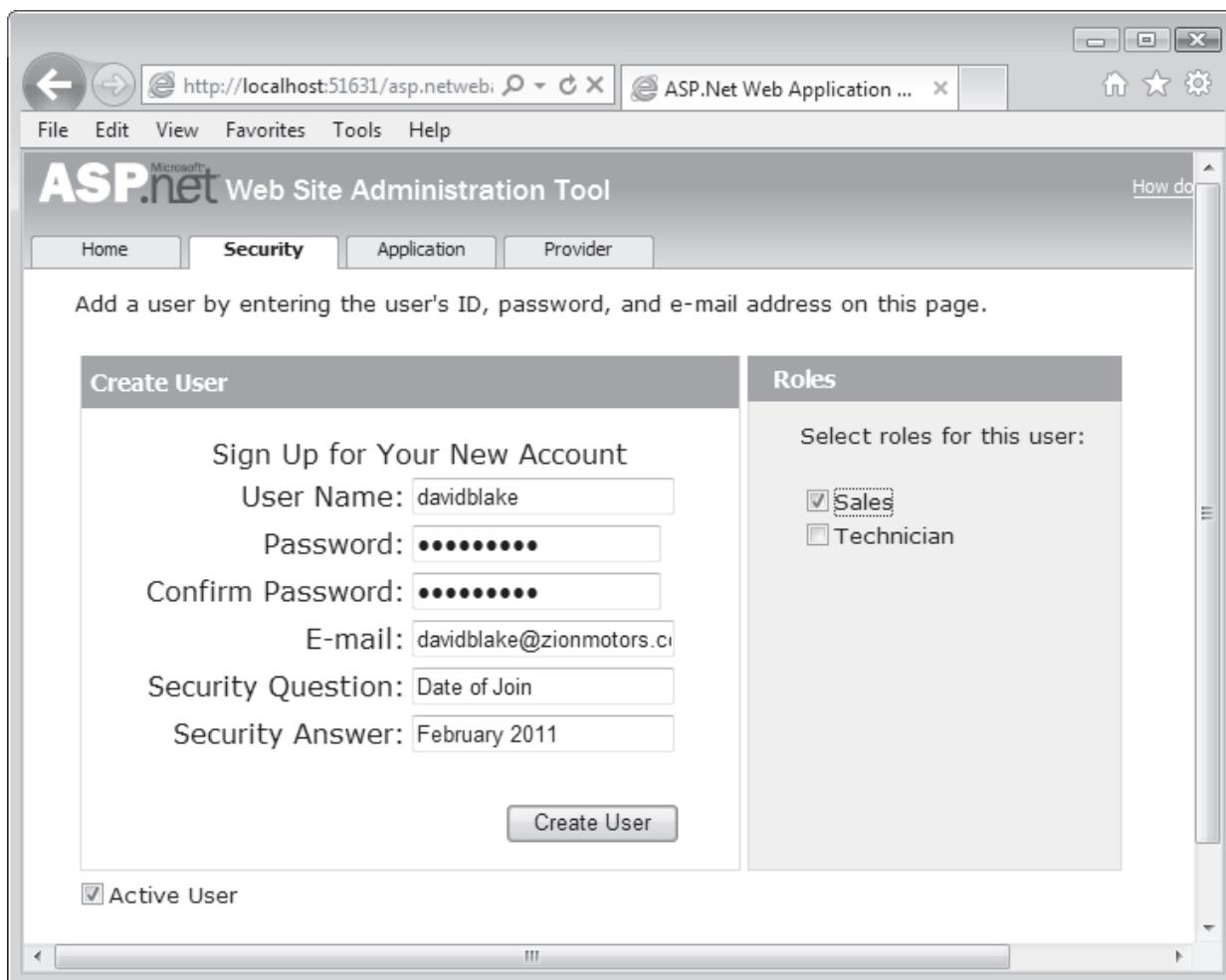


Figure 18.8: User Details

The user details are specified and the **Sales** role is assigned to the user.

5. To complete the user creation, in the **Create User** section, click **Create User**. To complete the user creation, in the **Create User** section, click **Create User**.
6. To complete the user creation, in the **Create User** section, click **Create User**.
- A message is displayed to indicate the successful creation of the user account. The new user, **davidblake**, is created with the role **Sales**.
7. Similarly, create one more user named **johnhonai** with **password, tech_123** and assign the role **Technician** to the user.

Creating Access Rules

The next step is to create access rules for the application.

1. Create two folders namely, **Sales** and **Technician**, with Web pages **Sales.aspx** and **Technician.aspx**

respectively in the application.

Figure 18.9 shows the **Sales.aspx** page and figure 18.10 shows the **Technician.aspx** page.

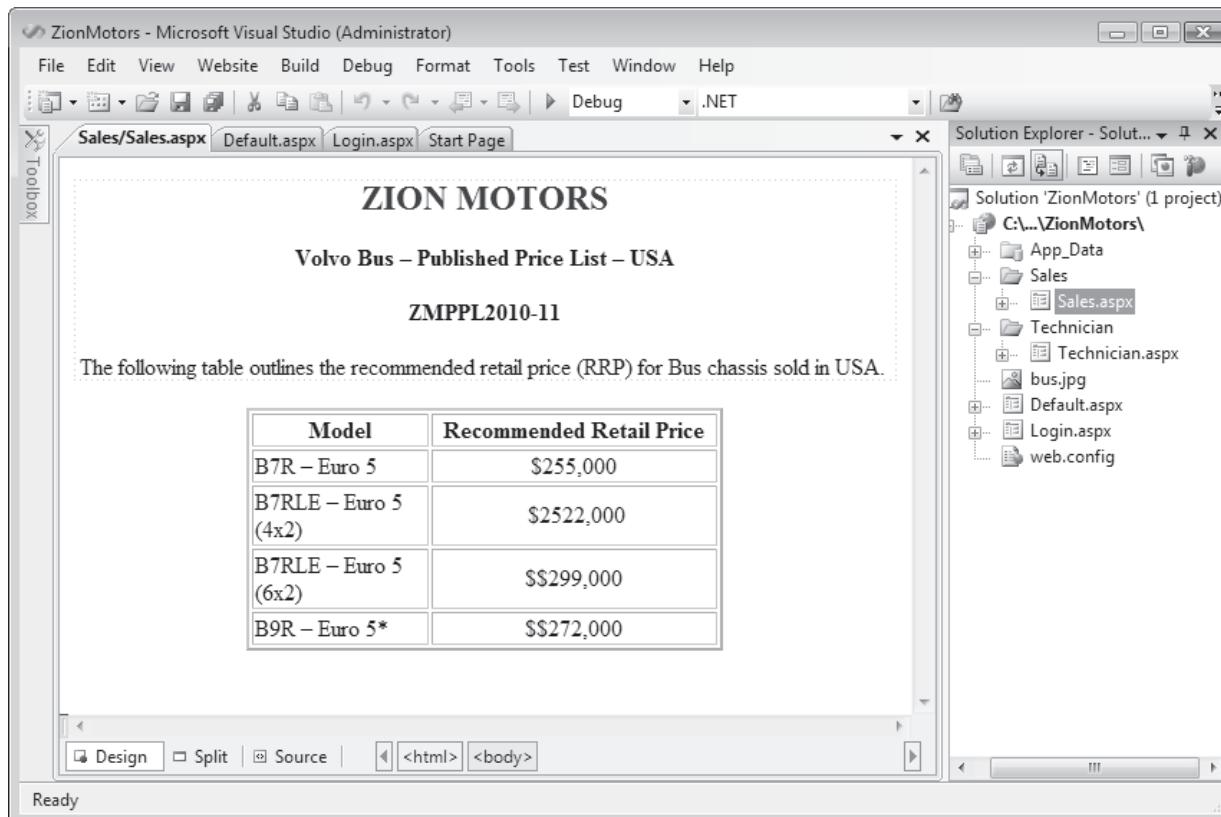


Figure 18.9: Sales.aspx Page

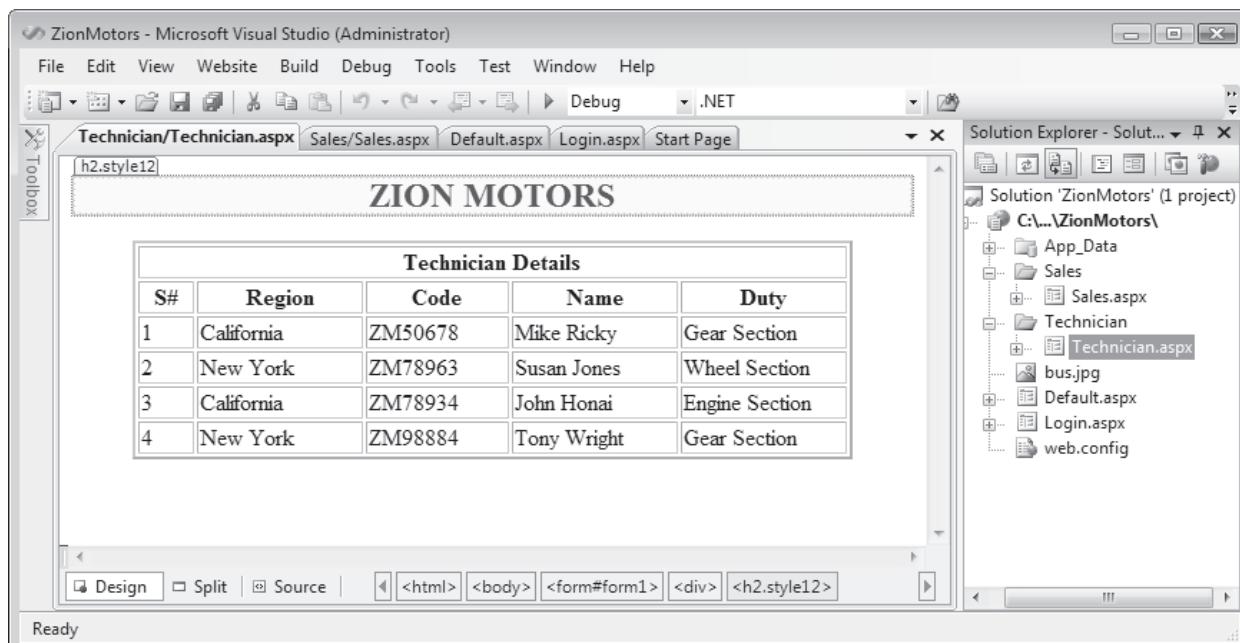


Figure 18.10: Technician.aspx Page

2. Add two **Hyperlink** controls with text **Sales Data** and **Technician Details** respectively to **Default.aspx** page. Name the controls as **InkSales** and **InkTechnician** respectively.
3. Set the **NavigateUrl** property of the **Sales Data** hyperlink control to **Sales.aspx** and **Technician Details** hyperlink control to **Technician.aspx** pages respectively. The Default.aspx page will now look as shown in figure 18.11.

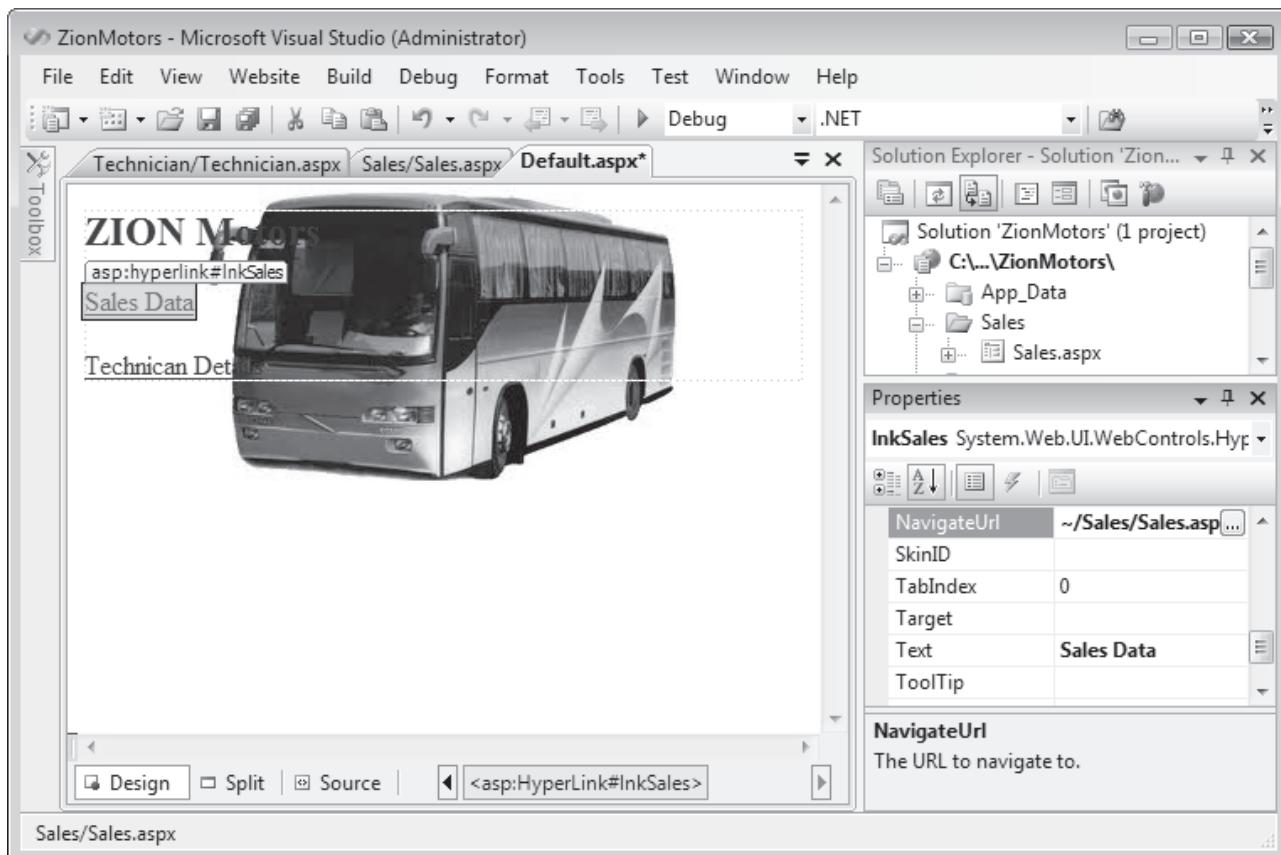


Figure 18.11: Default.aspx Page

4. Open the **ASP.NET Web Site Administration Tool**.
5. Select the **Security** page.

6. To display the access rule page for the application, in the **Access Rules** section, click **Create access rules**. The page that defines access rules for the application is displayed as shown in figure 18.12.

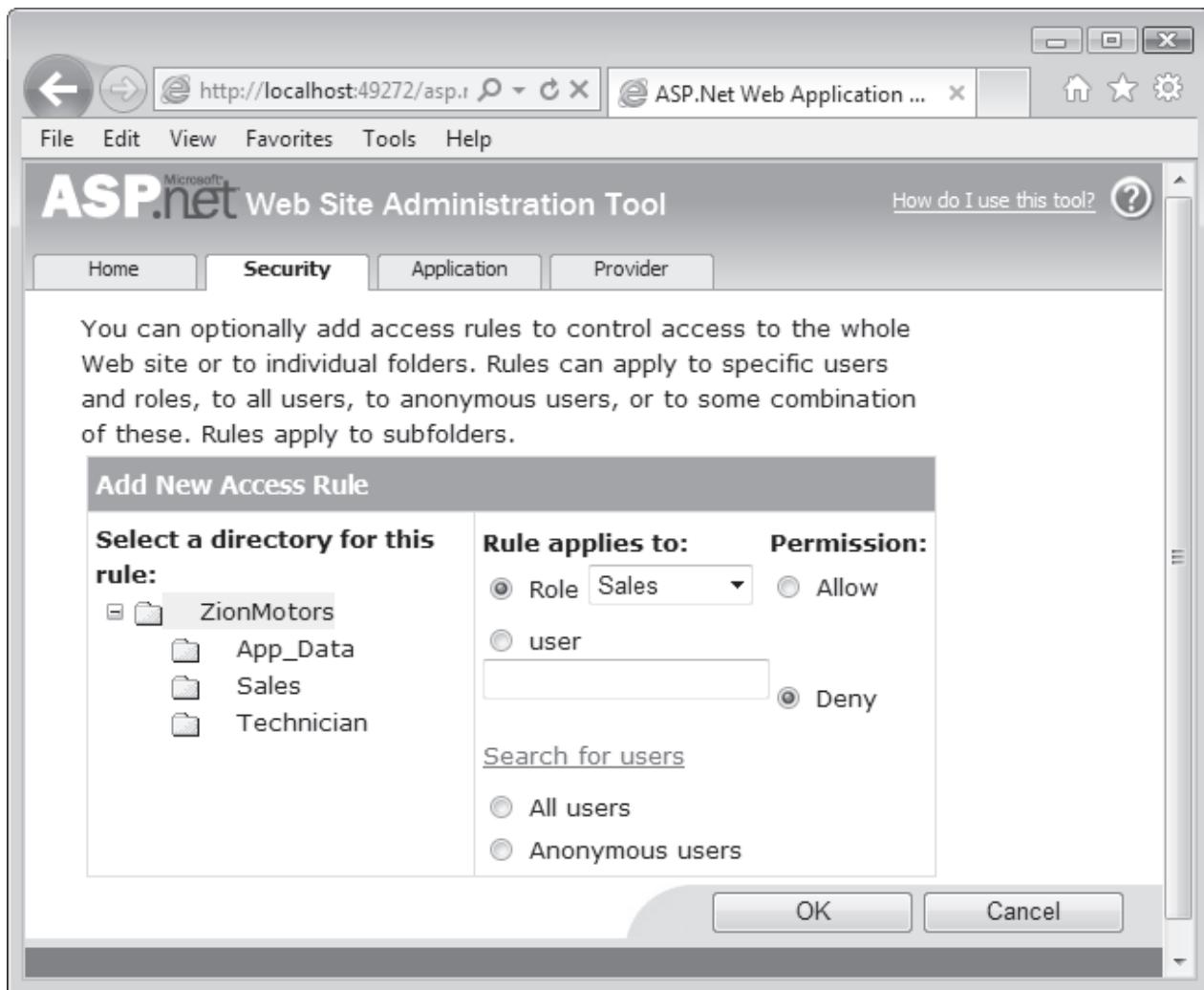


Figure 18.12: Access Rule Page

7. To create an access rule for the **Sales** directory for the **Technician** role, select the values as shown in figure 18.13.

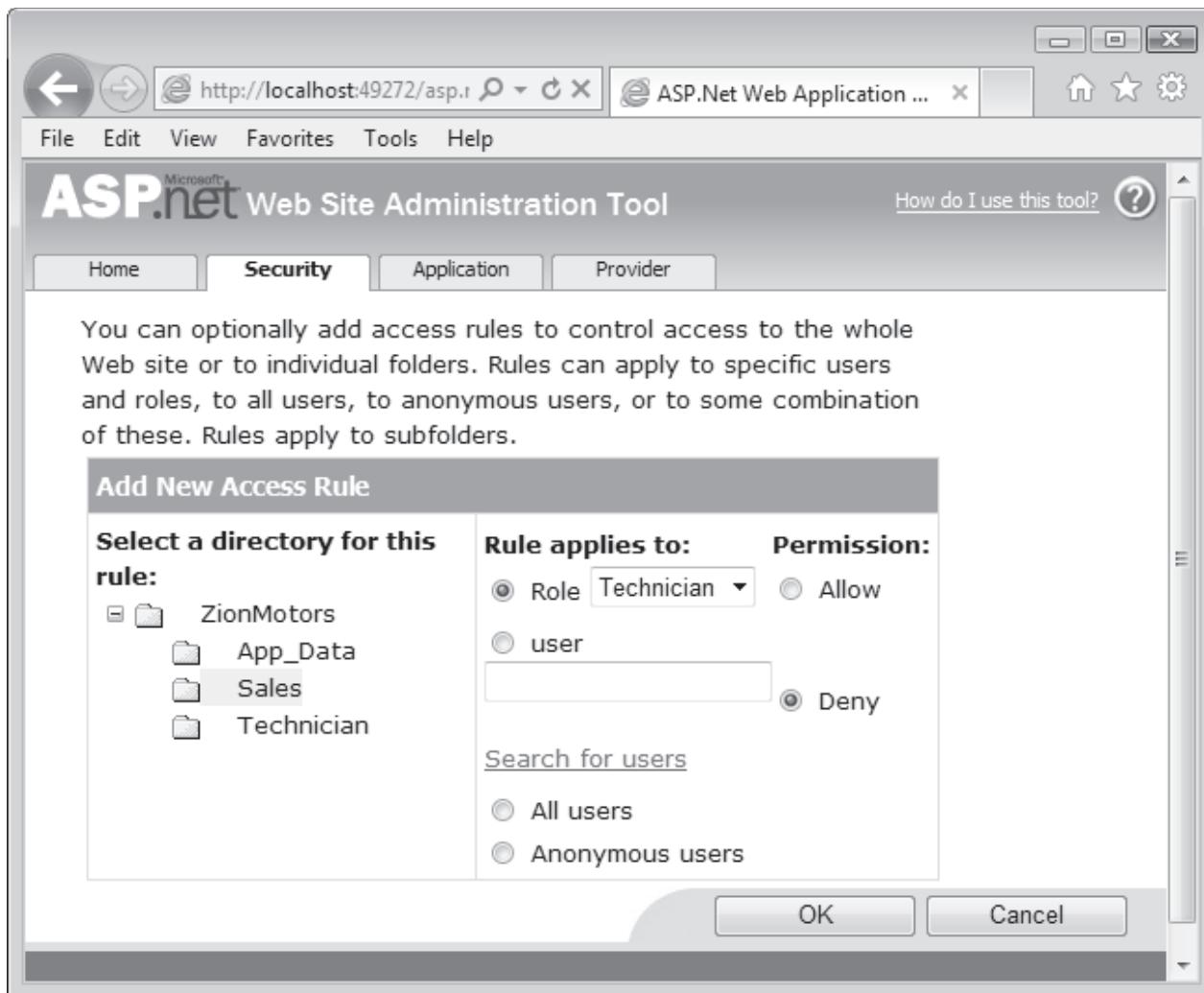


Figure 18.13: Creating Access Rule

8. Click **OK** to complete the creation of the rule.

The new rule that will be set for the Sales directory will apply to the Technician role. The Deny option in the Permission section indicates the access permission that will be set for the role. This means that access to the Sales directory will be denied to users in the Technician role.

9. Similarly, for the **Technician** directory, select the **Sales** role, and permission to **Deny**.

Now, users from the Sales role cannot access pages from Technician directory, and users with Technician role cannot access pages from Sales directory.

Executing the application

You can perform the following steps to build and execute the application:

1. To compile and build the application, on the **Build** menu, click **Build Solution**. The message, **Build**

succeeded will be displayed.

- To execute the application, on the **Debug** menu, click **Start Without Debugging**.

The application is executed and **Login.aspx** page is displayed as shown in figure 18.14.

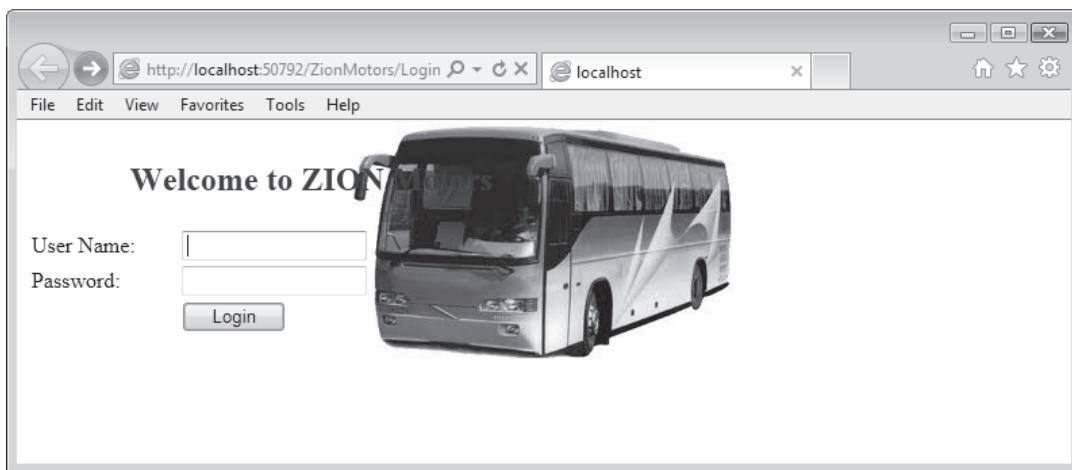


Figure 18.14: Login.aspx Page

- To provide the user details, in the **User Name:** box, type **davidblake** and in the **Password:** box, type **sales_123**.
- Click **Login** to submit the user login details. The Default.aspx page is displayed as shown in figure 18.15.



Figure 18.15: Default.aspx Page Showing the Roles

5. To access the **Sales** page, in Default.aspx, click **Sales Data**. The **Sales.aspx** page is displayed as shown in figure 18.16.

The following table outlines the recommended retail price (RRP) for Bus chassis sold in USA.

Model	Recommended Retail Price
B7R – Euro 5	\$255,000
B7RLE – Euro 5 (4x2)	\$2522,000
B7RLE – Euro 5 (6x2)	\$299,000
B9R – Euro 5*	\$272,000

Figure 18.16: Sales.aspx Page

Since the user **davidblake** is a member of the **Sales** role, user can access the pages in the **Sales** directory.

6. To return to **Default.aspx** page, click **Back** from the browser. The **Default.aspx** page will be displayed.
7. To access the **Technician** page, in **Default.aspx**, click **Technician Details**. Since the user is not a member of the **Technician** role, this action will redirect the user to **Login.aspx**.

18.2 Part II - 60 Minutes

Modify the application given in Part I to perform the following tasks:

- Secure the Web application by using Windows-based authentication
- Add more users to Sales and Technician roles
- Add one more role named Manager, create a user, and assign the role Manager to it
- Specify all rights to the Manager role

Hints:

Use ASP.NET Web Site Administration Tool to create users and roles

Verify in the `web.config` file for the Windows-based authentication

18.3 Do It Yourself

AK506 is a highly secure defense laboratory in USA. The company wants to develop a Web-based application to maintain their employee details. The Web-based application is intended to be available for each employee of AK506. However, some information is highly secure and must be made accessible only to the director and some other senior staff.

Assuming that you are a part of the developing team working on the application logic, perform the following operations in ASP.NET:

- Create roles
- Create users and assign them roles
- Create a Login form to authenticate users of the application
- Create access rules for users belonging to certain roles

Module - 19

Master Pages and Web Parts

Welcome to the module, **Master Pages and Web Parts**. This module describes master pages and how it can help in creating a consistent layout for all pages in an ASP.NET application. The module also explores the methods of creating and using Web Parts.

In this module, you will learn to:

- List and describe the features of master pages
- Describe the creation and use of master pages
- Explain nested master pages
- Define and describe Web Parts
- Explain the essentials of Web Parts
- Describe the `System.Web.UI.WebControls.WebParts` namespace
- Explain the `WebPartManager` and other Web Parts controls



19.1 Introduction

All pages in a professional Web site should have a standardized look. For example, most Web sites have a layout that places a navigation menu on the left side of the page, content in the middle, and the copyright on the bottom. It will be very difficult to maintain a consistent look to put the common pieces in place with every Web page you build. In ASP.NET, master pages make this job easier. A master page is a special type of Web page that serves as a template for one or more Web pages.

Web Parts are a set of controls that allow users to modify the appearance, layout, and content of a Web page. This can be done by using the Web Parts control set. Each control in the set is represented by a class that belongs to the `System.Web.UI.WebControls.WebParts` namespace. In addition to creating Web Parts, you can also connect Web Parts to interchange data.

19.2 Master Pages

Consider a scenario where you have designed a Web application of 15 pages. All of these pages have the same color combination, logos, menus, copyright notices, and specific text. Now, assume that the design of the logo changes. To reflect this change consistently throughout the Web application, you need to modify all 15 pages. Instead, if you design the layout on a special page that identifies the layout for a set of pages, and then, inherit that page in the other pages, you do not have to update the design on all pages of the application but only on the special page. Such a page is called a Master page. When there is a change in any page element, you can make the modification just by updating the Master page. Therefore, using ASP.NET Master pages, you can define consistent pages in the Web application, provide them a common look and feel and control them easily.

ASP.NET Master pages help you to create a template or a consistent layout for multiple pages in a Web application. You can define the standard features of a Web application in a single Master page. After this, the standard features can be repeated throughout all or a group of pages in the Web application by inheriting from the Master page. In this way, you can ensure that the look and feel of all or a group of pages in the Web application follows a consistent pattern.

Figure 19.1 shows the layout of the Web application with the master page.

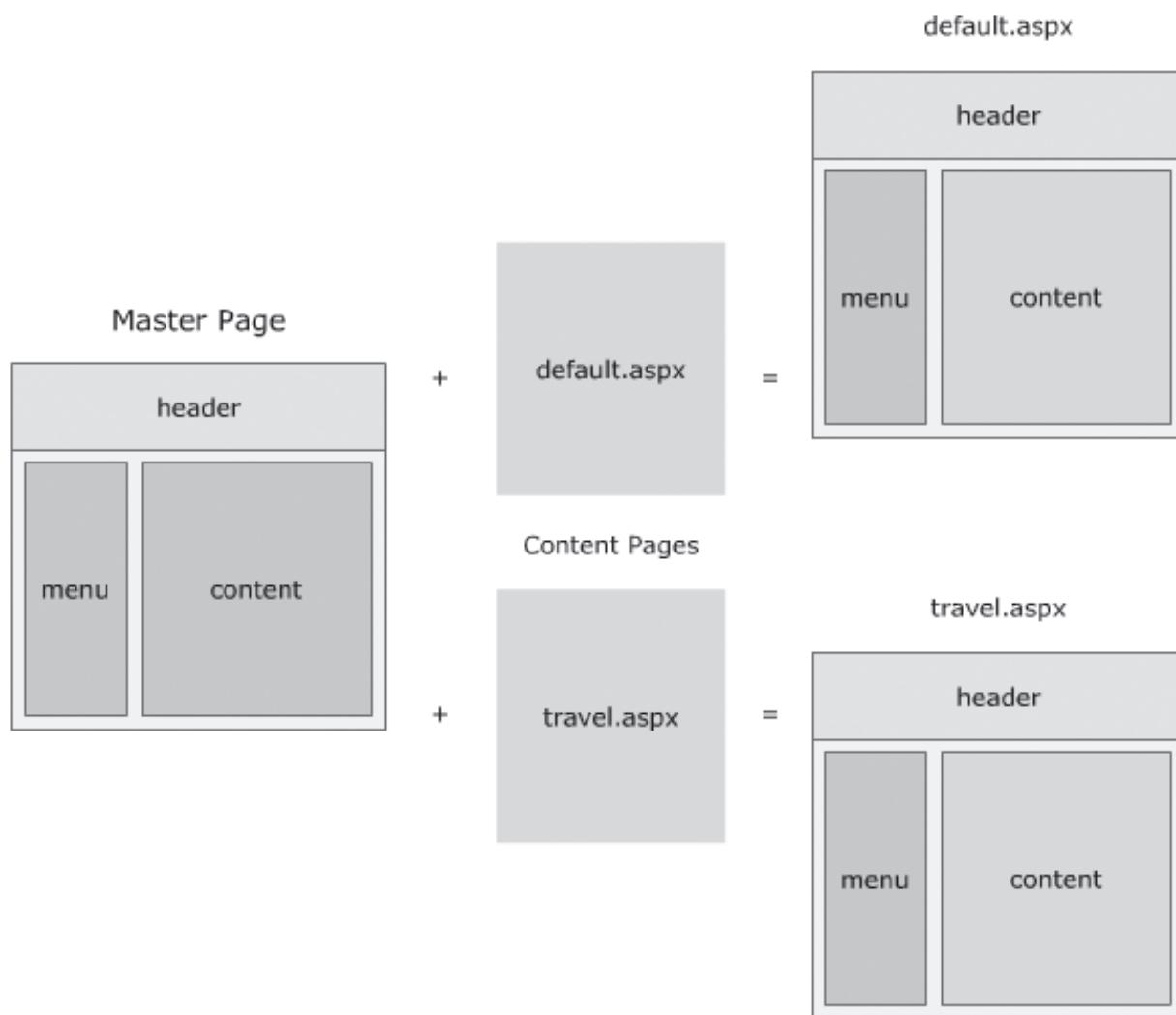


Figure 19.1: Master Pages

19.3 Features of Master Pages

A Master page is an ASP.NET file with the extension .master. A Master page is identified by the keyword `@ Master` directive.

When you make a request for specific content pages, the content in the content pages is displayed in the layout defined by the Master page.

Each Master page consists of two parts:

→ The Master Page

You can design the Master page to define the standard layout and design.

The syntax for the @ Master directive is given below:

Syntax:

```
<%@ Master Language="C#" %>
```

→ **Content Pages**

There can be one or more content pages. A page that uses a Master page is known as the content page. You create individual content pages that display relevant content on user request.

The syntax for the @Page directive in the content page is given as follows:

Syntax:

```
<%@ Page Language="C#" MasterPageFile="" ...>
```

The Master page file is linked to this content page using the `MasterPageFile` keyword.

19.3.1 Elements in a Master Page

Master pages work in the same way as Hypertext Markup Language (HTML) pages with regions that you can edit. The regions on a Master page that you can edit are called content placeholders. The remaining area on the page cannot be customized and is common to all pages on the Web application.

A Master page can contain the top-level HTML elements for a page, such as `html`, `head`, and `form`. You can use any HTML and any ASP.NET elements as part of your Master page.

Therefore, effectively, there are three elements in a Master page:

→ **Content Placeholders**

Content Placeholders are the editable regions on a Master page that you can customize for each page. You can have more than one Content Placeholders on a Master page.

The `ContentPlaceholder` control enables the ASPX Web forms to place content on the Master page in the way you want.

→ **Non-editable regions**

Non-editable regions are the areas where you can insert the common elements such as company logos, menus, or copyright notices.

→ **HTML Elements**

You can use an HTML table to design the layout, an `img` element for creating the company logo, a static text for the copyright notice, and server controls to create standard navigations for the Web application.

19.4 Working of Master Pages

A well-designed Web site always has a consistent site-wide page layout. For example, in figure 19.2, it shows the layout of a master page with four sections at the top, bottom, and right of every page and a ContentPlaceHolder in the middle-left, where the content for each Web page will be displayed.

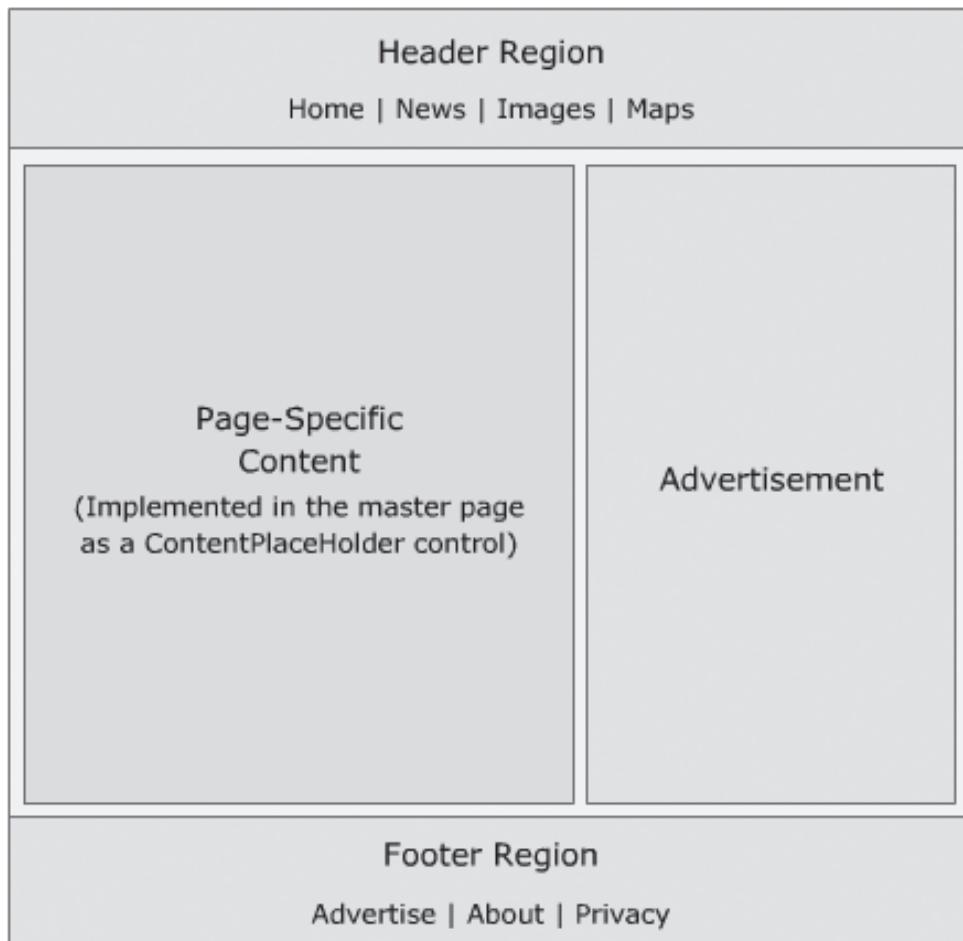


Figure 19.2: Layout of Master Pages

After creating the layout of the Master page, you can bind it to the content pages. These content pages include a Content control for each of the master page's ContentPlaceholder controls. When the content page is visited through a browser, the ASP.NET engine creates the master page's control hierarchy and injects the content page's control hierarchy into the appropriate places. This combined control hierarchy is rendered and the resulting HTML is returned to the end user's browser. Consequently, the content page emits both the common markup defined in its master page outside of the ContentPlaceholder controls and the page-specific markup defined within its own Content controls.

Figure 19.3 illustrates this concept.

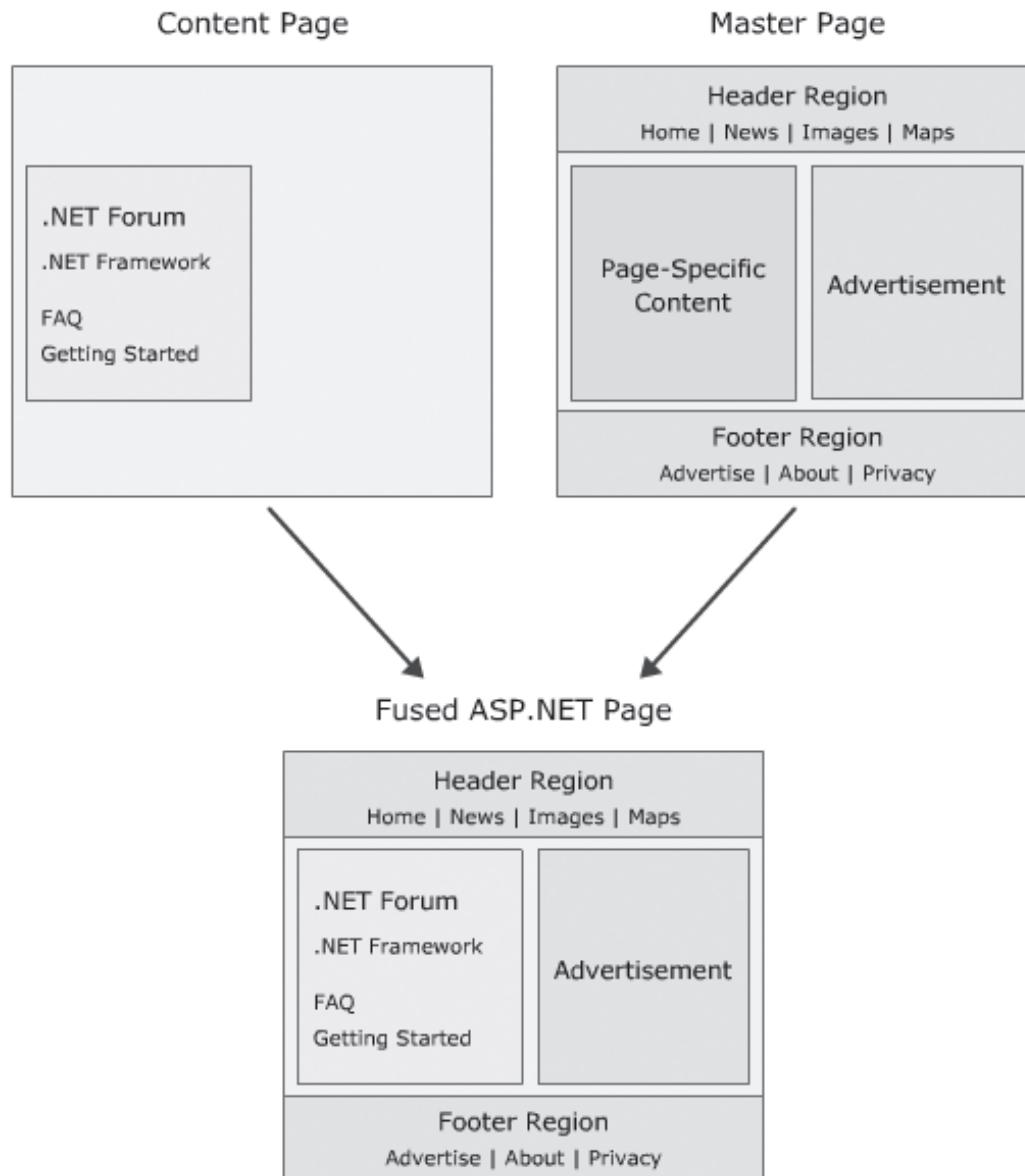


Figure 19.3: Master Page and Content Page

Master pages are advantageous because they:

- ➔ Prevent repeated placement of code, text, and controls on every page.
- ➔ Ease the process of updating the application, as common functionalities are stored at one place instead of multiple pages.
- ➔ Provide an easy way to create a repository of common controls and code, the results of which can be applied to a set of multiple pages.

- Allow you to customize Master page from individual pages.

19.5 Creating and Using Master Pages

If you are creating a new Web site, you may like to start with creating the Master page first. Master pages make it easy to design and control a consistent template or layout for the whole Web site.

You can create a Master page for a Web application in Visual Studio 2008 IDE using the **Add New Item** dialog box and then, selecting the **Master Page** item from the list of displayed templates. The Master Page that is created will have the extension **.master** by default.

19.5.1 Adding Content in a Master Page

A Master page can have content that is uniform for the entire Web site and need not be customized for every page. This is possible because Master pages can support server-side code. To add such dynamic, non page-specific content to the Master page, you can add code in the `Page_Load` event handler. This is useful if, for example, you want to display the news highlights on every page or the name of the user who has just logged in.

Alternatively, by adding content placeholders to a Master page, you can also add content that can be edited and customized for every page. When you add a new Master page, by default, it contains only one content placeholder. However, you can add more than one content placeholder to a Master page by:

→ Using the Designer view

In the Designer view, you can add a new content placeholder to the Master page by dragging and dropping the `ContentPlaceHolder` control from the Toolbox to the Master page.

→ Using a declarative syntax in the HTML view

If you want to add a content placeholder declaratively in the HTML view, you can use the declarative syntax for content placeholders.

The declarative syntax for a `ContentPlaceHolder` is as follows:

Syntax:

```
<asp:Content ID="Content1" ContentPlaceHolderID="mainContent"
runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="menuContent"
runat="server">
</asp:Content>
```

This will add two `ContentPlaceHolder` controls on the page, one with the name `mainContent` which holds the content and the other named `menuContent` which holds the menu links.

19.5.2 Adding Master Pages to an ASP.NET Page

After you have created the Master page for your Web site, you can create the ASP.NET pages for your project. In order to complete the process of creating the ASP.NET page using the Master Page, it is important that you specify the Master page from which a particular ASP.NET page should inherit the layout. If you want different layouts for different parts of your Web site, you can have more than one Master page in the project.

The following code snippet demonstrates the code for the content page:

Code Snippet:

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" Title="Untitled
Page" %>

<asp:Content ID="Content1" ContentPlaceHolderID="mainContent" Runat="server">
Welcome to my world. Know more about me by clicking the links on the right.
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="menuContent" Runat="server">
Home
<a href="AboutMe.aspx">About Me</a><br />
<a href="ContactMe.aspx">Contact Me</a><br />
<a href="http://www.google.com/">Search more about Me</a>
</asp:Content>
```

The code will create two ContentPlaceHolder controls named mainContent and menuContent. While the mainContent will hold some text content, menuContent will hold three menu items: **About Me**, **Contact Me**, and **Search more about Me**. Also, as this is inheriting the **MasterPage.master** file, the look and feel will be derived from that master file.

19.6 Nested Master Pages

In addition to Master pages, ASP.NET also allows you to create and use nested Master pages. A nested Master page is a Master page that refers to another Master page as its master.

You can include more than one nested Master pages into a Master page and there is no limit to the number of nested or child masters that you can create. For example, you can create a parent master page with the company banner at the top and in a side column the site navigation controls. Then, you can create a child master page for a specific product or department that uses the parent master page. The child master page can then act as a master page for all other related department or product pages. Like this, each department or product line will have a consistent look, and all pages will have a consistent overall look by using the parent master page.

Figure 19.4 shows the layout of the pages and the relationship between them in the context of how they get displayed.

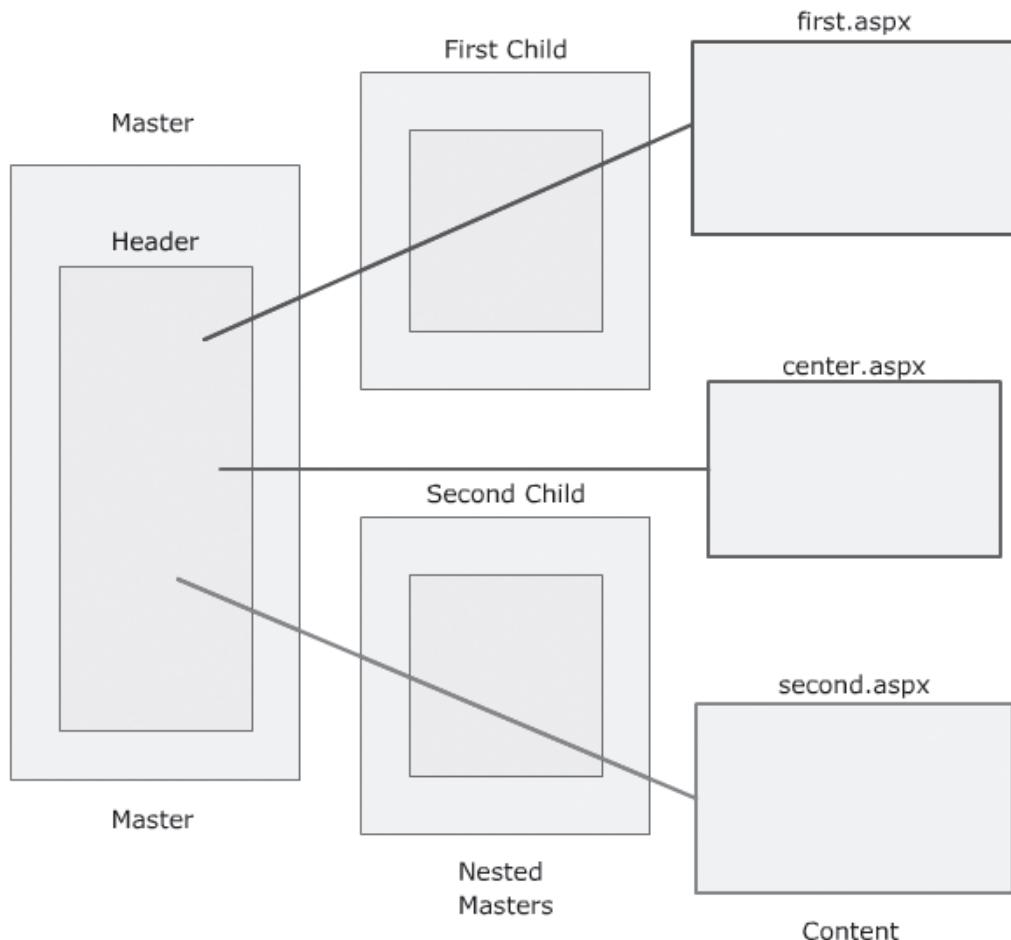


Figure 19.4: Nested Master Pages

You can create a parent Master page for the entire Web site. Then, you can nest several child Master pages corresponding to each section of the Web site into this parent page.

The following steps can be used to create a nested Master Page:

1. Create a Web Site.
2. Create a Parent Master Page.
3. Create a Child Master Page.
4. Create a Content Page that uses the Child Master Page.

19.6.1 Features of Nested Master Pages

The nested or the child Masters also have the extension `.master`. They have several specific features such as:

- Nested Master pages have controls that are mapped to the content placeholders in the parent Master page, and hence, the layout of the nested Master pages is similar to the Parent. They can, however, contain their own content placeholders.
- You can have any number of nested Master pages and till any depth. This will not affect the performance of the application.
- Child Master pages can provide uniqueness to individual pages while the parent Master page defines the look and feel of the total Web site.
- Unlike the parent Master page, the child Master page applies only to the body of a page.

19.7 Introduction to Web Parts

Consider a scenario where you have developed an online news portal that displays the weather reports, stock updates, and so on. You want to allow the users to personalize the Web page according to their requirements. This means that the user should be able to change the background or font color, page layout, size of search box, or font size. In addition, the users should also be able to add content to share information with other users accessing that Web site. This process of personalizing a Web page is done using Web Parts in ASP.NET.

Web Parts are a set of controls that allow users to personalize the appearance and content of Web pages.

Figure 19.5 shows how Web Parts controls are used to build personalized Web pages.



Figure 19.5: Using Web Parts Controls

19.8 Web Parts

Web Parts enable users at the client end to dynamically personalize a Web page without any assistance from a developer or administrator. This means that by developing Web Parts, you allow users to personalize the Web page as per their requirements. This feature of allowing users to personalize a Web page is referred to as personalization. ASP.NET provides the `System.Web.UI.WebControls.WebParts` namespace that contains various Web Parts controls. Web Parts enable end users to:

→ Customize Web Page Content

Allow end users to add, hide, and remove Web Parts controls. Users can also minimize controls like a normal browser window.

→ Customize Web Page Layout

Allow end users to customize the appearance and behavior of Web Parts controls. Users can also change the position of Web Parts controls by dragging and dropping them anywhere on the Web page.

→ Import and Export Web Parts Controls

Allow end users to import or export settings applied to Web Parts controls across various Web pages. This allows users to use the control settings in other Web pages.

→ **Establish Connection Between Controls**

Allow end users to establish connection between Web Parts controls. For example, you can link the Web Parts control if you want to display detailed information and a graph for the same share report.

19.8.1 Web Parts Essentials

ASP.NET provides the Web Parts controls set that allows you to create Web pages that can be personalized by end users. The Web Parts controls set consists of three main building blocks, that are as follows:

→ **Personalization**

Allows end users to customize the appearance, positioning, and behavior of Web Parts controls. The modified settings are not just saved for the current browser session but these settings are permanently saved. This means that when a user visits a specific page again in the future, the saved settings are retrieved.

→ **UI Structural Components**

Represents two important UI structural components for using Web Parts controls on a Web page. The most important component is the `WebPartManager` control that contains Web Parts zone controls.

The second important component is the Web Parts zone controls that provide a layout to contain and arrange the different Web Parts controls.

→ **Web Parts UI Controls**

Provides a list of Web Parts controls that are derived from the `Part` class. The `Part` class acts as a base class for all the part controls such as `WebPart` and `EditorPart`.

19.8.2 Web Parts Controls

ASP.NET provides a wide range of Web Parts controls. Table 19.1 lists the most commonly used Web Parts controls.

Control	Description
CatalogPart	Provides a list of Web Parts controls that the users can include in a Web page.
CatalogZone	Includes the CatalogPart controls.
EditorPart	Allows editing the Web Parts controls.
EditorZone	Includes the EditorPart controls.
WebPart	Represents a collection of custom Web Parts controls.
WebPartManager	Manages the Web Parts controls included within a Web page
WebPartZone	Includes the WebPart controls

Table 19.1: Web Parts Controls

Figure 19.6 shows the various Web Parts controls in the Visual Studio Toolbox.

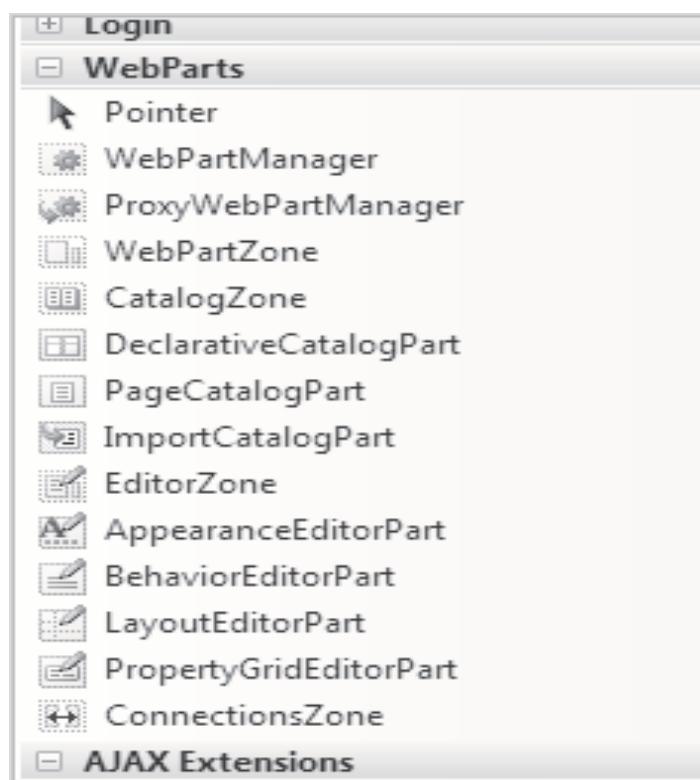


Figure 19.6: Web Parts Controls

19.9 System.Web.UI.WebControls.WebParts Namespace

The `System.Web.UI.WebControls.WebParts` namespace provides a collection of classes and interfaces that allow you to create Web pages that can be personalized by end users. Table 19.2 lists the most commonly used classes of the `System.Web.UI.WebControls.WebParts` namespace.

Class	Description
CatalogPart	Acts as a base class that provides a list of Web server controls that the end users can add to a Web page.
ConnectionsZone	Allows establishing connection between the two Web Parts controls.
EditorPart	Acts as a base class to edit and personalize the associated Web Parts control.
Part	Acts as a base class for all Web Parts controls such as EditorPart and CatalogPart.
WebPart	Acts as a base class for creating custom Web Parts controls.
WebPartManager	Manages the Web Parts controls of a Web page.
WebPartZone	Acts as a base class for all Web Parts zones that contain Web Parts controls, server controls, and custom controls.
WebPartZoneBase	Acts as a base class for all the zone controls.

Table 19.2: Classes in `System.Web.UI.WebControls.WebParts`

19.9.1 WebPartManager Control

The **WebPartManager** control is a server control that manages the Web Parts controls contained within a Web page. If you are using the Web Parts control set for your Web page, you should include the **WebPartManager** control. The **WebPartManager** control manages all the personalization settings of the Web page. In addition, it manages the communication between different Web Parts controls. There are various tasks that the **WebPartManager** control allows you to perform, some of which are as follows:

- Maintain track of the different Web Parts controls within a Web page
- Allow adding, deleting, and removing the Web Parts controls
- Establish connection between the Web Parts controls
- Allow personalizing the Web page by moving controls to different locations, modify the appearance and behavior of controls
- Manage events that are triggered when a control is added, modified, or removed
- Allow switching between different Web page views to carry out different actions such as changing layout or colors

The **WebPartManager** class allows creating the **WebPartManager** control. It provides members that allow you to manage Web Parts controls.

ASP.NET allows you to set different display modes for a Web page. Display mode represents a state in which certain elements of a user interface are visible while others are hidden. These display modes allow end users to perform various tasks such as editing and personalizing the Web page. At a given point of time, a Web page can be only in one display mode. The **WebPartManager** class provides different fields to manage the different display modes. Table 19.3 lists the most commonly used fields of the **WebPartManager** class.

Field	Description
BrowseDisplayMode	Restricts end users from editing or shifting controls within a Web page.
CatalogDisplayMode	Allows end users to add controls from a catalog of controls.
DesignDisplayMode	Allows end users to reorganize the Web Parts within a Web page.
EditDisplayMode	Provides end users to edit the controls within a Web page

Table 19.3: **WebPartManager** Fields

The **WebPartManager** class provides properties that allow you to track and retrieve a collection of Web Parts controls. Table 19.4 lists the most commonly used properties of the **WebPartManager** class.

Property	Description
----------	-------------

Controls	Retrieves a collection of all the WebPart, server, or user controls contained within zones.
DisplayMode	Specifies or retrieves an active display mode for a Web page containing Web Parts controls.
Personalization	Retrieves an object containing all the personalization settings for a Web page.
SelectedWebPart	Retrieves a reference to Web Part control that is selected for editing.
WebParts	Retrieves a reference to track Web Parts controls within the WebPartManager control.

Table 19.4: WebPartManager Properties

The `WebPartManager` class provides methods that allow you to add new Web Parts controls, verify if the users are authorized to add new Web Parts, and so on. Table 19.5 lists the most commonly used methods of the `WebPartManager` class.

Method	Description
<code>AddWebPart</code>	Allows adding a WebPart control programmatically.
<code>CreateWebPart</code>	Enable server controls that are not WebPart controls to have the same functionality as WebPart control.
<code>ExportWebPart</code>	Exports data such as state data, property data, and assembly data from Web Parts to an XML file.
<code>Focus</code>	Prevents focus to be set on the WebPartManager control.
<code>GetCurrentWebPartManager</code>	Retrieves a reference to the current WebPartManager control.
<code>IsAuthorized</code>	Determines whether users can add WebPart or other server controls to a Web page.

Table 19.5: WebPartManager Methods

19.9.2 WebPartZone Class

A zone is an area on a Web page that holds the Web Parts controls. The `WebPartZone` class provides a user interface to contain the WebPart controls. The `WebPart` class is a base class to create custom Web Parts controls. These custom controls extend the functionalities of the part controls, which are derived from the `Part` class.

Apart from the `WebPart` controls, `WebPartZone` can contain other user controls, server controls, or ASP.NET controls. By adding these controls in the `WebPartZone`, you give them the functionality of Web Parts.

Figure 19.7 displays the Web Parts manager with three Web Part zones.

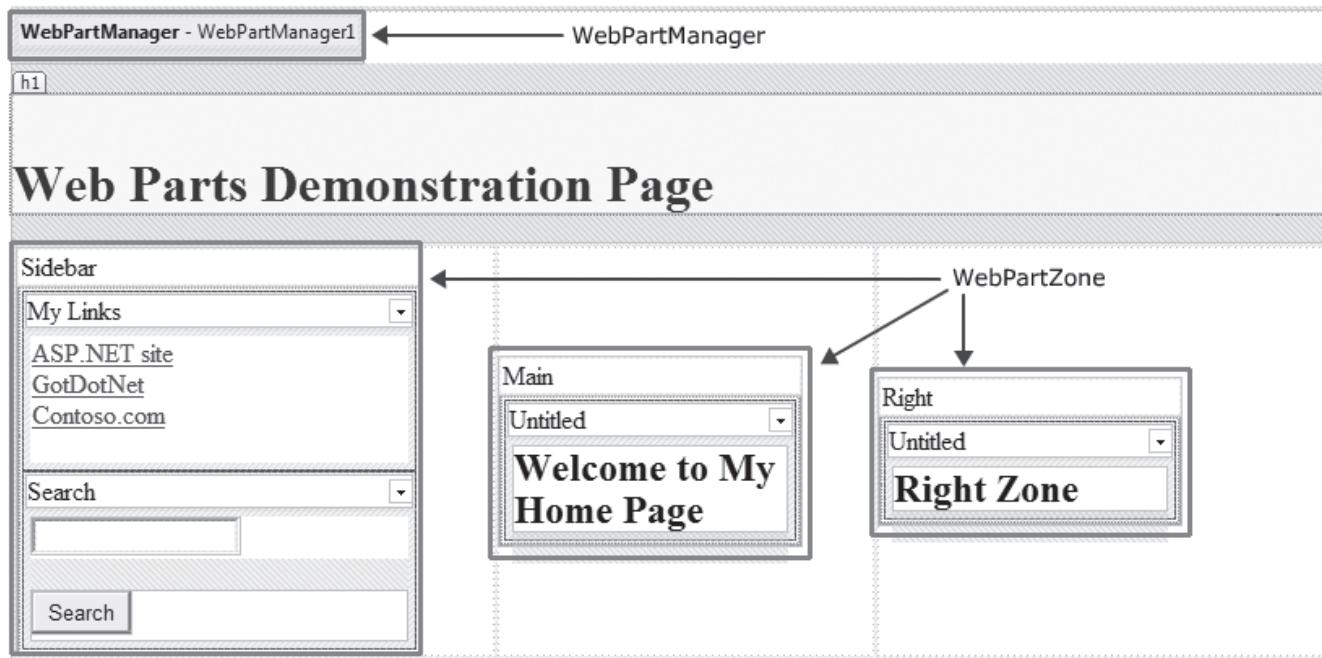


Figure 19.7: Web Part Zones

The `WebPartZone` class provides various properties that are inherited from the `WebPartZoneBase` class. The `WebPartZoneBase` class acts as a base class to contain zone controls, which hold the different WebPart and server controls.

Table 19.6 lists the commonly used properties of the `WebPartZone` class.

Property	Description
<code>DisplayTitle</code>	Retrieves a value from the <code>HeaderText</code> property to set the title of a zone.
<code>HeaderText</code>	Specifies or retrieves the header text for a zone.
<code>VerbButtonType</code>	Specifies or retrieves the type of buttons associated with the verbs in the <code>WebPartZoneBase</code> class. The <code>WebPartZoneBase</code> class allows you to handle verbs that specify the action the user can perform. For example, controls like buttons and links contain standard verbs such as OK, Close, and Edit.
<code>VerbStyle</code>	Retrieves the different style attributes for verbs that are related to Web Parts controls.
<code>WebParts</code>	Retrieves a set of Web Parts controls contained with a zone.

Table 19.6: WebPartZone Properties

The following code snippet uses the `WebPartZone` class to convert the `Calendar` control to a `Web Part` control.

Code Snippet:

```
protected void Page_Init (object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        WebPartManager wpmDiary = new WebPartManager();
        wpmDiary.ID = "WpmDiary";
        frmSearch.Controls.Add (wpmDiary);
        WebPartZone wpzDiaryZone = new WebPartZone();
        wpzDiaryZone.ID = "wpzSearchZone";
        wpzDiaryZone.HeaderText = "To Do";
        frmsearch.Controls.Add (wpzDiaryZone);
        Calendar calDiary = new Calendar();
        calDiary.ID = "calDiary";
        WebPart wpCalendar = wpmDiary.CreateWebPart (calDiary);
        wpCalendar.Title = "Calendar";
        wpmDiary.AddWebPart (wpCalendar, wpzDiaryZone, 0);
    }
}
```

The code `wpzDiaryZone.HeaderText = "To Do"` specifies the header text for the zone.

Then, it adds the `Web Parts` zone to the `Web Form` by using the `Add()` method. The `CreateWebPart()` method converts the `calDiary` control to a `Web Part` control. The `AddWebPart()` method adds the `Web Part` control, `wpCalendar` to the `Web Parts` zone, `wpzDiaryZone`.

19.9.3 CatalogPart Control

The `CatalogPart` control provides a catalog of controls that the end users can add to a `Web page`. A catalog represents a list of `Web Parts` controls, server controls, user controls, or custom controls. In addition, a catalog contains various functionalities such as common header, footer, border, and descriptive text for server control.

The `CatalogPart` controls are placed in `CatalogZoneBase`. The `CatalogZoneBase` class acts as a base class for zone controls. These zone controls act as containers to hold lists of `Web Parts` controls that a user can add to a `Web page`.

You can include CatalogPart controls if you want to allow end users to add or remove server controls. The Web Parts control set provides three types of CatalogPart controls. Table 19.7 lists the types of CatalogPart controls that can be added to a Web page.

Property	Description
DeclarativeCatalogPart	Enables you to declaratively add a set of server controls to a catalog.
ImportCatalogPart	Imports a description file containing the settings of WebPart or server controls to allow users to add the settings to a zone.
PageCatalogPart	Maintains a list of the previously added controls to a page that has been closed. These controls that are maintained can be added back to the page.

Table 19.7: Types of CatalogPart Controls

19.9.4 CatalogPart Class

The CatalogPart class allows creating the CatalogPart control. It is inherited from the Part class. Table 19.8 lists the commonly used properties and methods of the CatalogPart class.

Name	Description
DisplayTitle	This property retrieves the title of the CatalogPart control.
GetAvailableWebPartDescriptions()	This method retrieves the explanation of the Web Parts controls within a catalog.
GetWebPart()	This method retrieves the Web Part control identified by the WebPartDescription object, which contains description about the WebPart control.

Table 19.8: CatalogPart Class – Commonly used Properties and Methods

The following code snippet creates a custom template for adding PageCatalogPart controls to the CatalogZone zone:

Code Snippet:

```
public class CustomTemplate : WebPart, ITemplate
{
    void System.Web.UI.ITemplate.InstantiateIn(Control container)
    {
        CatalogPart cpPageCatalog = new PageCatalogPart();
        cpPageCatalog.ID = "cpPageCatalog";
        cpPageCatalog.Title = "Page Catalog";
        container.Controls.Add(cpPageCatalog);
    }
}
```

The `InstantiateIn()` defines a method that specifies the `Control` object, which contains different controls. Then, the values of the `ID` and `Title` properties are set. Finally, the `Add()` method adds the `PageCatalogPart` control to the container, `CatalogZone`.

The following code snippet adds the `PageCatalogPart` control to `CatalogZone` control:

Code Snippet:

```
CatalogZone czUtilities = new CatalogZone();
CatalogPart pcpPageCatalog = new PageCatalogPart();

protected void Page_Init(object sender, EventArgs e)
{
    czUtilities.ID = "czUtilities";
    czUtilities.ZoneTemplate = new CustomTemplate();
    frmMegaShopping.Controls.Add(czUtilities);
}

protected void btnAddToZone_Click(object sender, EventArgs e)
{
    pcpPageCatalog = czUtilities.CatalogParts[0];
    WebPartDescriptionCollection descriptionsCollection = pcpPageCatalog.
    GetAvailableWebPartDescriptions();
    try
    {
        WebPartDescription wpdDescription = descriptionsCollection[0];
        WebPart wp = pcpPageCatalog.GetWebPart(wpdDescription);
        wpmMegaShopping.AddWebPart(wp, wpzUtilitiesNew, 0);
    }
    catch (Exception ex)
    {
        Response.Write("Catalog is empty");
    }
}
```

The `ZoneTemplate` property of the `CatalogZone` class retrieves the created custom template to add `PageCatalogPart` controls to zone. Then, it retrieves the first `PageCatalogPart` control from the `CatalogZone` collection by using the code `czUtilities.CatalogParts[0]`. The

`GetAvailableWebPartDescriptions()` method retrieves the explanation of Web Parts controls within the retrieved catalog.

The `GetWebPart(wpdDescription)` method retrieves the Web Part control identified by the `wpdDescription`, which contains description about the WebPart control. Finally, the `AddWebPart(wp, wpzUtilitiesNew, 0)` adds the Web Part control to the WebPartZone.

19.9.5 EditorPart Control

The `EditorPart` control allows end users to edit Web Parts controls that reside in `EditZoneBase`. The `EditorZoneBase` class acts as a base class for zone controls that act as containers to hold controls that enable users to edit Web Parts controls. The `EditorPart` control allows editing the appearance, layout, and other functionalities of the associated Web Parts control.

The users can edit the `WebPart` control only when a Web page is in edit mode. Once the user enters the edit mode, the user can select the `WebPart` control to edit it. The Web Parts control set provides several editing controls that are derived from the `EditorPart` control. Table 19.9 lists the types of `EditorPart` controls that can be added to a Web page.

Name	Description
AppearanceEditorPart	Allows end users to edit the appearance of the control such height, width, and text.
BehaviorEditorPart	Allow end users to edit certain behaviors such as edit, close, or move the control.
LayoutEditorPart	Allows end users to edit the layout properties of the control.
PropertyGridEditorPart	Allows end users to edit custom properties of controls.

Table 19.9: Types of EditorPart Controls

19.9.6 EditorPart Class

The `EditorPart` class allows creating the `EditorPart` control. It is inherited from the `Part` class. Table 19.10 lists the properties and methods of the `EditorPart` class.

Name	Description
Display	This property determines whether a control associated with the Web Part control should be displayed when the Web Part control is in edit mode.
DisplayTitle	This property retrieves the text to be displayed on the title bar of the <code>EditorPart</code> control.
WebPartToEdit	This property retrieves a reference of the recently edited Web Part control.
ApplyChanges()	This method saves the properties' values of the <code>EditorPart</code> control to reflect the changes made by the end user to the related Web Parts control.
SyncChanges()	This property retrieves the current properties' values from the WebPart control to the <code>EditorPart</code> control so that the user can edit them.

Table 19.10: EditorPart Class – Properties and Methods

The following code snippet uses the EditorPart control to enable the user to modify the background color and font size of a control:

Code Snippet:

```
public class CustomFormatEditorPart : EditorPart
{
    private DropDownList ddlColors = new DropDownList();
    private DropDownList ddlFontSize = new DropDownList();

    public override bool ApplyChanges()
    {
        EnsureChildControls();
        GenericWebPart genwp = (GenericWebPart)WebPartToEdit;
        WebControl wcControl = (WebControl)genwp.ChildControl;

        wcControl.BackColor = System.Drawing.Color.FromName(ddlColors.SelectedValue);
        wcControl.Font.Size = FontUnit.Parse(ddlFontSize.SelectedValue);
        return true;
    }
}
```

```

public override void SyncChanges()
{
    this.Title = "Edit";
}

protected override void CreateChildControls()
{
    ddlColors.Items.Add(new ListItem("Silver"));
    ddlColors.Items.Add(new ListItem("Yellow"));
    ddlColors.Items.Add(new ListItem("Gray"));

    ddlFontSize.Items.Add(new ListItem("Small"));
    ddlFontSize.Items.Add(new ListItem("Medium"));
    ddlFontSize.Items.Add(new ListItem("Large"));

    this.Controls.Add(ddlColors);
    this.Controls.Add(ddlFontSize);
}
}

```

The code created two drop-down list boxes named `ddlColors` and `ddlFontSize`. The `ApplyChanges()` method is overridden to save the property values of the `EditorPart` control. The `EnsureChildControls()` method ensures that the current server control has child controls. The `(GenericWebPart)WebPartToEdit` retrieves a reference of the recently edited Web Part control by type-casting it to `GenericWebPart`. The `GenericWebPart` class includes all the controls that are not Web Parts controls.

The code `(WebControl)genwp.ChildControl` creates an instance of the `WebControl` class to retrieve the child controls of the `GenericWebPart` class.

The `SyncChanges()` method is overridden to allow users to edit the control. Finally the controls are populated with values and added to the container.

19.10 Check Your Progress

1. Which of the following statements about Web Parts are true?
 - a. Personalization represents two important UI structural components for using Web Parts controls on a Web page.
 - b. The Web Parts zone controls contain the WebPartManager control.
 - c. Web Parts allow end users to import or export settings applied to Web Parts controls across multiple Web pages.
 - d. Web Parts allow end users to personalize the appearance but not the content within the Web page.
 - e. Web Parts UI controls provide a list of Web Parts controls that are derived from the Part class.

(A)	a, b, d	(C)	c, e
(B)	a, c, e	(D)	b, d, e

2. Match the different Web Parts controls with their corresponding descriptions.

	Web Part Control		Description
(a)	EditorPart	(1)	Provides a list of Web Parts controls that the users can include in a Web page.
(b)	WebPartManager	(2)	Includes the CatalogPart controls.
(c)	CatalogPart	(3)	Allows editing the Web Parts controls.
(d)	EditorZone	(4)	Includes the EditorPart controls
(e)	CatalogZone	(5)	Manages the Web Parts controls included within a Web page.

(A)	a-5, b-1, c-2, d-3, e-4	(C)	a-2, b-3, c-4, d-1, e-2
(B)	a-4, b-5, c-1, d-2, e-3	(D)	a-3, b-5, c-1, d-4, e-2

3. Which of the following statements about different Web Parts controls and classes are true?
 - a. The CatalogPart class acts as a base class for providing a list of Web server controls that can be edited by end users.
 - b. The DisplayTitle property of the WebPartZone class specifies or retrieves the header text for

a zone.

- c. The WebPartManager control allows adding, deleting, and removing the Web Parts controls.
- d. The PageCatalogPart control allows adding a set of server controls declaratively.
- e. The AppearanceEditorPart control allows end users to edit the appearance of the control such height, width, and text.

(A)	b, c, d	(C)	a, b, c
(B)	a, c, e	(D)	a, d, e

4. Master pages have two parts and three elements. Match the parts or elements with their respective features.

	Feature		Element
(a)	Can design the layout to create standard navigations for the Web application Master page in the way you want.	(1)	Content placeholders
(b)	Enable the ASPX Web forms to place content on the Master page.	(2)	HTML Elements
(c)	Can be used to insert the common elements such as company logos, menus, or copyright notices.	(3)	Content pages
(d)	Display relevant content on user request.	(4)	Non-editable regions

(A)	a-3, b-4, c-1, d-2	(C)	a-2 , b-1, c-4, d-3
(B)	a-2, b-3, c-4, d-1	(D)	a-4, b-1, c-2, d-3

5. Which of the following statements about the features of ASP.NET Master pages are true?

- a. Master pages consist of one or more content pages.
- b. Master pages are used for configuring session information.
- c. Master pages enable specifying the access authentication method.
- d. Master pages enable creating a template for multiple pages in an application.
- e. Master pages can contain HTML elements such as html, head, and form.

(A)	b, c, d	(C)	a, b, c
(B)	c, d, e	(D)	a, d, e

6. Which of the following statements about features of nested Master pages are true?
- Multiple child masters can load multiple content files into different content placeholders on the main Master file.
 - You can add more than one nested Master page into a parent Master page.
 - Adding multiple nested Master pages into a parent Master page decreases the performance of the parent page.
 - Nested Master pages have all the controls that are mapped to content placeholders on the parent Master page.
 - Using the Visual Studio or Visual Web Developer, you can create a nested Master page only in the Designer view.

(A)	a, b, c	(C)	a, b, d
(B)	c, d, e	(D)	a, b, e

7. Which one of the following codes correctly creates a master page?

(A)	<pre> <head runat="server"> <title></title> <asp:ContentHolder ID="head" runat="server"> </asp:ContentHolder> </head> <body> <form id="form1" runat="server"> <div> <h2> My Site</h2> <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server"> </asp:ContentPlaceHolder> </div> <div> <h6> Copyright (c) Michigan Technologies 2011</h6> </div> </form> </body> </pre>
-----	--

(B)

```
<head runat="server">
    <title></title>
    <asp:ContentPlaceHolder ID="head" >
        </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h2>
                My Site</h2>
            <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
runat="server">
                </asp:ContentPlaceHolder>
            </div>
            <div>
                <h6>
                    Copyright (c) Michigan Technologies 2011</h6>
                </div>
            </div>
        </form>
    </body>
```

(C)	<pre> <head runat="server"> <title></title> <asp:ContentPlaceHolder ID="head" runat="server"> </asp:ContentPlaceHolder> </head> <body> <form id="form1" runat="server"> <div> <h2> My Site</h2> <asp:ContentPlaceHolder ID="ContentPlaceholder1" runat="server"> </asp:ContentPlaceHolder> </div> <div> <h6> Copyright (c) Michigan Technologies 2011</h6> </div> </div> </form> </body> </pre>
(D)	<pre> <body> <form id="form1" runat="server"> <div> <h2> My Site</h2> <asp:ContentHolder ID="ContentPlaceholder1" runat="server"> </asp:ContentHolder> </div> </pre>

Module Summary

In this module, **Master Pages and Web Parts**, you learnt about:

→ **Master Pages**

Master pages help you to create a template or a consistent layout for multiple pages in a Web application. A Master page is an ASP.NET file with the extension .master. A nested Master page is a Master page that refers to another Master page as its master.

→ **Web Parts**

Web Parts are a set of controls that allow users to personalize the appearance, layout, and content of Web pages. ASP.NET provides a diverse set of Web Parts controls, where each control is represented by different class. The WebPartManager class manages the Web Parts of a Web page. ASP.NET allows you to connect Web Parts for sharing and interchanging data.

Module - 20

Master Pages and Web Parts (Lab)

Welcome to the module, **Master Pages and Web Parts (Lab)**.

In this module, you will learn to:

- Create an application that uses Web Parts
- Use Master Pages to create applications

20.1 Part I – 60 Minutes

Exercise 1

DBHeights College was affiliated to the University of Michigan on November 18, 1972. The college awards degrees in arts, science, and commerce.

The college is very famous for its well-designed curriculum. A governing body of the Society of Michigan manages the college. The director of the governing body serves as the Rector of the college, and the Chief Executive of the body serves as the Principal of the college.

The college is planning to create a Web-based solution to store and display the details of the students. The Web-based solution will provide an application to staff members of DBHeights to view the student details online. Using this application, the staff can view the student details, minimize, close, and

drag-drop the data appearing on the browser to various parts in the window.

Consider that you are one of the programmers in the IT team and are assigned the following tasks:

- Create the application that displays the student details
- Display the calendar in one part of the window
- Allow the user to drag and drop the details from one part of the window to another part
- Allow the user to minimize and close parts of the window
- You must use Visual Studio 2008 and ASP.NET 3.5 to create the application.

Solution:

The solution is to develop a Web-based application that creates Web Parts and displays data in those Web Parts.

Creating a database to store the student details

To create a database named DBHeights in SQL Server 2008, perform the following steps:

1. Open Microsoft SQL Server Management Studio.
2. Create a database named **DBHeights**.
3. Create a table named **Student** with the structure specified in figure 20.1.

Column Name	Data Type	Allow Nulls
StudentID	nchar(4)	<input type="checkbox"/>
StudentName	nchar(20)	<input type="checkbox"/>
Stream	nchar(10)	<input type="checkbox"/>

Figure 20.1: Student - Table Design

- Add a few records in the table.

Adding WebPartManager and WebPartZone controls to the Web application

- Create an ASP.NET Web site named **DBHeights**. The ASP.NET Web site named **DBHeights** will be created with **Default.aspx** as the Home page.
- Change the title of Home page to '**DBHeights College**' and form id to '**frmHome**'.
- Specify the heading of the page as '**DBHeights College**'.
- Switch to **Design** mode and drag a **WebPartManager** control from the WebParts **Toolbox** tab to the Home page.
- Change the **ID** of the WebPartManager control to **wpmControl**. The resulting tag is displayed as follows:

```
<asp:WebPartManager ID="wpmControl" runat="server">
</asp:WebPartManager>
```

- Design the user interface as shown in figure 20.2.

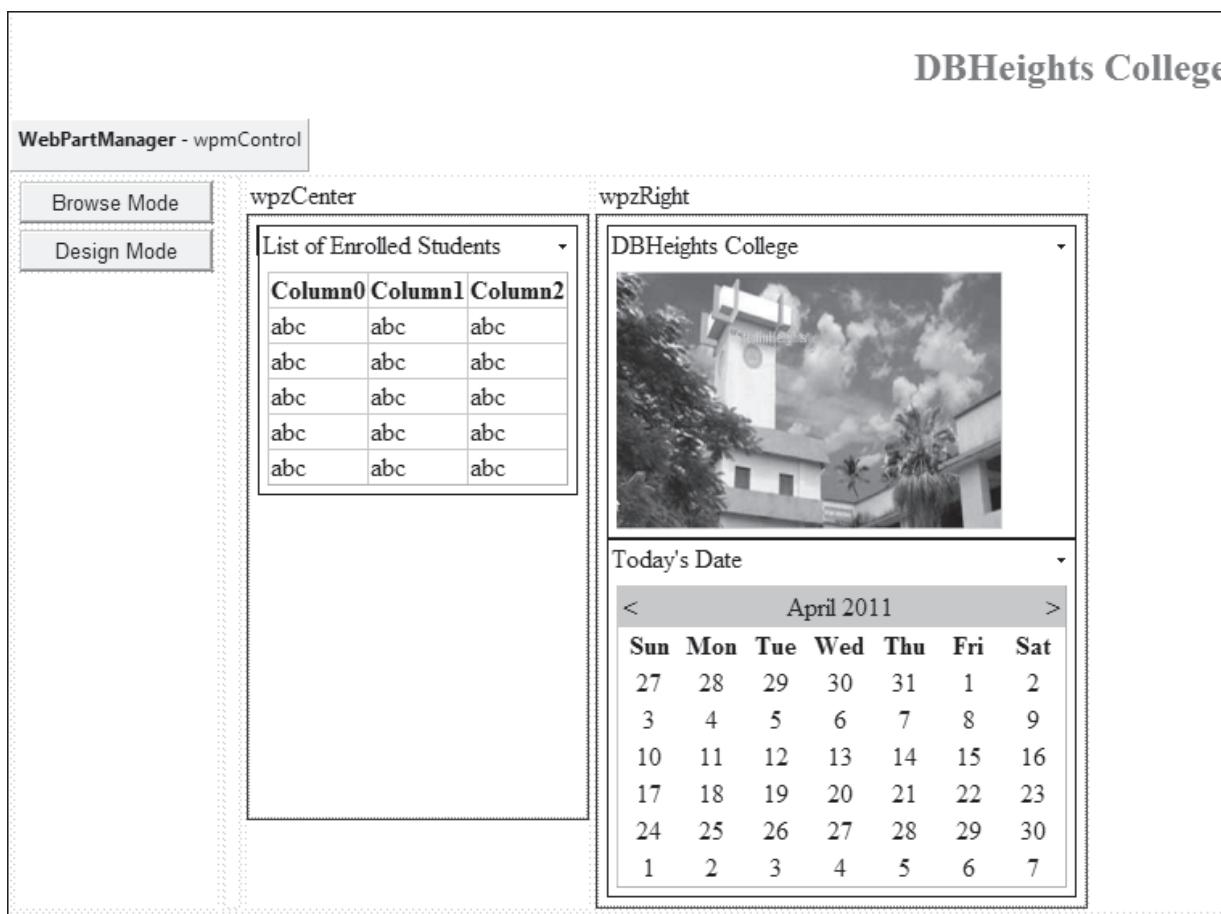


Figure 20.2: User Interface

The user interface consists of two buttons named **btnBrowse** and **btnDesign**, and two WebPartZones

named wpzCenter and wpzRight. The WebPartZone, **wpzCenter**, consists of a GridView control named **gvStudents**, to display the student records. The WebPartZone, **wpzRight**, consists of an Image control and a **Calendar** control with IDs, **imgCollege** and **calDate** respectively.

7. Add the following code for creating the user interface:

```
<table>
  <tr valign="top">
    <td>
      <table>
        <tr>
          <td>
            <asp:Button ID="btnBrowse" runat="server"
              Text="Browse Mode" Width="120px" />
          </td>
        </tr>
        <tr>
          <td>
            <asp:Button ID="btnDesign" runat="server"
              Text="Design Mode" Width="120px" />
          </td>
        </tr>
      </table>
    </td>
    <td>
    </td>
    <td>
      <asp:WebPartZone ID="wpzCenter" runat="server" BorderColor="Blue"
        Height="400px" Padding="6">
        <ZoneTemplate>
          <asp:GridView ID="gvStudents" runat="server" Title="List of Enrolled
            Students">
          </asp:GridView>
      </asp:WebPartZone>
    </td>
  </tr>
</table>
```

```

        </ZoneTemplate>
    </asp:WebPartZone>
</td>
<td>
    <asp:WebPartZone ID="wpzRight" runat="server" BorderColor="Blue"
Height="384px" Padding="6" Width="144px">
        <ZoneTemplate>
            <asp:Image ID="imgCollege" runat="server" Height="160px"
ImageUrl="~/college.jpg" Title="DBHeights College" Width="240px" />
            <asp:Calendar ID="calDate" runat="server" Title="Today's Date"
Height="180px" Width="280px">
                </asp:Calendar>
            </ZoneTemplate>
        </asp:WebPartZone>
    </td>
</tr>
</table>

```

Working with WebPart Controls

The next step is to add code to display data in the corresponding controls.

1. Switch to the **Code Editor** window of the page, **Default.aspx**.
2. To import .NET Framework Data Provider namespace for SQL Server, in the code window, type the code as follows:

```
using System.Data.SqlClient;
```

The **System.Data.SqlClient** namespace is used to import classes that are used to connect to SQL Server.

3. Add the following variables:

```
string strCon;
SqlDataAdapter sqldaStudents;
DataSet dsetStudents;
```

Three variables named **strCon**, **sqldaStudents**, and **dsetStudents** are declared of types **string**, **SqlDataAdapter**, and **DataSet** respectively.

4. To specify the SQL Server connection details, in **Page _ Load** event handler, add the following code:

```

strCon = System.Configuration.ConfigurationManager.ConnectionStrings["SQL
    ConnString"].ToString();
SqlConnection sqlCon = new SqlConnection(strCon);
sqlCon.Open();

```

The code retrieves the SQL connection string from the **web.config** file by using the **ConnectionStrings** property of **ConfigurationManager** class and then, assigns it to the **strCon** variable. Then, an **SqlConnection** instance is created and opens the connection.

- To specify the table details and to create a data set, immediately after **sqlcon.Open()**, add the following code:

```

sqlDaStudents = new SqlDataAdapter("Select * from Student", sqlCon);
dsetStudents = new DataSet();
sqlDaStudents.Fill(dsetStudents);

```

The code creates a **SqlDataAdapter** instance and populates it with the records of the Student table. Then, a new **DataSet** instance is created and fills it with the records of the Student table.

- To bind the data source of the grid view control with the data of **DataSet**, type the following code:

```

gvStudents.DataSource = dsetStudents;
gvStudents.DataBind();

```

The code binds the data source to the **GridView** control.

- Generate the Click event of the button, **btnBrowse**, and add code as follows:

```
wpmControl.DisplayMode = WebPartManager.BrowseDisplayMode;
```

The code assigns the **BrowseDisplayMode** property of the **WebPartManager** class to the **DisplayMode** property of **WebPartManager** control, **wpmControl**. The **BrowseDisplayMode** displays the Web Parts controls and user interface elements in the normal mode. In this mode, you cannot drag and drop the Web Parts. You can see two options: **Minimize** and **Close**. Minimizing a Web Part will still show it, but in minimized state.

- Generate the Click event of the button, **btnDesign**, and add code as follows:

```
wpmControl.DisplayMode = WebPartManager.DesignDisplayMode;
```

The code assigns the **DesignDisplayMode** field of **WebPartManager** class to the **DisplayMode** property of **WebPartManager** control, **wpmControl**. The **DesignDisplayMode** displays the zone user interface elements and enables users to drag and drop Web Parts controls to change the layout of a page.

Adding Database Connection String in web.config file

The next step is to add the database connection string in the **web.config** file.

- Open **web.config** file and add the database connection string as follows:

```
<connectionStrings>
    <add name="SQLConnString" connectionString="Data Source=BIN\
SQLEXPRESS;Integrated Security=true; Initial Catalog=DBHeights;uid=sa;
pwd=password_123" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

These statements specify the connection string details.

Creating the aspnetdb database

The steps to create the aspnetdb database are as follows:

1. Open Visual Studio 2008 Command Prompt.

2. Type `aspnet _ regsql` and press the **Enter** key. This will open the **ASP.NET SQL Server Setup Wizard** as shown in figure 20.3.



Figure 20.3: ASP.NET SQL Server Setup Wizard

3. Click **Next**. The **Select a Setup Option** screen appears.

4. Select the option **Configure SQL Server for application services** as shown in figure 20.4.

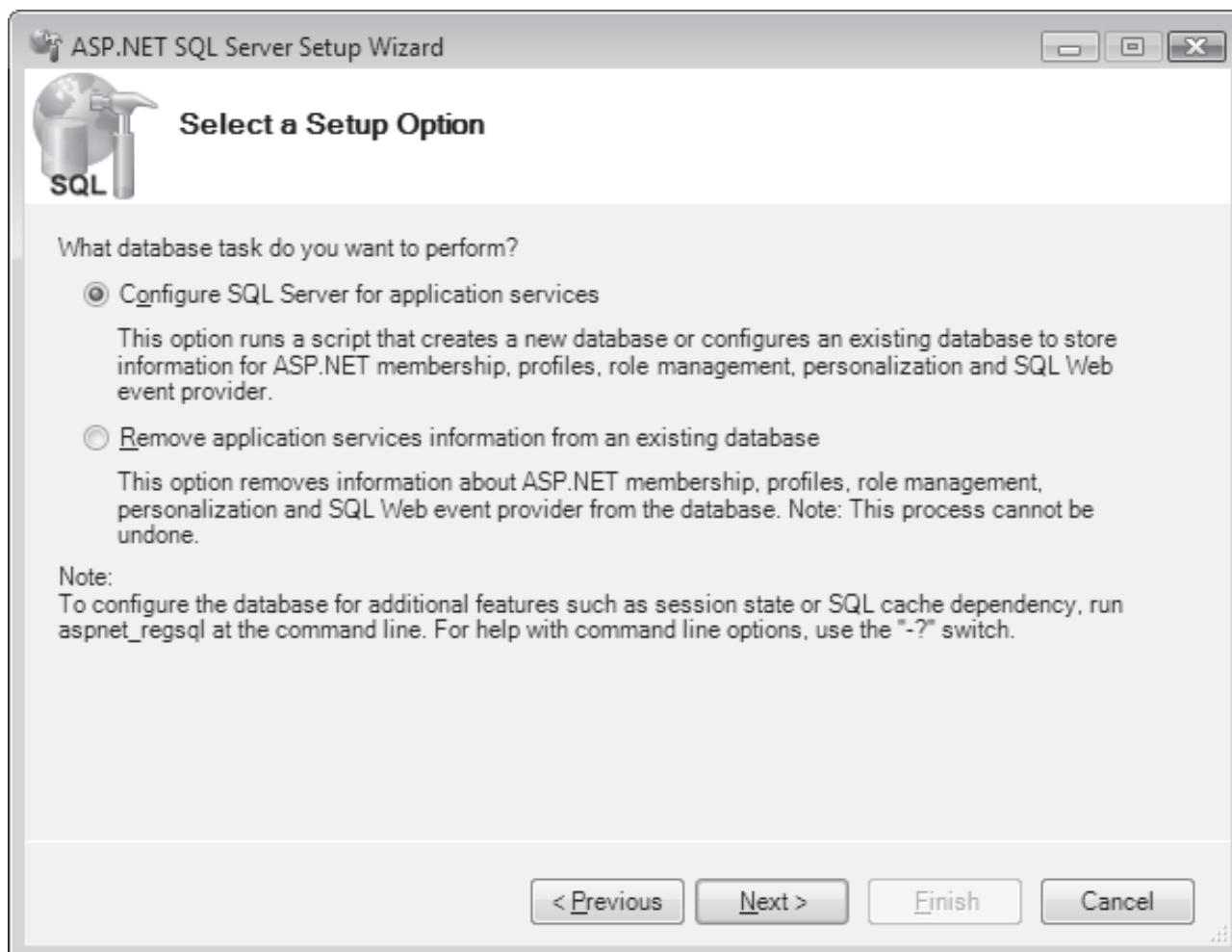


Figure 20.4: Select a Setup Option Screen

This option executes a script to configure the database for managing user profiles, roles, membership, and personalization.

5. Click **Next**. The **Select the Server and Database** screen appears.

6. Enter the SQL Server name and choose the right authentication method. Leave the database field to default as shown in figure 20.5.

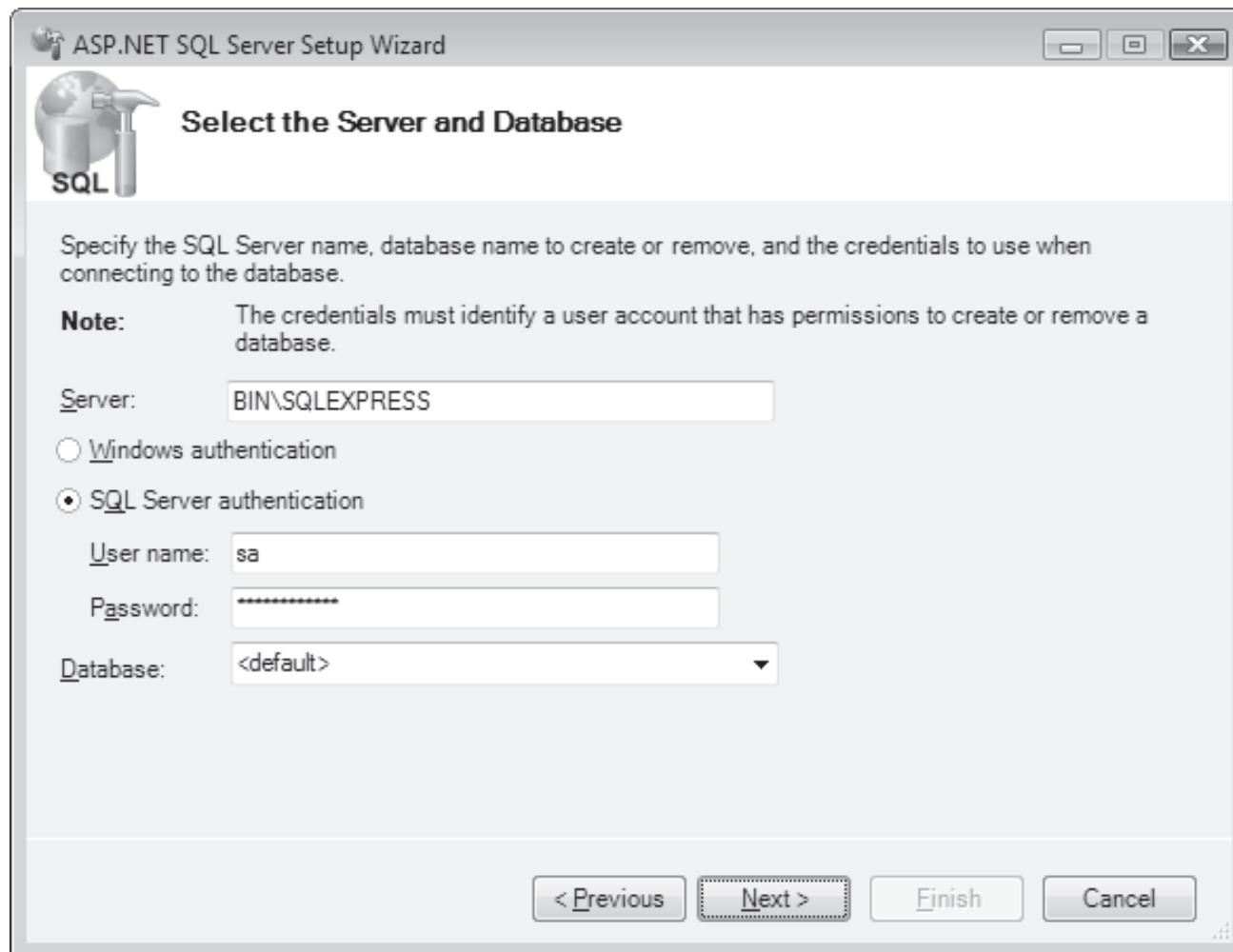


Figure 20.5: Entering the SQL Server Details

7. Click **Next**. The **Confirm Your Settings** screen appears as shown in figure 20.6.

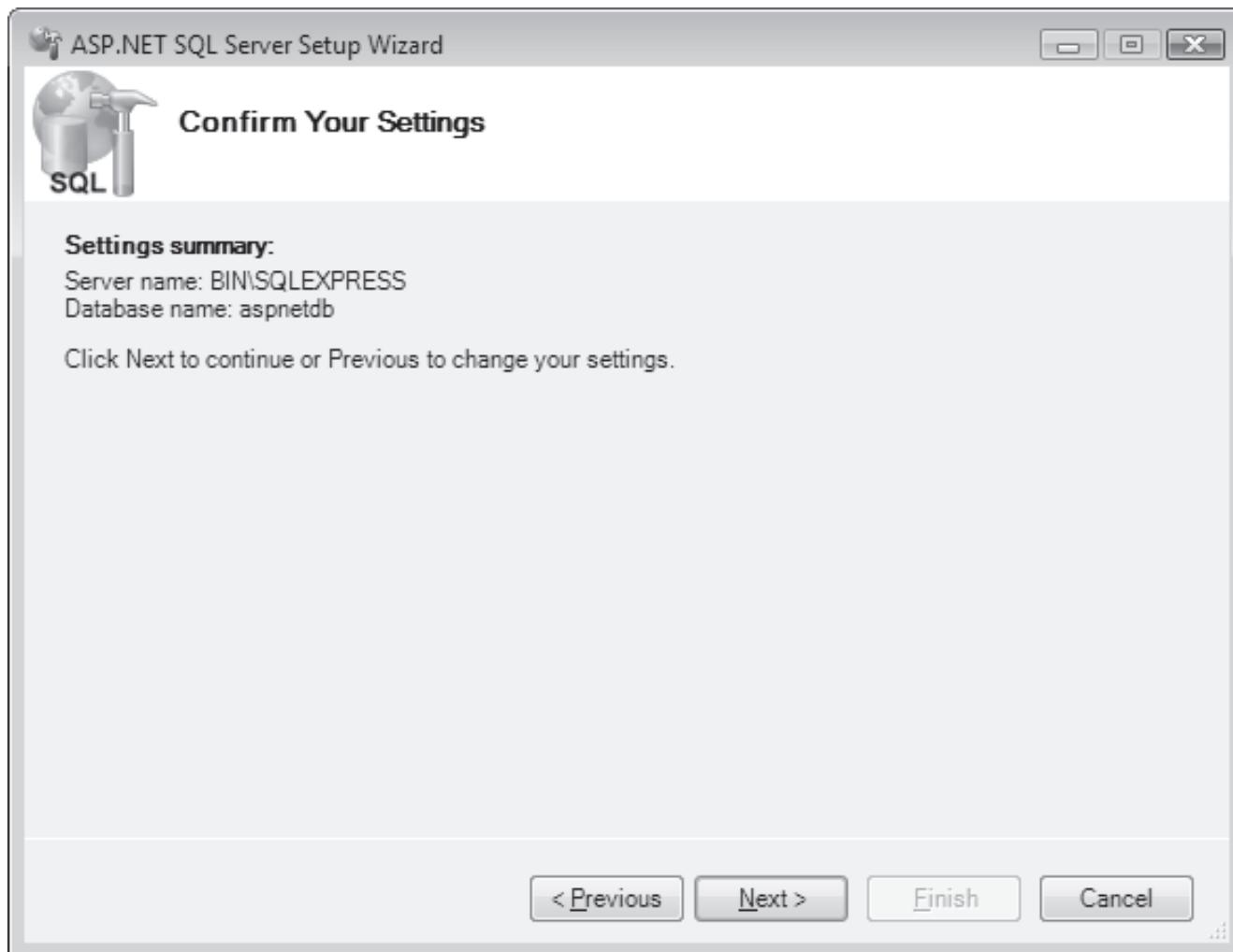


Figure 20.6: Confirming Your Settings Screen

8. Click **Next**. The screen The database has been created or modified appears as shown in figure 20.7.

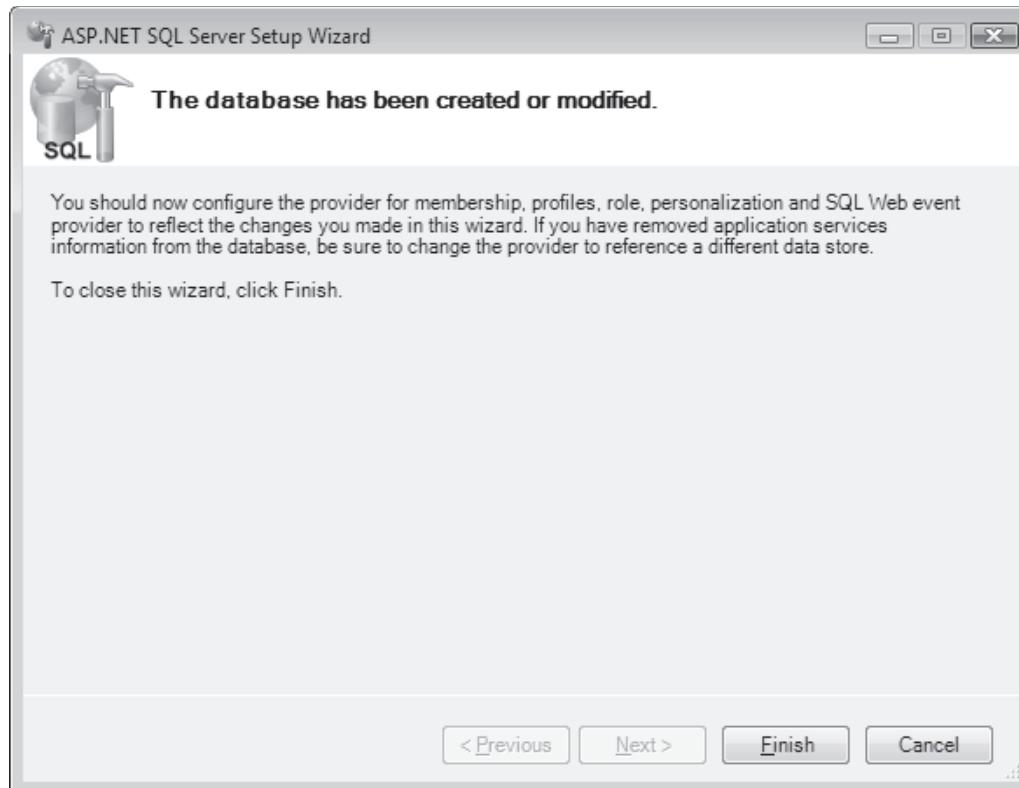


Figure 20.7: Database Created or Modified Screen

9. Click **Finish**. This will create the **aspnetdb** database.

Executing the application

You can perform the following steps to build and execute the application:

1. To compile and build the application, on the **Build** menu, click **Build Solution**. The message **Build succeeded** will be displayed.

2. To execute the application, on the **Debug** menu, click **Start Without Debugging**. The application is executed and **Default.aspx** page is displayed as shown in figure 20.8.

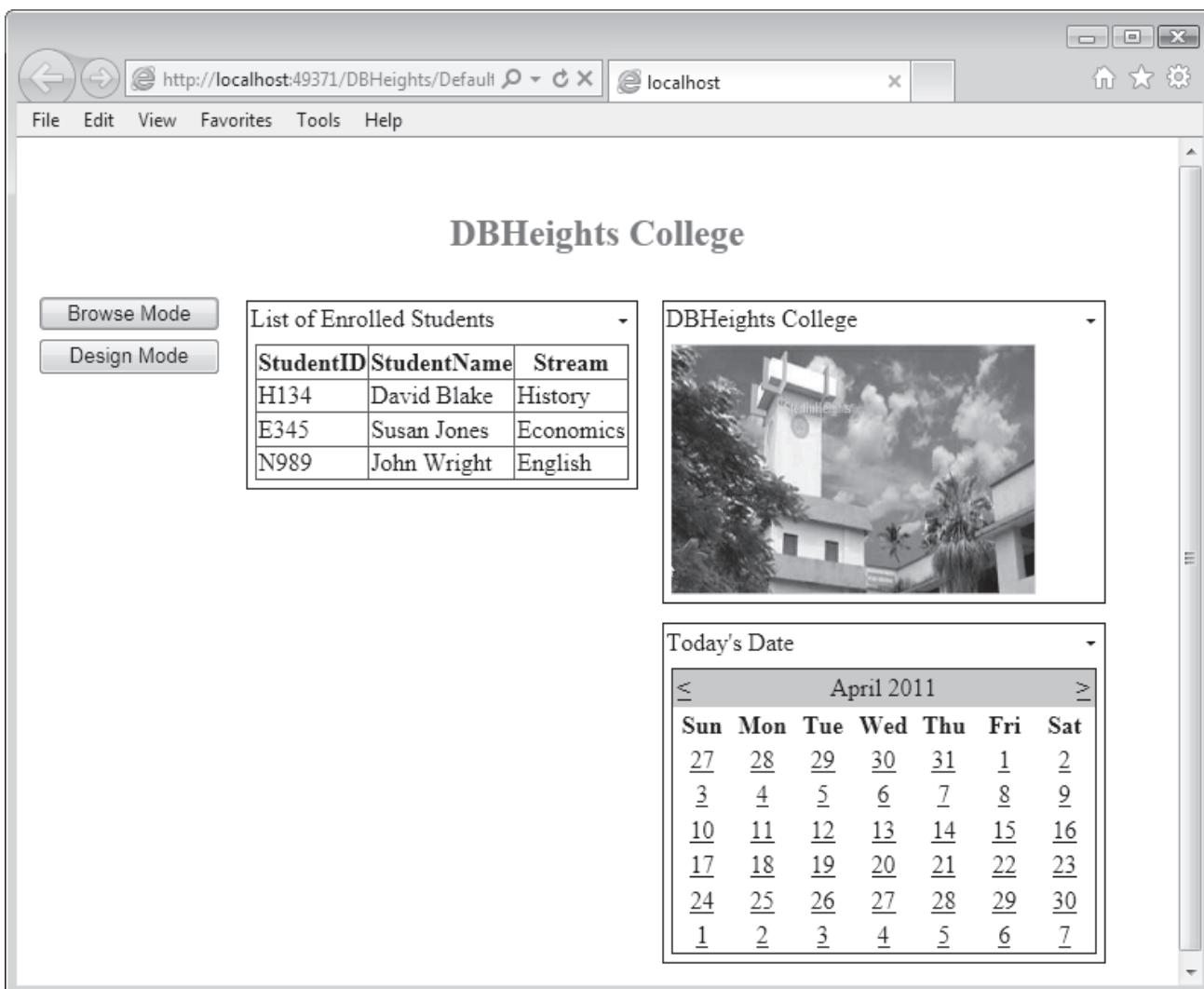


Figure 20.8: Data displayed in Web Parts

Now, the details are displayed in `BrowseDisplayMode`.

3. To change the display to `DesignDisplayMode`, on the Web page, click **Design Mode**. Borders will appear around each `WebPartZone` as shown in figure 20.9.

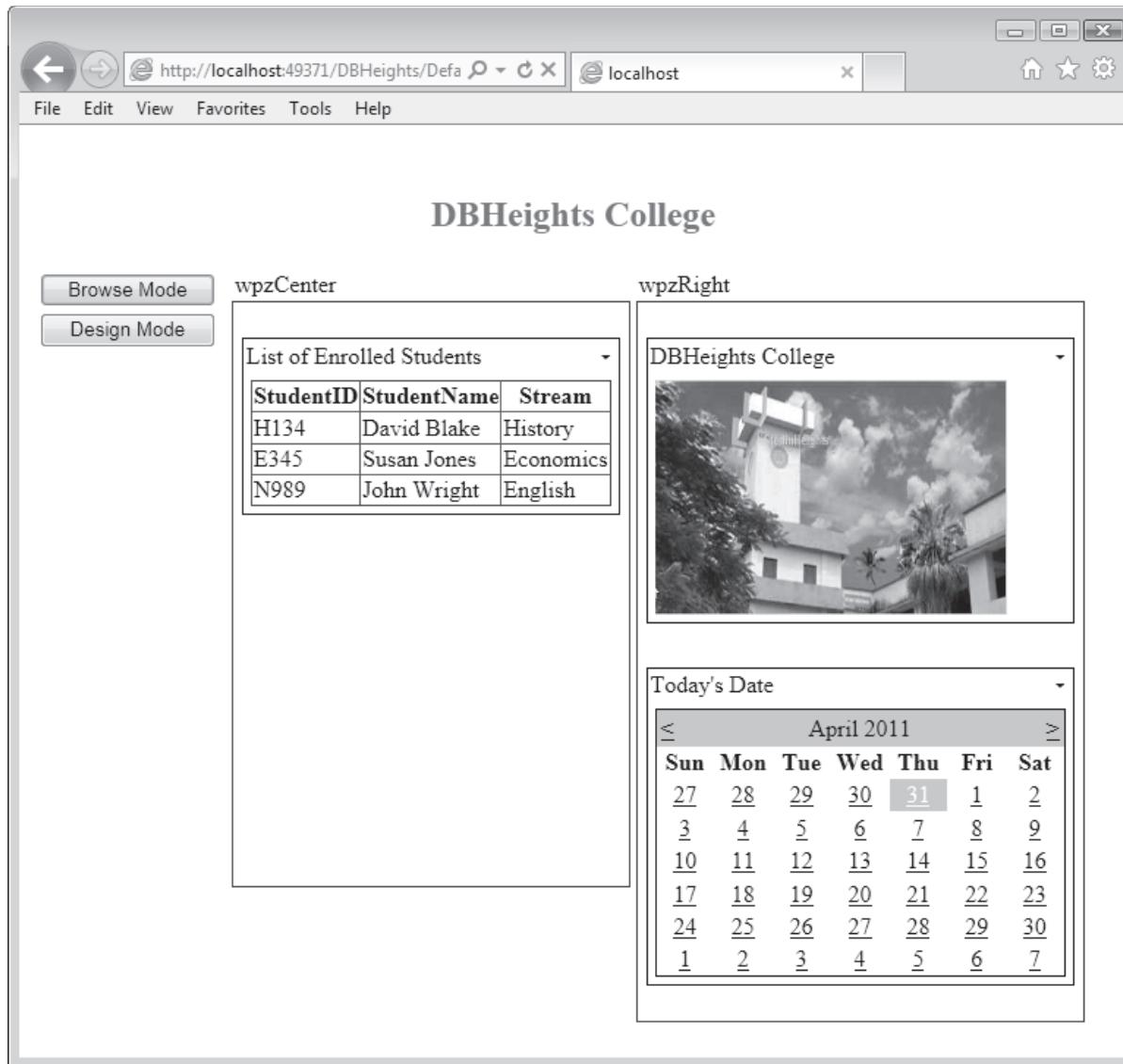


Figure 20.9: Design Mode

4. To drag the Calendar control from the `wpzRight` WebPartZone to the `wpzCenter` WebPartZone, on the Web page, place the mouse at the top of the calendar control, and drag and drop in the `wpzCenter` WebPartZone.

The resulting output is displayed as shown in figure 20.10.

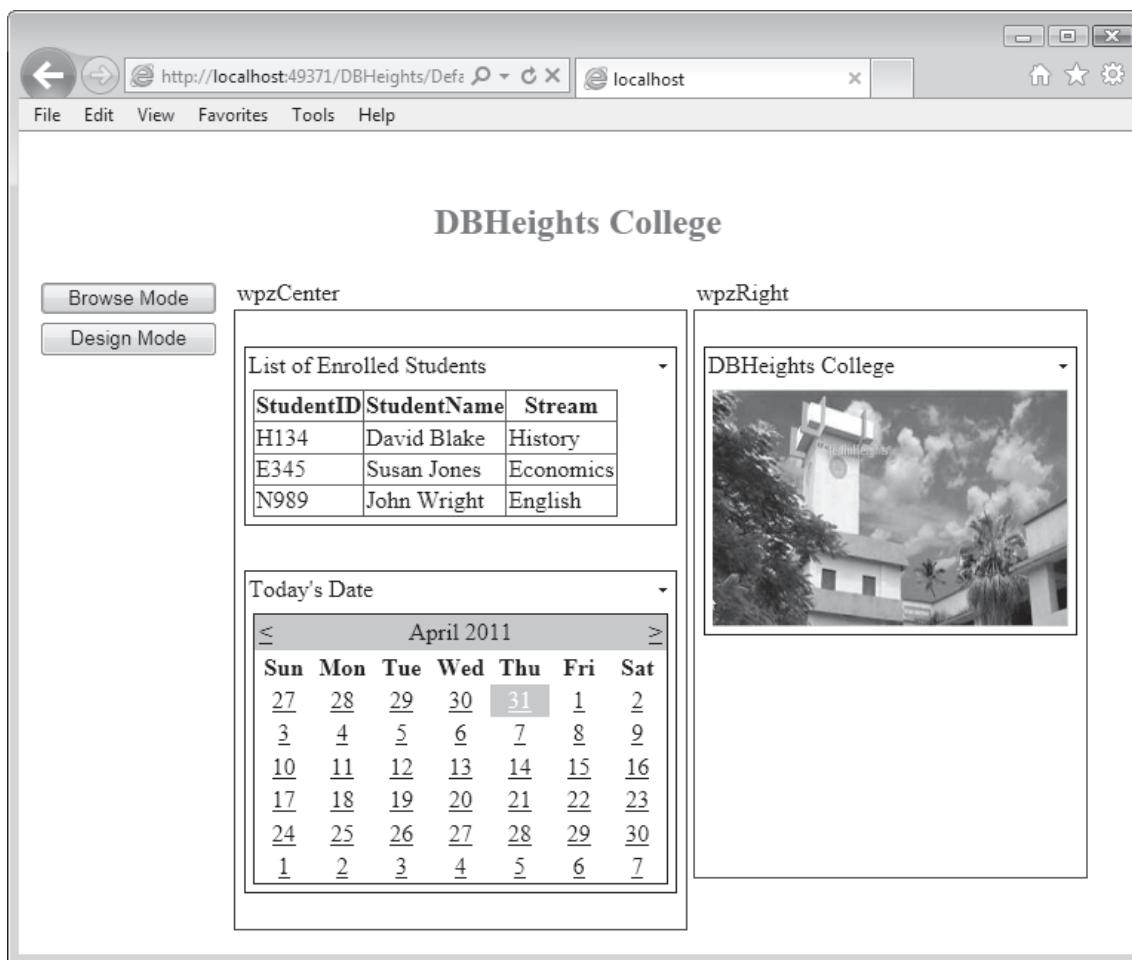


Figure 20.10: Calendar Control dragged and dropped in wpzCenter WebPartZone

5. To display the options, **Minimize** and **Close** of Calendar control, on the Calendar control, click the drop-down arrow next to **Today's Date**.

When the user clicks the drop-down arrow, the options **Minimize** and **Close**, will appear as shown in figure 20.11.

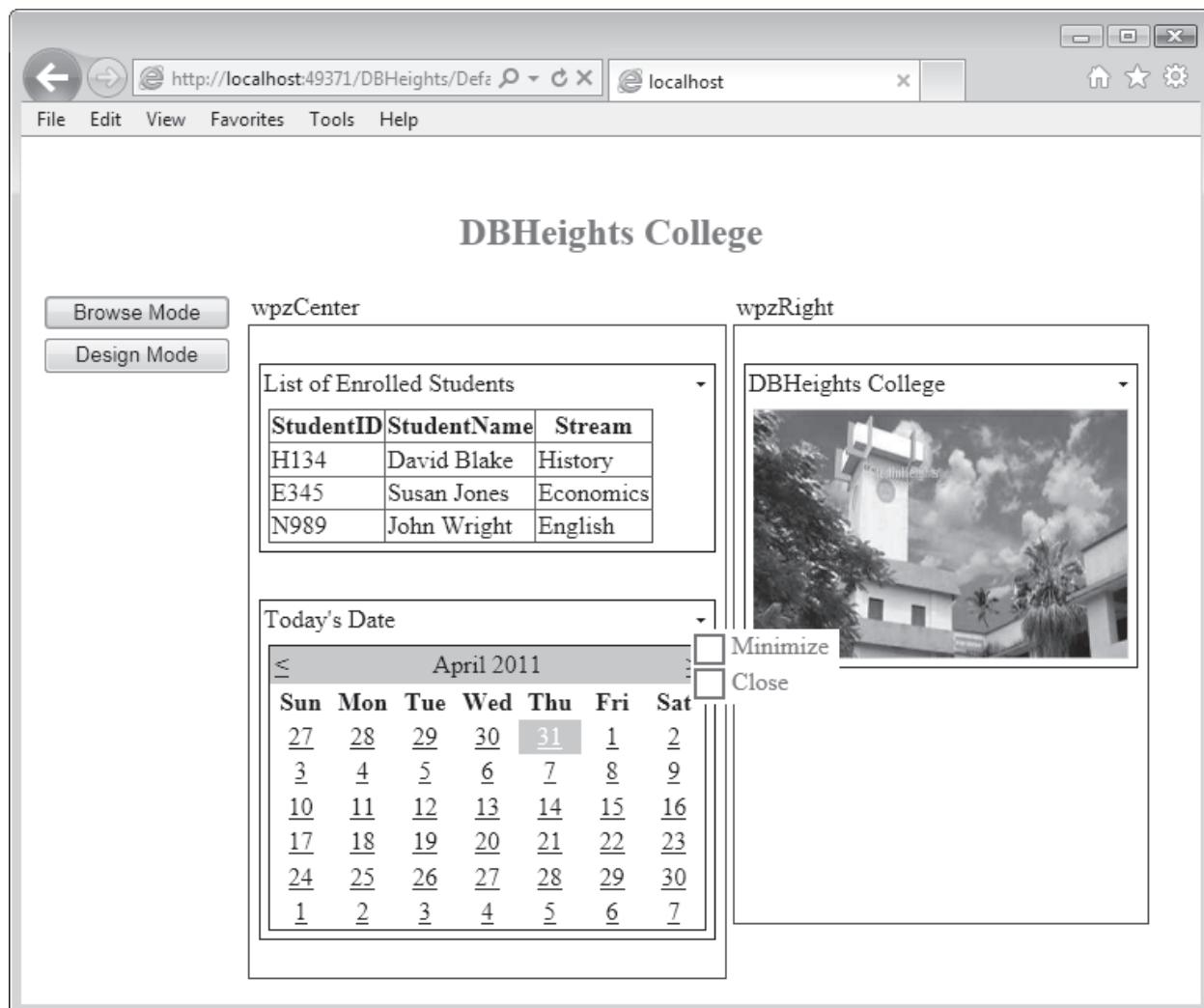


Figure 20.11: Minimize and Close Options

6. To minimize the Calendar control, click **Minimize**. The output is displayed as shown in figure 20.12.

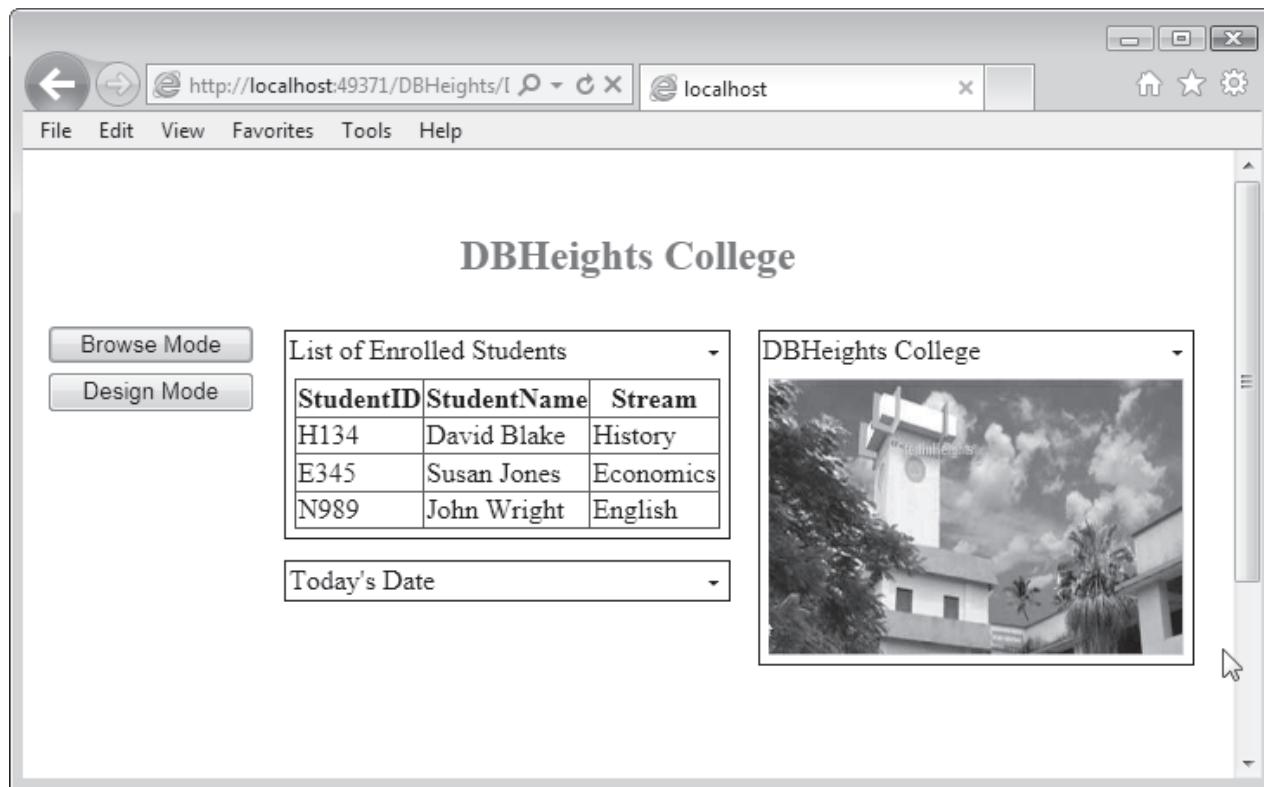


Figure 20.12: Calendar Control Minimized

If you again click the drop-down arrow on the Calendar control, two options, **Restore** and **Close**, will be displayed. You can restore the control to its original form by clicking **Restore**.

7. To display the options, **Restore** and **Close** of Calendar control, on the Calendar control, click the drop-down arrow next to **Today's Date**.

8. To close the Calendar control, click **Close**. The calendar is closed and output is displayed as shown in figure 20.13.

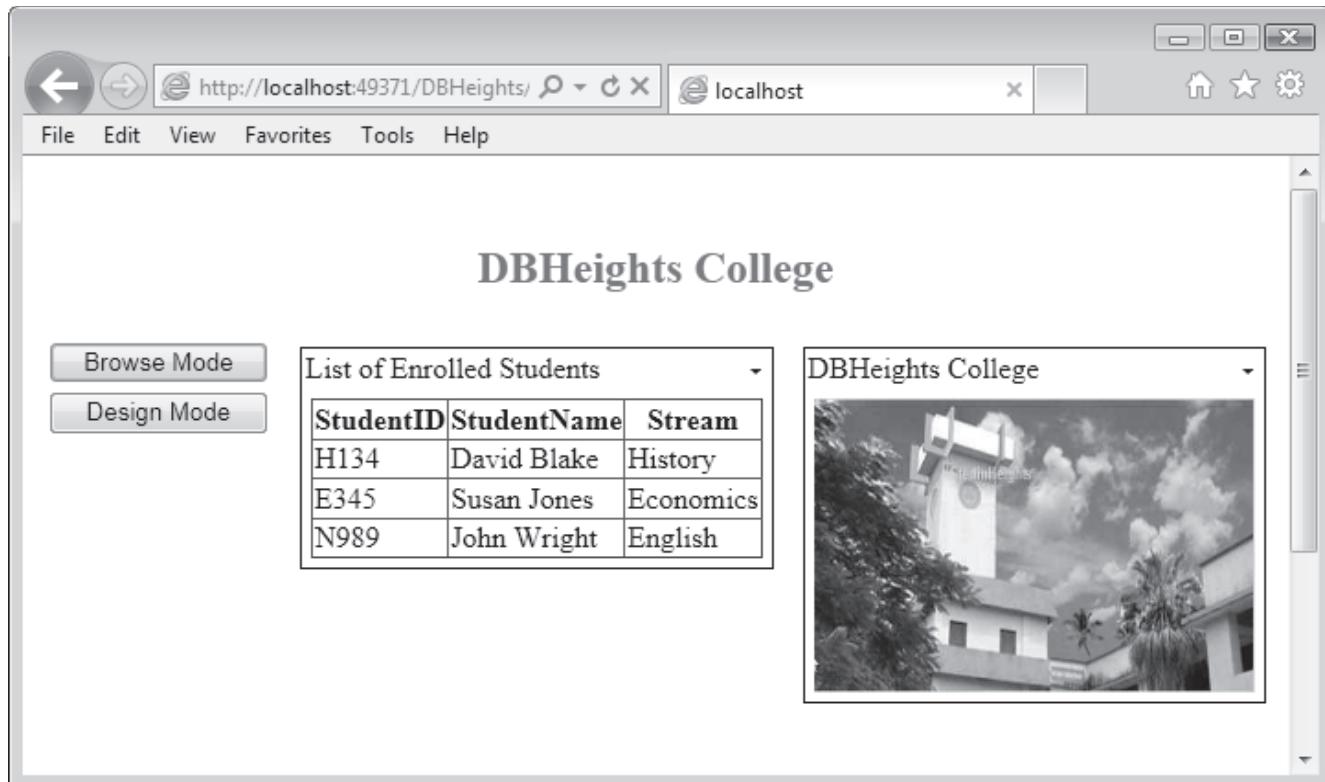


Figure 20.13: Calendar Control Closed

20.2 Part II - 60 Minutes

ElectraToys is an electronic toys manufacturing company in USA. The company wants to start selling its toys online. However, with constant increase in the number of toys, the created online application needs to display all the available items on all Web pages.

As part of the development team, you need to perform the following task through ASP.NET 3.5 and Visual Studio 2008:

Create a Master page containing a menu that can be included on all the Web pages and the headers of the application.

Additionally, the team leader for this project has also assigned you the following task:

Create nested Master Pages so that the Web pages for the sponsors display different headers.

Hints:

Create a Master page with the menus about the toys and company details

Create nested Master pages to display the sponsor details

Store all the toy details in a database table

Create all the pages related to toys

20.3 Do It Yourself

PizzaHome is a leading pizza delivery chain in USA. They want to develop a Web-based solution to showcase the ongoing Pizza festival where they will be serving pizzas from across the world. In the application, they want to maintain separate information for Chinese pizzas and Italian pizzas. They want to show different categories of pizza by dragging and dropping the pizzas into various sections. Also they want to create master page to show the main pizza items and the festival details.

Assuming that you are a part of the developing team working on the application logic, perform the specified operations in ASP.NET 3.5.

Test the application using Visual Studio IDE 2008.

Module - 21

Working with Stored Procedures and LINQ

Welcome to the module, **Working with Stored Procedures and LINQ**. This module defines and describes stored procedures and their use in ASP.NET. The module also defines LINQ, describes the LINQ to SQL model, and explores the DataContext class and its use. Finally, the module describes use of LINQ with stored procedures.

In this module, you will learn to:

- Explain stored procedures
- Describe the steps to create and call a stored procedure in ASP.NET
- Describe parameterized stored procedures in ASP.NET
- Define and describe LINQ
- Describe LINQ to SQL
- Describe the DataContext class and its use
- Explain use of LINQ with stored procedures

Web Development

http://www



21.1 Introduction

ASP.NET Web-based applications created for enterprises and businesses often make use of data manipulation for their day-to-day transactions. Accessing and manipulating data directly from a Web Form to or from a database can lead to security risks and inefficient use of resources. One alternative to direct database access from Web applications is to use stored procedures. Using stored procedures in ASP.NET applications can improve the efficiency and security of database access.

21.2 Overview of Stored Procedures

A stored procedure is a set of precompiled database commands that is stored on the database server. A stored procedure can be used in .NET applications to access and manipulate data in the database.

Instead of writing your own SQL statements each time you want to perform an operation, if you use existing stored procedures or share stored procedures across your ASP.NET applications, it will help you reduce the Web Form code. As the number of round trips to the database would be reduced, stored procedures can also help to restrain the network bandwidth.

Thus, the key benefits offered by stored procedures are efficiency and secure data access.

Table 21.1 lists the different types of stored procedures.

Type	Description
Stored procedures that return records	Used to find particular records, sort and filter those records, and then, return the result to a data set or to a list-bound control. Are based on SQL Select statements
Stored procedures that return values, also known as scalar stored procedures	Used to execute one or more database commands or functions to return a single value
Action stored procedures	Used to perform an operation in the database. May include updates, edits, or deletion of data and do not return a record or a value

Table 21.1: Types of Stored Procedures

There are many benefits of using stored procedures in your Web applications. Some of them are as follows:

- **Better Database Security:** A stored procedure can provide enhanced security for a database by restricting direct access to the data from Web Forms. Only the stored procedures that are trusted or local can access the database directly from Web applications. This reduces the risks on the database.
- **Faster Execution:** If some database operations need a large set of Transact-SQL statements or they are performed repeatedly, each time you call them in a Web application will lead to a reduction in speed and performance. Instead, if you use stored procedures, it will be faster. Stored

procedures are precompiled after they are created, so the application need not compile the SQL statements every time the database operation is to be performed. Also, an application can make use of in-memory versions of a stored procedure after the procedure is executed for the first time. All these lead to faster execution.

- **Modular Programming:** Stored procedures implement modular programming. You just create a stored procedure once, and can reuse it any number of times from multiple applications.

21.3 Creating and Calling Stored Procedures

Visual Studio 2008 enables you to create and call stored procedures in ASP.NET Web applications.

21.3.1 Creating a Stored Procedure

You can create stored procedures either in SQL Server Management Studio or in Visual Studio 2008 IDE through the Server Explorer window. Here, the second approach will be used.

Consider that you have a database named Library in SQL Server. Create three tables named BookCategories, Books, and IssueReturn having structure as shown in figure 21.1.

BookCategories:

	Column Name	Data Type	Allow Nulls
CategoryID	nchar(10)	<input type="checkbox"/>	
Category	nchar(10)	<input type="checkbox"/>	

Books:

	Column Name	Data Type	Allow Nulls
BookId	nchar(10)	<input type="checkbox"/>	
Title	nchar(50)	<input checked="" type="checkbox"/>	
Author	nchar(50)	<input checked="" type="checkbox"/>	
CategoryID	nchar(10)	<input type="checkbox"/>	

IssueReturn:

	Column Name	Data Type	Allow Nulls
MemberID	nchar(10)	<input type="checkbox"/>	
BookID	nchar(10)	<input type="checkbox"/>	
IssueDate	date	<input checked="" type="checkbox"/>	
ReturnDate	date	<input checked="" type="checkbox"/>	
Fine	int	<input checked="" type="checkbox"/>	

Figure 21.1: Tables in Library Database

Ensure that the table **Books** has a foreign key for **CategoryID**, mapping to **CategoryID** in **BookCategories**. Insert suitable records in all the three tables. The following steps are used to create

a stored procedure that will display the records from the **BookCategories** table:

1. Create a new ASP.NET Web site named **LibraryApplication**.
2. In the **Server Explorer** in Visual Studio 2008, right-click **Data Connections**. The context menu is displayed as shown in figure 21.2.

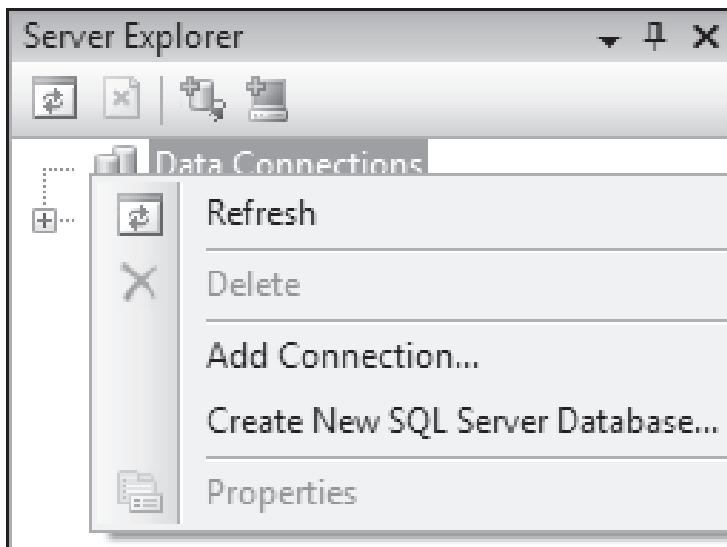


Figure 21.2: Adding a New Connection

3. Click **Add Connection**. The **Add Connection** dialog box is displayed.
4. In the **Add Connection** dialog box, type the SQL Server name, user name, and password to access the server.
5. Type **Library** in the **Select or enter a database name** drop-down. The new connection to Library database is created and displayed under **Data Connections** in **Server Explorer**.
6. In **Server Explorer**, under **Data Connections**, expand the data connection to display **Tables**, **Views**, and **Stored Procedures**.
7. Right-click **Stored Procedures** and select **Add New Stored Procedure** from the context menu.
8. Replace the default code in the new stored procedure with the code shown in the following Code Snippet. This will create a stored procedure that lists all the records from **BookCategories**.

Code Snippet:

```
CREATE Procedure BookCategoryList
AS
SELECT * FROM BookCategories
```

Close the window and specify a name for the stored procedure as **BookCategoryList**.

21.3.2 Calling a Stored Procedure through a DataAdapter object

In order to call a stored procedure, you must identify the procedure name and the available parameters, if any. You can then call the procedure, pass any input parameters that are required to process your request, and handle the output parameters.

The following steps are used in general to call a stored procedure:

1. Identify the type and name of the stored procedure.
2. Choose the method that you want to use to call the stored procedure. You can use a `DataAdapter` object or a `DataReader` object to call all three types of stored procedures that are listed in table 21.1.
3. Set the `Connection`, `CommandText`, and `CommandType` properties of the `DataAdapter`. The `CommandText` property will be assigned the name of the stored procedure, while the `CommandType` property will be set to `CommandType.StoredProcedure`.

Consider that you want to call the `BookCategoryList` stored procedure to return records and assign them to a `GridView` control in a Web application. Add a `GridView` control named `gvwCategories` to the default Web Form in `LibraryApplication`. Rename the default form to `Library.aspx`.

Specify the connection details through the `web.config` file as shown in the following Code Snippet:

Code Snippet:

```
<connectionStrings>
<add name="LibConnString" connectionString="Data Source =
MM\SQLEXPRESS;Initial Catalog=Library; Integrated Security=True"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

A connection string named `LibConnString` is added to the Web application as a result of the code.

The following code snippet uses `SqlConnection` and `SqlDataAdapter` objects to call the `BookCategoryList` stored procedure:

Code Snippet:

```
SqlConnection objConn = new SqlConnection();
objConn.ConnectionString = ConfigurationManager.ConnectionStrings
["LibConnString"].ConnectionString;
SqlDataAdapter objAdapter = new SqlDataAdapter("BookCategoryList",
objConn);
objAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;
DataSet objDs = new DataSet();
```

```
objAdapter.Fill(objDs, "Categories");
gvwCategories.DataSource = objDs.Tables["Categories"];
gvwCategories.DataBind();
```

The connection is retrieved from the **web.config** file through the `ConfigurationManager` class. A new `SqlDataAdapter` object is created. The constructor of the `SqlDataAdapter` object takes two parameters, the `CommandText` property, which is set to the name of the stored procedure, and the connection object. The `Fill` method of the `SqlDataAdapter` object fills a `DataTable` object in the `DataSet`, `objDs`, with the returned records of the stored procedure and then, populates the data into the `GridView`.

The output will be as shown in figure 21.3.

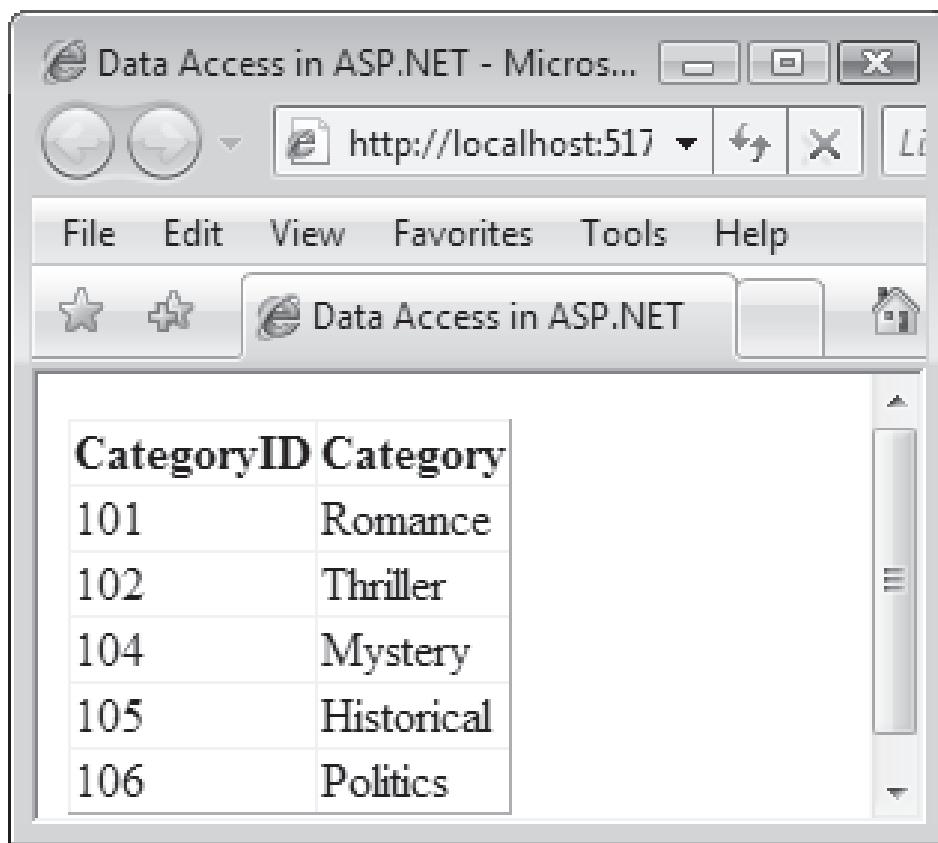


Figure 21.3: Output

21.3.3 Calling a Stored Procedure by Using a DataReader

You can also call a stored procedure by using a `DataReader` object. The following code snippet shows how to do this:

Code Snippet:

```
SqlConnection objConn = new SqlConnection();
objConn.ConnectionString = ConfigurationManager.ConnectionStrings
    ["LibConnString"].ConnectionString;
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConn;
objConn.Open();
objCommand.CommandText = "BookCategoryList";
objCommand.CommandType = CommandType.StoredProcedure;
SqlDataReader objReader = objCommand.ExecuteReader();
gvwCategories.DataSource = objReader;
gvwCategories.DataBind();
objReader.Close();
```

The `ExecuteReader()` method is called on the `Command` object to execute the stored procedure. The output of this code is similar to figure 21.3 shown earlier.

21.4 Parameters in Stored Procedures

Assume that you are creating an application that deals with data from the database. The user wants to display results based on certain conditions. You can display data based on these conditions by taking input from the user and using that input to form an SQL query. For example, you want a query to display titles based on author name.

The query assigned with the `SqlCommand` object is a string. You can create a query as shown in the following Code Snippet:

Code Snippet:

```
SqlCommand objCommand = new SqlCommand("SELECT Title FROM Books WHERE Author = '" +
authorName + "'");
```

When you use string concatenation while forming a query as shown in the code, the database will have to parse the command, compile it, and then, form an execution plan for each value of `authorName`. Sometimes, a hacker can replace the string value with some malicious data. This is termed as SQL injection attack and can be quite damaging.

The solution for this is to use parameters in your query or stored procedures. Anything that the parameter

contains will be treated as field data, not as part of the SQL statement. This makes the application much more secure.

A query parameter in .NET is a kind of variable that is used to pass and return values between your application and database.

The data type for the parameter is assigned using the values defined in the `System.Data.SqlDbType` enumeration. You pass parameter values to SQL statements and stored procedures when you want to change the criteria of the queries instantly.

Parameterized queries or stored procedures help you with prevention of injection attacks and they also help with performance.

Parameters can be of three types namely `Input`, `Output`, and `InputOutput`. `Input` parameters are used when the data is sent to the database, while `Output` is used to retrieve information from the database. `InputOutput` parameters are used to send as well as receive data when executing a command.

The type of a parameter can be specified by using the `Direction` property which takes its value from the `ParameterDirection` enumeration. The possible values of `Direction` property are `Input`, `Output`, `InputOutput`, or `ReturnValue`.

Note - The default value of a parameter is `Input`.

Names of the parameters added to the `Parameters` collection of the `SqlCommand` object must match the names of the parameters present in the stored procedure. Though the order of the parameters is flexible, it is recommended to list the parameters in the order that they are defined in the stored procedure.

After you identify the parameters that a stored procedure supports, you must then, add the parameters to the `Parameters` collection of the `Command` object.

21.4.1 Creating Input Parameters

To create a parameter, create a new `SqlParameter` object with the same name and data type of the parameter that is specified in the definition of the stored procedure. You must set the `Direction` property of the new parameter to indicate how the stored procedure uses the parameter. If the parameter is an input parameter, set the `Value` property to specify the data to send to the SQL Server database.

For example, consider that you want to display books based on category. You write the code for the stored procedure as shown in the following code snippet:

Code Snippet:

```
CREATE PROCEDURE BooksByCategory (@CategoryID nchar(10) )
AS
    SELECT Title FROM Books
    WHERE CategoryID=@CategoryID
GO
```

In this code, the **BooksByCategory** stored procedure takes one input parameter. This parameter is of type nchar(10) and is named **@CategoryID**.

The following code snippet shows how to create an input parameter and pass it to the stored procedure:

Code Snippet:

```
SqlConnection objConn = new SqlConnection();
objConn.ConnectionString = ConfigurationManager.ConnectionStrings
    ["LibConnString"].ConnectionString;
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConn;
objConn.Open();
objCommand.CommandType = CommandType.StoredProcedure;
objCommand.CommandText = "BooksByCategory";

// Create a SqlParameter instance and set its properties
SqlParameter objParam = new SqlParameter("@CategoryID", SqlDbType.
    NChar, 10);
objParam.Direction = ParameterDirection.Input;
objParam.Value = "101";

// Add the parameter to the Parameters collection of the Command object
objCommand.Parameters.Add(objParam);
SqlDataReader objReader = objCommand.ExecuteReader();
gvwCategories.DataSource = objReader;
gvwCategories.DataBind();
objReader.Close();
```

An **SqlParameter** instance is created and its properties such as **Direction** and **Value** are set in the code. Then, this parameter is added to the **Parameters** collection of the **Command** object. When the stored procedure is invoked as a result of **ExecuteReader()** method, the input parameter will be passed to the stored procedure.

Instead of hard-coding the parameter value as shown in the code, you can accept its value from a text box and store it in the parameter. It is recommended that you validate the contents of the text box before you

send the user input to the stored procedure.

After you finish with the `SqlDataReader`, call the `Close` method of the `SqlDataReader` before you reuse the `SqlConnection` object.

21.4.2 Accessing Output Parameters

Once the stored procedure has finished executing, you can read the returned value from a stored procedure by accessing the value of the output parameter in the `Parameters` collection.

The following code snippet demonstrates how to use an output parameter while creating a stored procedure. The stored procedure retrieves the maximum value of fine paid by a specific member of the library.

Code Snippet:

```
CREATE Procedure DisplayFine (@MemberID nchar(10) , @MaxFine int OUTPUT)
AS
    SELECT @MaxFine = Max(Fine)      FROM IssueReturn
    WHERE MemberID = @MemberID
```

The following code snippet shows how to call this stored procedure and retrieve the output value returned by the procedure.

Code Snippet:

```

...
objCommand.CommandType = CommandType.StoredProcedure;
objCommand.CommandText = "DisplayFine";
// Create and add input parameter.
SqlParameter objParam = new SqlParameter("@MemberID",
    SqlDbType.NChar, 10);
objParam.Direction = ParameterDirection.Input;
objParam.Value = "1501";
objCommand.Parameters.Add(objParam);

// Create and add output parameter.
SqlParameter objParam2 = new SqlParameter("@MaxFine", SqlDbType.Int);
objParam2.Direction = ParameterDirection.Output;
objCommand.Parameters.Add(objParam2);
// Run the storedprocedure by calling ExecuteScalar() method.
objCommand.ExecuteScalar();
// Retrieve the output value.
int fine = Convert.ToInt16(objCommand.Parameters["@MaxFine"].Value);

Response.Write("The maximum fine paid by Member with ID " + objParam.Value +
    " is :" + fine);

```

In this code snippet, two parameters are created, one of type `Input` and one of type `Output`. The return value from the stored procedure will be stored in the output parameter, which can then, be retrieved using the `Value` property. The stored procedure returns a scalar value, which is the maximum value of fine amount paid by a specific member. Instead of using the `DataAdapter` object here, you can use `ExecuteScalar()` method on the `Command` object to execute the stored procedure. The input parameter is `@MemberID` and the output parameter named `@MaxFine`. They are added to the `Parameters` collection of the `SqlCommand` object. `ExecuteScalar()` is used to call the stored procedure.

The output of the code is shown in figure 21.4.

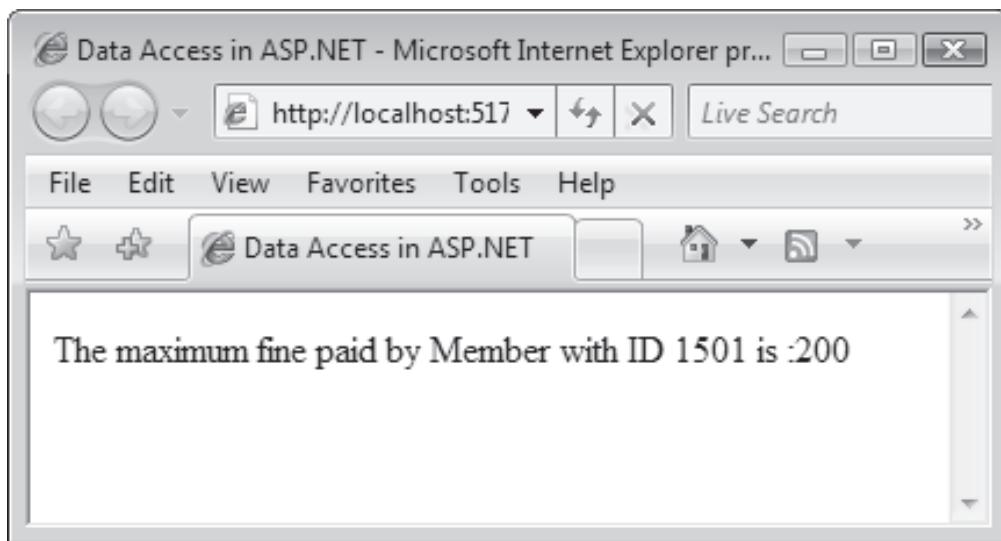


Figure 21.4: Output of the Code

Note - You can also access the return value of a stored procedure by using the `ExecuteReader()` method of a `Command` object.

21.5 LINQ

Applications often need to store, retrieve, and manipulate data in some form or another. Data may not always be in the same format. While some applications work with data in relational data sources, some store and manage XML data. There are also applications that may use data from in-memory collections. Retrieving and maintaining data requires creation of queries and statements. For each kind of data source, a developer may have to use a different approach. This is cumbersome and time consuming.

Another issue is that the Visual Studio environment has no provision to check for any syntax errors present in SQL queries written in C#. If any such errors exist or the query is malformed, it would be detected only at runtime.

To overcome all these issues, Microsoft introduced a new technology, Language-Integrated Query, LINQ. LINQ is a set of features that adds query capabilities to .NET languages such as C#. LINQ enables you to query data from diverse data sources in an easy manner. The only condition is that these data sources must be LINQ-compatible, which means they must be supported by LINQ.

LINQ can be used with SQL, XML files, objects (such as C# arrays and collections), and ADO.NET DataSet objects.

The various LINQ technologies are listed in table 21.2.

Type	Description
LINQ to DataSet	Used to query data cached in a DataSet object.
LINQ to Objects	Used to query in-memory data such as lists.
LINQ to SQL	Used to query data in relational databases.
LINQ to XML	Used to query data stored in XML documents.
LINQ to ADO.NET	Used to query data in relational databases.
LINQ to Entities	Used to query data stored in XML documents.

Table 21.2: LINQ technologies

Among these, one of the most commonly used technologies is LINQ to SQL.

21.6 LINQ to SQL

LINQ to SQL allows you to query and manipulate data easily from SQL Server databases in a .NET language of your choice, such as C#.

Table 21.3 lists the main elements in the LINQ to SQL object model mapped against the relationship to elements in the relational data model.

LINQ To SQL Object Model	Relational Data Model
Entity class	Table
Class member	Column
Association	Foreign-key relationship
Method	Stored procedure or function

Table 21.3: Elements in the LINQ to SQL object model

In LINQ to SQL, the `DataContext` class is the means by which objects are retrieved from the database and changes are sent back. The `DataContext` object converts your requests for objects into SQL queries against the database and then, converts the results into objects.

Visual Studio 2008 IDE includes an Object Relational (O/R) Designer which is a visual design tool to create object models in an application. These object models map to objects in a database. In the O/R Designer, the `DataContext` class acts as a link between a SQL Server database and the LINQ to SQL entity classes mapped to that database.

The step-by-step procedure to query a table using LINQ to SQL is as follows:

1. Create a new ASP.NET Web site named **LibraryApplication**.
2. Add a `ListBox` to the form and set its `Name` property to `lstBooks`.
3. Select **Website → Add New Item**. In the **Add New Item** dialog box that is displayed, click the **LINQ to SQL Classes** template. Enter a name, `Books`, for the **LINQ to SQL Classes** file, and then, click **Add**.

If you get a message asking if you want to create the **App_Code** folder and to store the **LINQ to SQL Classes** file in that folder as shown in figure 21.5, click **Yes**.

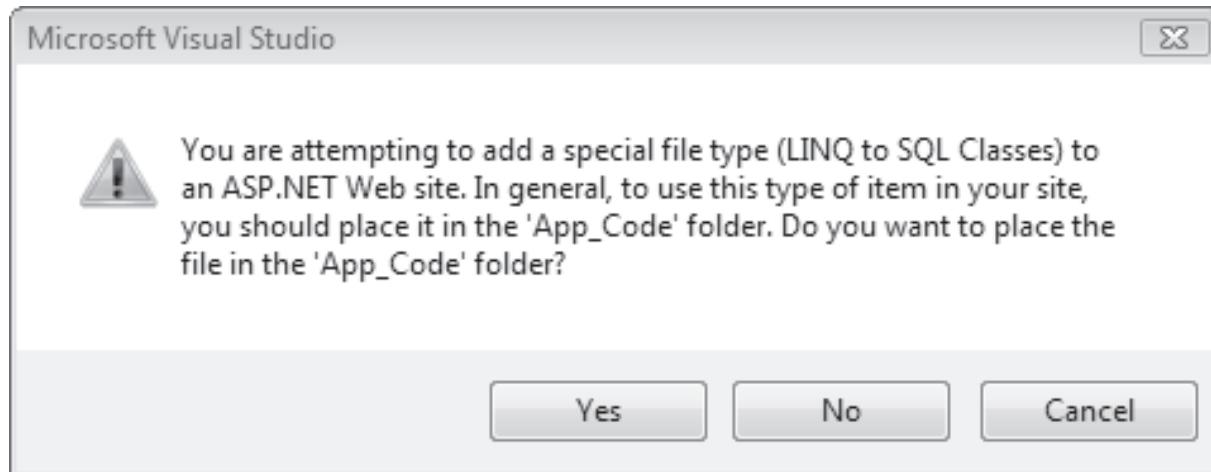


Figure 21.5: Confirmation to Create LINQ to SQL Classes file in App_Code Folder

This will result in a file, **Books.dbml**. When you save the LINQ to SQL Classes file that contains the objects that you want to model in your SQL Server database, Visual Studio 2008 creates an entity class for each database table in the model. In addition, Visual Studio 2008 creates a `DataContext` class. Hence, **Books.dbml** will now contain the entity classes, associations, and `DataContext` methods modeling the relational database items. Use the **Class View** window in Visual Studio 2008 to see the `DataContext` class and the entity classes.

Drag database items such as tables from **Server Explorer** or **Database Explorer** on the O/R Designer. In this example, you will drag the **Books** table from the database. This will automatically create a class named **Book**.

The O/R Designer showing Book class is displayed in figure 21.6.

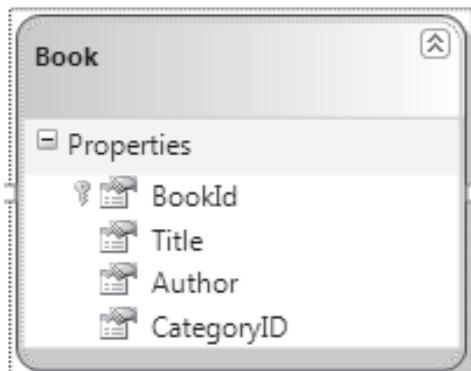


Figure 21.6: O/R Designer Showing Book Data Class Mapping to Books Table

You can also drag items from the Object Relational Designer Toolbox tab.

Figure 21.7 shows the Object Relational Designer Toolbox.

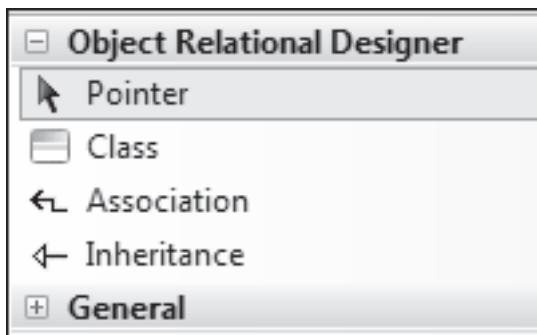


Figure 21.7: Object Relational Designer Toolbox

In LINQ to SQL, database tables are mapped to entity classes. The properties of each class map to the columns of the table. The `DataContext` class contains several built-in methods. For instance, the `SubmitChanges()` method is used to send data updates back to the database. In addition, when you drag a stored procedure or function onto the **Methods** pane in the Designer, the O/R Designer creates methods for each item.

Now, you have generated the `Book` class in the O/R Designer, so next you need to write code to run queries against the database.

A query expression must always begin with a `from` clause and end with either a `select` or a `group` clause. In addition, there are several other optional clauses.

Table 21.4 describes the clauses that you can use in a query expression.

Clause	Description
<code>from</code>	Used to specify a data source and a range variable representing each successive element in the source sequence.
<code>group</code>	Used to produce a sequence of groups organized by a specified key. The key can be any data type.
<code>let</code>	Used to store the result of an expression such as a method call in a temporary identifier.
<code>orderby</code>	Used to sort the results in either ascending or descending order.
<code>select</code>	Used to produce a sequence of results.
<code>where</code>	Used to filter out elements from the source data based on one or more predicate expressions.

Table 21.4: Clauses Used in Query Expressions

A query operation comprises the following actions:

1. Create the data source and an instance of the `DataContext` class by creating a **LINQ to SQL Classes** item. For the current scenario, you have already accomplished this.
2. Create the query using one or more clauses from table 21.4 to retrieve desired data from the database and also create a variable to hold the result of the query.

The following code snippet demonstrates an example of a LINQ to SQL query. This code is to be added in the `Page_Load` event handler so that the data is retrieved as soon as the Web page loads.

Code Snippet:

```
BooksDataContext objContext = new BooksDataContext();
var titles = from books in objContext.Books
            where books.Author == "Karim Khan"
            select books.Title;
```

In the code, the data source consists of the reference to the `DataContext` class. Here, the `BookDataContext` class represents the data source. The query expression contains the three clauses `from`, `where`, and `select`, and the expression is stored in the variable, `titles`. Though the query is created, it is not yet executed.

Execute the query. You can use either deferred or immediate execution to retrieve the query results. Deferred execution means that when a query results in a sequence of values, the execution of the query is deferred or postponed until you use a loop to iterate through the query variable. In this case, the query will be executed only when the query variable is used in a `foreach` loop.

Immediate execution means that when a query returns a single value, for example, a scalar query, the execution is immediate.

The following code snippet demonstrates execution of the LINQ to SQL query using deferred execution approach. This code is also to be added in the `Page_Load` event handler, following the code that was given in the earlier code.

Code Snippet:

```
foreach (var title in titles)
{
    lstBooks.Items.Add(title);
}
```

The `foreach` loop is used to iterate through the results. The query will be executed when the variable `titles` is used in the `foreach` loop. Each title retrieved from the collection of titles is stored into the `ListBox`, `lstBooks`.

The output is as shown in figure 21.8.

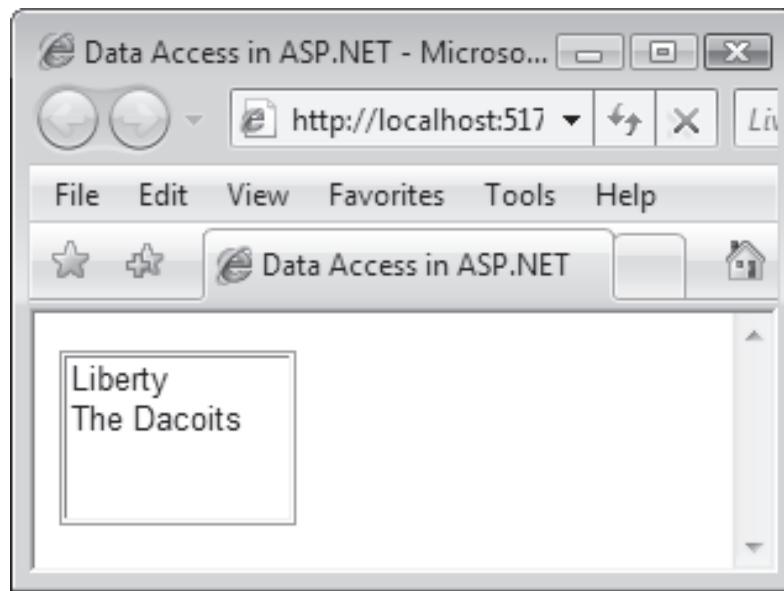


Figure 21.8: Output of the Program

21.7 Inserting, Updating, and Deleting with LINQ to SQL

Using LINQ to SQL, you can also insert, update, and delete data in your database easily from the Web application. To achieve this, you work with your object model to add a new instance to an object model collection, modify a given object, or remove an object from the collection. After performing the required actions, call the `SubmitChanges ()` method of the `DataContext` object to send the results back to the database.

For example, if you wish to add a new book to the Books table, create a new Book object, set the values for its members, and call the method `InsertOnSubmit ()`. When you are ready to submit all changes to the database, you call `SubmitChanges ()`.

The following code snippet demonstrates inserting a record using the LINQ to SQL approach:

Code Snippet:

```
BooksDataContext objContext = new BooksDataContext();
Book objBook = new Book();
objBook.Author = "George Harrison";
objBook.BookId = "18";
objBook.Title = "Recurrent Dreams";
objBook.CategoryID = "101";
objContext.Books.InsertOnSubmit(objBook);
objContext.SubmitChanges();
Response.Write("Record Added");
```

The code adds a record into the database through the `DataContext` class. Upon successful insertion, a message is displayed on the browser page.

You can update data by first retrieving the existing data in the table using a query and then, updating the data. Call `SubmitChanges()` to commit the changes to the database.

The following code snippet demonstrates updating a record using the LINQ to SQL approach:

Code Snippet:

```
BooksDataContext objContext = new BooksDataContext();
Book objBook = new Book();
var query = from books in objContext.Books
           where books.BookId == "18"
           select books;
Book updateBook = query.First();
updateBook.Title = "The Recurrent Dream";
objContext.SubmitChanges();
```

For deleting data, call the `DeleteOnSubmit()` method. You can pass the instance you wish to delete to this method. You can then call `SubmitChanges()`.

21.8 Using Stored Procedures with LINQ to SQL

You can drag and drop a stored procedure onto the O/R Designer window. This will result in Visual Studio 2008 creating a method on the LINQ to SQL `DataContext` class. The default name of the method is the same as the stored procedure name, although you can change this name if required. To run the stored procedure, call the appropriate method on the `DataContext` class. The method returns a sequence of strongly typed results from the stored procedure. One common task for which you can use a stored procedure is to bind the results of the procedure to a list-bound control or a `GridView` control on a Web page.

Consider an example that demonstrates this. You will create a stored procedure named `GetBooksByCategory` as shown in following Code Snippet:

Code Snippet:

```
CREATE PROCEDURE GetBooksByCategory ( @Category nchar(10) )
AS
    SELECT Title
    FROM Books A, BookCategories B
    WHERE B.Category=@Category
    AND A.CategoryID = B.CategoryID
```

In the design view of `Books.dbml`, drag the `BookCategories` and the stored procedure `GetBooksByCategory` from the **Server Explorer** to the designer area. Then, add the code shown in the following Code Snippet to the `Page_Load` event handler of `Library.aspx.cs`. The code binds the results of the `GetBooksByCategory` stored procedure to a `GridView` control.

Code Snippet:

```
BooksDataContext objContext = new BooksDataContext();
gvwBooks.DataSource = objContext.GetBooksByCategory("Thriller");
gvwBooks.DataBind();
```

The code creates an instance of a `DataContext` class and uses a method on the class to run the stored procedure.

The results of the call to the stored procedure through the method are assigned to the `DataSource` property of the `GridView` and then, the `DataBind()` method is called.

The output is shown in figure 21.9.

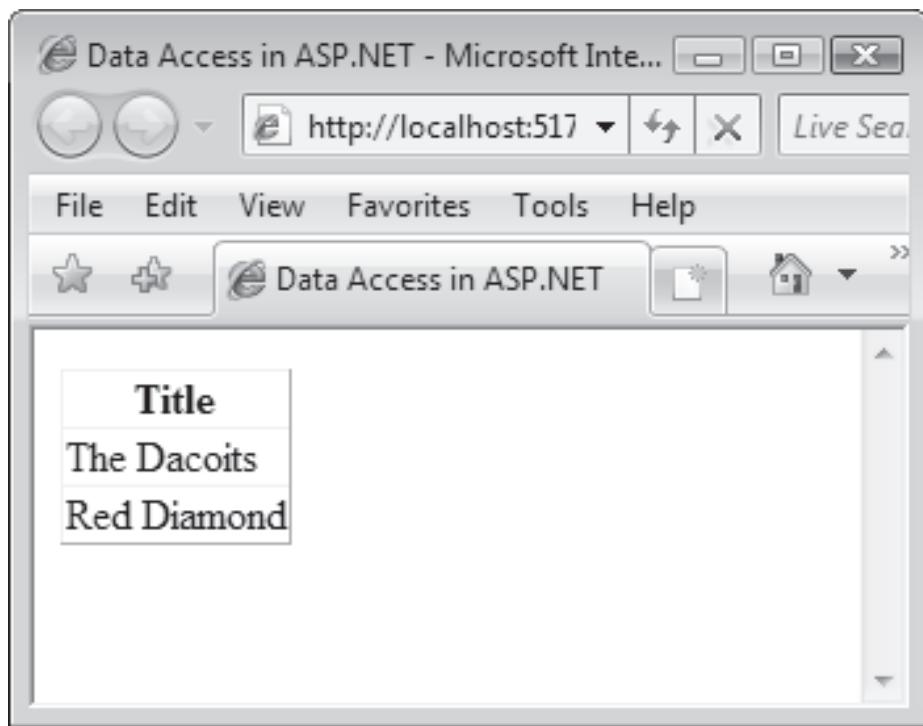


Figure 21.9: Output of the Code

21.9 Check Your Progress

1. When using LINQ to SQL to insert records from .NET applications to the database, which method needs to be called to save the changes in the database?

(A)	SubmitChanges	(C)	CommitChanges
(B)	AcceptChanges	(D)	Commit

2. You can call a stored procedure in .NET by using _____.

(A)	DataAdapter or DataReader objects	(C)	DataSets
(B)	DataContext objects	(D)	Call Stored Procedure command

3. The type of a parameter can be specified by using the Direction property which takes its value from the _____ enumeration.

(A)	Direction	(C)	InputOutput
-----	-----------	-----	-------------

(B)	ParameterDirection	(D)	Input
-----	--------------------	-----	-------

4. Which of the following statements are true about LINQ?

(A)	It helps you easily query against various data sources
(B)	It adds query capabilities to .NET languages such as C#.
(C)	LINQ can be used with SQL, XML files, objects (such as C# arrays and collections), and ADO.NET DataSet objects.
(D)	LINQ to Stored Procedure is used to execute stored procedures from .NET applications using LINQ.

(A)	a, b, d	(C)	a, b, c
(B)	a, c, d	(D)	b, c, d

5. Which of the following statements are true about LINQ clauses used in query expressions?

(A)	from is used to specify a data source and a range variable representing each successive element in the source sequence.
(B)	orderby is used to produce a sequence of groups organized by a key that you specify. The key can be any data type
(C)	where is used to store the result of an expression such as a method call in a temporary identifier
(D)	select is used to filter out elements from the source data based on one or more predicate expressions

(A)	a	(C)	a, b
(B)	b, c	(D)	c, d

6. You have created a stored procedure as follows:

```
CREATE PROCEDURE GetProducts (@Price int)
```

AS

```
SELECT *
FROM Products
WHERE Price > @Price
```

Which of the following code snippets correctly binds the GridView control to display products having price more than 300?

Module Summary

In this module, **Working with Stored Procedures and LINQ**, you learnt about:

→ **Stored Procedures**

A stored procedure is a set of precompiled database commands stored on the database server and can be used in .NET applications for data manipulation operations. You can create a stored procedure either in SQL Server Management Studio or in Visual Studio 2008 IDE through the Server Explorer window. Parameterized queries or stored procedures can prevent SQL injection attacks and enable better performance.

→ **LINQ**

LINQ is a set of features that adds query capabilities to .NET languages such as C# and enables you to query data from diverse data sources in an easy manner. LINQ to SQL is a technology that allows you to query and manipulate data from SQL Server databases in .NET applications. The DataContext class acts as a link between a SQL Server database and the LINQ to SQL entity classes mapped to that database. You can use LINQ to SQL with stored procedures to perform query operations as well as insert, update, and delete operations.

Module - 22

Working with Stored Procedures and LINQ (Lab)

Welcome to the module, **Working with Stored Procedures and LINQ (Lab)**.

In this module, you will learn to:

- Create and use parameterized stored procedures in ASP.NET
- Use LINQ to SQL with stored procedures

Web Development

<http://www>



22.1 Part I – 60 Minutes

Problem:

Alpha Development Corporation is a software development company based in Missouri, USA. They have now decided to diversify into social media tools. They plan to develop a small-scale Web-based application named QuickConnect that is similar in functionality to Facebook and LinkedIn.

QuickConnect allows new users to create accounts in a quick and easy manner by accepting only the most essential information from users. Once a user is registered, the user can add connections to his/her profile. They want to ensure that the Web application is safe from SQL injection attacks and accepts and stores user and connection data in a safe manner.

Assume that you are a developer at Alpha Development Corporation and that you have been given the task of creating the Web pages to accept and store user and connection information.

You must use Visual Studio 2008, ASP.NET 3.5, and SQL Server 2008 to build this application.

Solution:

The solution for this problem is to use SQL Server stored procedures to avoid injection attacks and perform the required data manipulation operations. The steps to achieve this are as follows:

1. Create a database named **QuickConnect** in SQL Server 2008 and two tables named **Users** and **Connections** respectively. The table design for these two tables is shown in figure 22.1.

Users Table Structure:

Column Name	Data Type	Allow Nulls
User Id	int	<input type="checkbox"/>
Full Name	nchar(50)	<input checked="" type="checkbox"/>
Password	nchar(10)	<input checked="" type="checkbox"/>
Address	nchar(100)	<input checked="" type="checkbox"/>
ProfileSummary	nvarchar(MAX)	<input checked="" type="checkbox"/>
EmailId	nchar(25)	<input checked="" type="checkbox"/>
Phone	nchar(20)	<input checked="" type="checkbox"/>

Connections Table Structure:

Column Name	Data Type	Allow Nulls
User Id	int	<input type="checkbox"/>
ConnUser Id	int	<input type="checkbox"/>
Conn Name	nchar(50)	<input type="checkbox"/>
Conn Gender	nchar(1)	<input checked="" type="checkbox"/>
Conn Type	nchar(20)	<input checked="" type="checkbox"/>

Figure 22.1: Table Design

2. Add a few records into the two tables.

3. Launch Visual Studio 2008.
4. Create an ASP.NET Web site named **QuickConnect** on the localhost. Figure 22.2 shows the Web site being created.

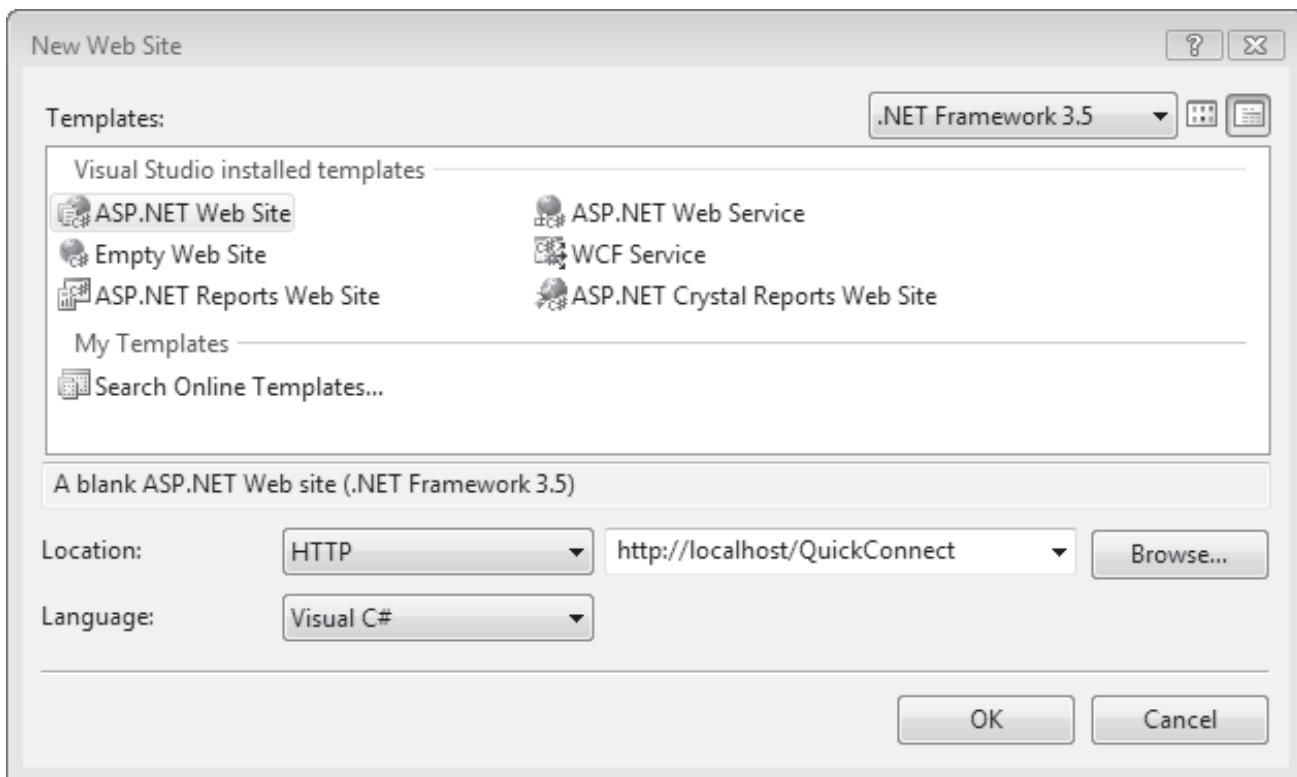


Figure 22.2: Creating a new ASP.NET Web site

Exercise 1: Adding Users

1. Rename the default Web Form, **Default.aspx**, to **Home.aspx**. Also, rename all instance of **Default** to **Home**.
2. Open the form in **Source view** and edit the title section of the form as well as the form id as follows:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Home.aspx.cs"
Inherits="Home" %>

<!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>A Quick and Easy Way to Connect!</title>
```

```

        </head>
<body>
    <form id="frmHome" runat="server">
        </form>
    </body>
</html>

```

3. Add the following markup below the form tag:

```

<div>
    <h2>Welcome to QuickConnect - the easy way to connect to people
    </h2>
</div>

```

4. Add two HyperLink and two Image controls on the form and configure their properties as shown in table 22.1.

Server Control	Property	Value
HyperLink	ID	InkUsers
	Text	Add Users
	Font	Garamond
	Font-Size	Medium
	Bold	True
	Italic	True
	NavigateUrl	~/AddUsers.aspx
HyperLink	ID	InkConnections
	Text	Add Connections
	Font	Garamond
	Font-Size	Medium
	Bold	True
	Italic	True
	NavigateUrl	~/AddConnections.aspx
Image	ID	lblPassword
	ImageUrl	~/Images/user.jpg:
Image	ID	imgConnections
	ImageUrl	~/Images/connections.jpg:

Table 22.1: Properties of the Home Page Controls

5. Arrange them as shown in figure 22.3.

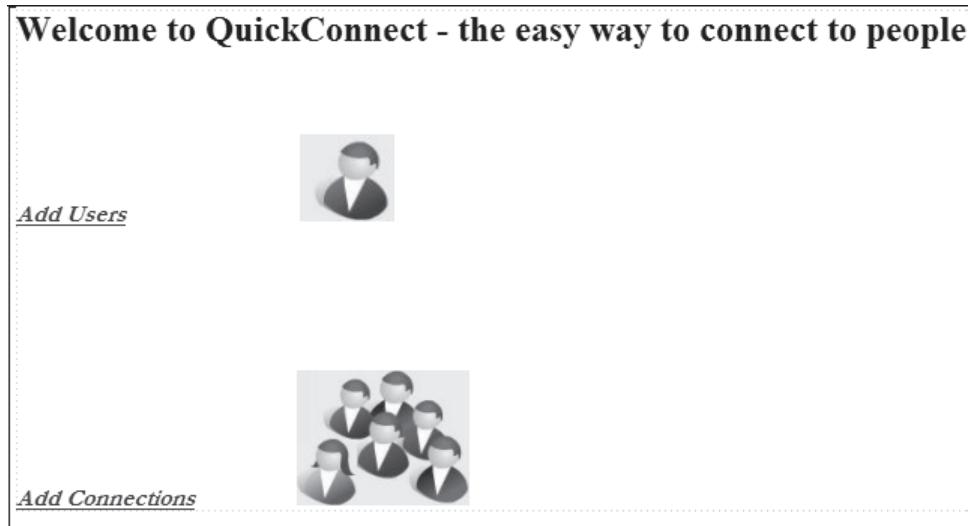


Figure 22.3: User Interface of Home Page

6. Add a new Web Form to the Web site and name it as **AddUsers.aspx**.
7. Open the form in **Source view** and edit the title section of the form as well as the form id as follows:

```
<!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>Add Users</title>
</head>
<body>
<form id="frmUsers" runat="server">
    <div>
    </div>
</form>
</body>
</html>
```

8. Add eight Label, seven Text box, and two Button controls to the Web Form. Also, add a HyperLink control.
9. Configure the properties of the controls as shown in table 22.2.

Server Control	Property	Value
Label	ID	lblUserId

	Text	User ID:
Label	ID	lblName
	Text	Full Name:
Label	ID	lblPassword
	Text	Password:
Label	ID	lblAddress
	Text	Address:
Label	ID	lblProfileSummary
	Text	Profile Summary:
Label	ID	lblEmailID
	Text	Email ID:
Label	ID	lblPhone
	Text	Phone:
Label	ID	lblStatus
	Text	
	Font-Bold	True
	Font-Color	Red
Text box	ID	txtUserId
Text box	ID	txtName
Text box	ID	txtPassword
	TextMode	Password
Text box	ID	txtAddress
	TextMode	MultiLine
Text box	ID	txtProfile
	TextMode	MultiLine
Text box	ID	txtEmail
Text box	ID	txtPhone
Button	ID	btnCancel
	Text	Cancel
Button	ID	btnDone
	Text	Done
HyperLink	ID	InkHome
	NavigateUrl	~/Home.aspx
	Text	Back to Home Page

Table 22.2: Properties of the AddUsers Page Controls

10. Add four validation controls and set their properties as shown in table 22.3.

Server Control	Property	Value
RequiredFieldValidator	ID	reqvalUserId
	ControlToValidate	txtUserId
	Text	* User ID Required
RequiredFieldValidator	ID	reqvalName
	ControlToValidate	txtName
	Text	* Name Required
RequiredFieldValidator	ID	reqvalPassword
	ControlToValidate	txtPassword
	Text	* Password Required

Table 22.3: Validation Controls on AddUsers Page

11. Arrange all the controls as shown in figure 22.4.

h2>Welcome to QuickConnect - the easy way to connect to people

Your User Id:

Connection User Id: * Please select Connection User Id

Connection Name:

Connection Address:

Connection Gender: Male Female

Connection Type:

[lblStatus]

[Back to HomePage](#)

Figure 22.4: User Interface of the AddUsers Web Form

12. Launch **Server Explorer** using the **View → Server Explorer** option.

13. Right-click **Data Connections** on the **Server Explorer** and select **Add Connection**. The **Add Connection** dialog box is displayed.
14. Configure the connection settings as shown in figure 22.5.

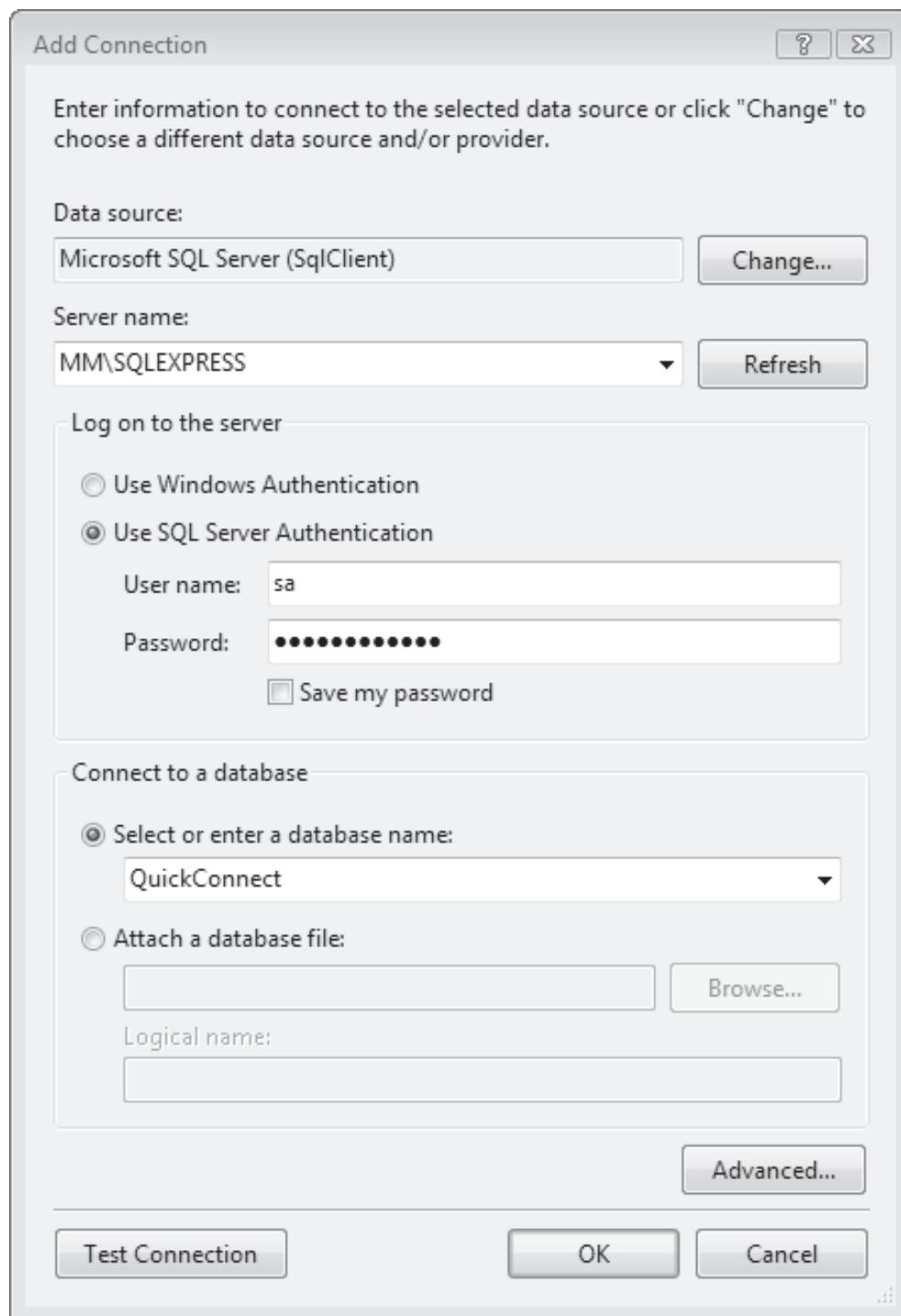


Figure 22.5: Add Connection Dialog Box

15. Open the **web.config** file and in the connection strings section, rename the default connection string

name to **QuickConnectionString**.

16. Launch SQL Server Management Studio and create the following two procedures:

```

CREATE Procedure [dbo].[AddUsers] (@UserId int, @Name nchar(50),
@Password nchar(10), @Address nchar(100), @ProfileSummary
nvarchar(MAX), @EmailId nchar(25), @Phone nchar(20))

AS

INSERT INTO Users
(
    [UserId]
    , [FullName]
    , [Password]
    , [Address]
    , [ProfileSummary],
    [EmailId],
    [Phone])
VALUES
(
    @UserId, @Name, @Password, @Address, @ProfileSummary, @
EmailId, @Phone)

CREATE Procedure [dbo].[AddConnections] (@UserId int, @ConnUserId int,
@ConnName char(50), @ConnGender nchar(1), @ConnType nchar(20))

AS

INSERT INTO Connections
(
    [UserId],
    [ConnUserId]
    , [ConnName]
    , [ConnGender]
    , [ConnType])
VALUES
(
    @UserId, @ConnUserId, @ConnName, @ConnGender, @ConnType)

```

17. Revert to Visual Studio 2008 IDE and in the QuickConnect Web site, open **AddUsers.aspx.cs** and add the following namespace declaration in the using section.

```
using System.Data.SqlClient;
```

18. Generate the Click event of the button, **btnDone**. Add the following code to the event handler.

```

/// <summary>
/// Event Handler for Click event
/// Contains the logic to call parameterized stored procedure
/// and pass parameters
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void btnDone_Click(object sender, EventArgs e)
{
    try
    {
        SqlConnection objConn = new SqlConnection();
        objConn.ConnectionString = ConfigurationManager.
        ConnectionStrings["QuickConnectionString"].ConnectionString;
        objConn.Open();
    }
}

```

This code will retrieve the connection string from **web.config** and set the connection object **objConn**.

19. Further, in the same event handler, add the code as follows:

```

SqlDataAdapter objAdapter = new SqlDataAdapter("AddUsers", objConn);
objAdapter.InsertCommand = new SqlCommand();
objAdapter.InsertCommand.Connection = objConn;
objAdapter.InsertCommand.CommandType = CommandType.StoredProcedure;
objAdapter.InsertCommand.CommandText = "AddUsers";

```

This code will create an **SqlDataAdapter** object and configure its **InsertCommand** property to a stored procedure named **AddUsers**. Now, the **AddUsers** procedure accepts several input parameters. Therefore, you need to create **SqlParameter** objects and add them to the **Parameters** collection of **InsertCommand**.

20. Add the following code in the **btnDone** event handler.

```
// Create parameter for user id
SqlParameter objParam = new SqlParameter("@UserID", SqlDbType.
                                         Int);
objParam.Direction = ParameterDirection.Input;
objParam.Value = this.txtUserId.Text;

// Add the parameter to the Parameters collection of InsertCommand
objAdapter.InsertCommand.Parameters.Add(objParam);
```

This code will create an `SqlParameter` object, `objParam`, of type `Int`, having `Direction` as `Input`, and value from the text box, `txtUserId`. This parameter is then added to the `Parameters` collection of the `InsertCommand` instance.

21. Similarly, add the other parameters as follows:

```
objParam = new SqlParameter("@Name", SqlDbType.NChar, 50);
objParam.Direction = ParameterDirection.Input;
objParam.Value = this.txtName.Text;
objAdapter.InsertCommand.Parameters.Add(objParam);

objParam = new SqlParameter("@Password", SqlDbType.NChar, 10);
objParam.Direction = ParameterDirection.Input;
objParam.Value = this.txtPassword.Text;
objAdapter.InsertCommand.Parameters.Add(objParam);

objParam = new SqlParameter("@Address", SqlDbType.NChar, 100);
objParam.Direction = ParameterDirection.Input;
objParam.Value = this.txtAddress.Text;
objAdapter.InsertCommand.Parameters.Add(objParam);

objParam = new SqlParameter("@ProfileSummary", SqlDbType.NVarChar, -1);
objParam.Direction = ParameterDirection.Input;
objParam.Value = this.txtProfile.Text;
```

```

objAdapter.InsertCommand.Parameters.Add(objParam);

objParam = new SqlParameter("@EmailId", SqlDbType.NChar, 25);
objParam.Direction = ParameterDirection.Input;
objParam.Value = this.txtEmail.Text;
objAdapter.InsertCommand.Parameters.Add(objParam);

objParam = new SqlParameter("@Phone", SqlDbType.NChar, 20);
objParam.Direction = ParameterDirection.Input;
objParam.Value = this.txtPhone.Text;
objAdapter.InsertCommand.Parameters.Add(objParam);

```

22. Next, add the code to execute the InsertCommand, display appropriate message, and clear the text boxes.

```

int result = objAdapter.InsertCommand.ExecuteNonQuery();
if (result > 0)
{
    lblStatus.Text = "Your record has been successfully created.";
    Clear();
}

```

23. Finally, end the event handler code with a catch statement to trap any runtime errors that may occur.

```

catch (Exception ex)
{
    lblStatus.Text = "An error occurred while adding data. Ensure
                    that data is not duplicate or invalid. ";
}

```

24. Write the following code for the Clear() method which will clear the contents of the text boxes:

```
private void Clear()
{
    txtUserId.Text = "";
    txtName.Text = "";
    txtPassword.Text = "";
    txtAddress.Text = "";
    txtProfile.Text = "";
    txtEmail.Text = "";
    txtPhone.Text = "";
}
```

25. Generate Click event for the **Cancel** button. Add the following to the event handler.

```
/// <summary>
/// Event Handler for Click event of btnCancel
/// Contains the logic to clear the textboxes by calling Clear()
/// method
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void btnCancel_Click(object sender, EventArgs e)
{
    Clear();
}
```

26. Save, build, and execute the application. The **Home.aspx** page is displayed as shown in figure 22.6.

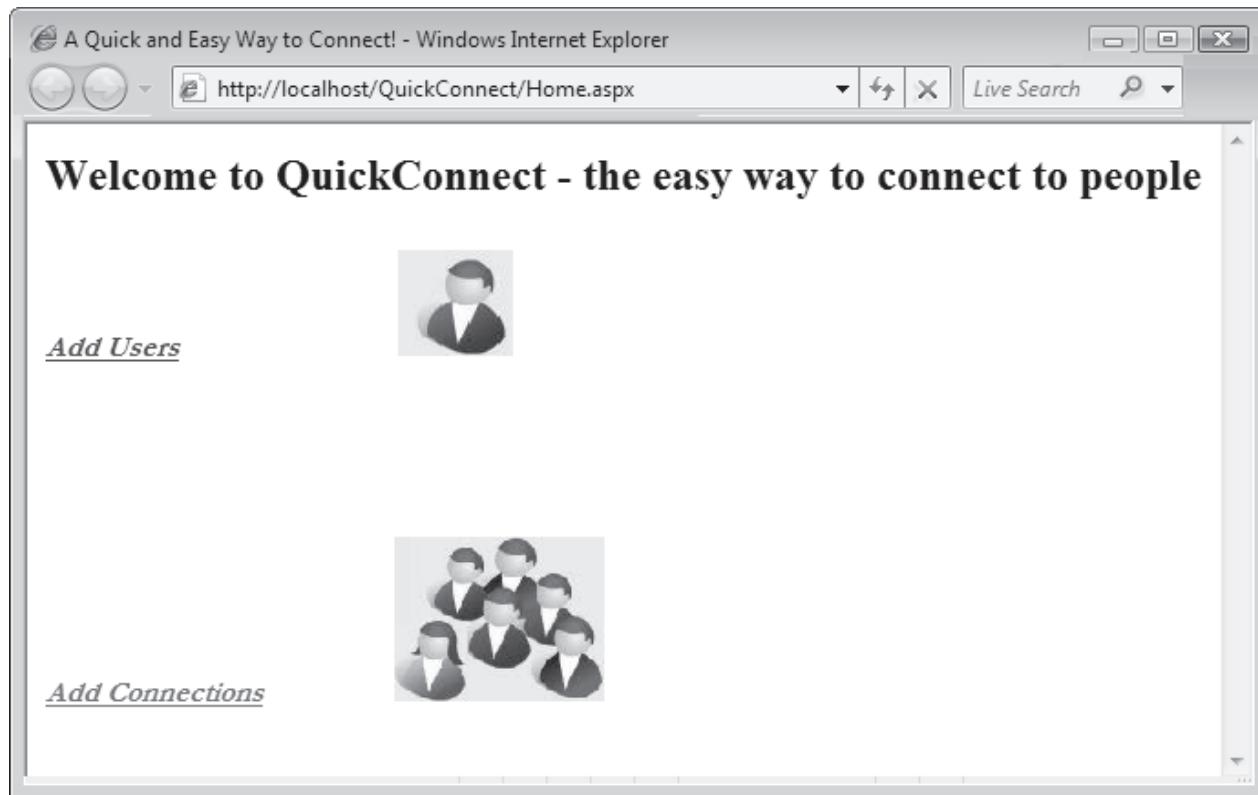


Figure 22.6: Home.aspx Page

27. Click '**Add Users**'. The **AddUsers** page will be displayed.

28. Enter a user id, 7654, and click **Done**. The output with invalid data is shown in figure 22.7.

Add Users - Windows Internet Explorer
http://localhost/QuickConnect/AddUsers.aspx

Welcome to QuickConnect - the easy way to connect to people

User ID: 7654

Full Name: * Name Required

Password: * Password Required

Address:

Profile Summary:

Email ID:

Phone:

Done Cancel

[Back to Home Page](#)

Figure 22.7: Output Showing Data Being Validated

The output with valid data is shown in figure 22.8.

User ID: 499

Full Name: Jeremy Ford

Password: *****

Address:
117, Sunshine Apts,
Fifth Avenue, Main
Street, Iowa

Profile Summary:
Jeremy likes football,
music and reading.

Email ID: jeremyford@gmail.com

Phone: 7723005

Done Cancel

Figure 22.8: Output with Valid Data

Upon success, the record will be entered into the Users table in the database.

Exercise 2: Adding Connections

Similar to the approach of adding users, you will now add connections for each user.

1. Create another Web Form in the same solution named **AddConnections.aspx** for adding new connections for a user.
2. Add controls on the form as listed in table 22.4 and configure their properties.

Server Control	Property	Value
Label	ID	lblUserId
	Text	Your User Id:

Server Control	Property	Value
Label	ID	lblConnUserId
	Text	Connection User Id:
Label	ID	lblConnName
	Text	Connection Name:
Label	ID	lblAddress
	Text	Connection Address:
Label	ID	lblGender
	Text	Connection Gender:
Label	ID	lblType
	Text	Connection Type:
DropDownList	ID	ddlUserId
	AutoPostBack	True
DropDownList	ID	ddlConnUserId
	AutoPostBack	True
RadioButtonList	ID	rdolstGender
	AutoPostBack	True
	Items	Male
		Female
DropDownList	ID	ddlType
	AutoPostBack	True
	Items	Friend
		Family
		Colleague
Text box	ID	txtName
Text box	ID	txtLocation
Button	ID	btnCancel
	Text	Cancel
Button	ID	btnDone
	Text	Done
Label	ID	lblStatus
	Text	
	Font-Bold	True
HyperLink	ID	InkHome
	Text	Back to Home Page
	NavigateUrl	~/Home.aspx

Table 22.4: Properties of the controls on Add Connections Web Form

3. Add a validation control and set the properties as shown in table 22.5.

Server Control	Property	Value
RequiredFieldValidator	ID	valConnUserId
	ControlToValidate	ddlConnUserId
	ErrorMessage	Connection User Id is Required
	Text	* Please select Connection User Id

Table 22.5: Validation Control on Add Connections Web Form

4. Arrange the user interface of the form as shown in figure 22.9.

h2

Welcome to QuickConnect - the easy way to connect to people

Your User Id:

Connection User Id: * Please select Connection User Id

Connection Name:

Connection Address:

Connection Gender: Male Female

Connection Type:

[lblStatus]

[Back to HomePage](#)

Figure 22.9: User Interface of the Add Connections Web Form

5. Add the following namespace declaration in the ‘using’ section of **AddConnections.aspx.cs**.
- ```
using System.Data.SqlClient;
```
6. Add the following code to the declaration section of **AddConnections.aspx.cs**. This code declares various variables that would be used throughout the application.

```
SqlConnection objConn;
SqlDataAdapter objAdapter;
string userId = "", type="";
char gender='F';
```

7. Add the following code to the Page \_ Load event handler:

```
protected void Page_Load(object sender, EventArgs e)
{
 // Populate the user ids into a dataset which can be used
 // as a source for the drop down list
 objConn = new SqlConnection();
 objConn.ConnectionString = ConfigurationManager.ConnectionStrings
 ["QuickConnectionString"].ConnectionString;

 objConn.Open();
 objAdapter = new SqlDataAdapter("SELECT UserId FROM Users",
 objConn);
 DataSet ds = new DataSet();
 objAdapter.Fill(ds);

 // Carry out these actions if the form is being loaded
 // for the first time
 if (!IsPostBack)
 {
 // Bind the user id
 ddlUserId.DataSource = ds.Tables[0];
 ddlUserId.DataTextField = "UserId";
 ddlUserId.DataBind();
 // Bind the connection user id
 ddlConnUserId.DataSource = ds.Tables[0];
 }
}
```

```

 ddlConnUserId.DataTextField = "UserId";
 ddlConnUserId.DataBind();

 userId = ddlUserId.Items[0].Text.ToString();
 connUserId = ddlConnUserId.Items[0].Text.ToString();

 type = ddlType.Items[0].Text;
 rdolstGender.Items[1].Selected = true;
 }

 else
 {
 // Set the defaults so that if the user does not select any
 // option, the default value would be read
 userId = ddlUserId.SelectedItem.Text.ToString();
 connUserId = ddlConnUserId.SelectedItem.Text.ToString();
 if (rdolstGender.SelectedItem.Value == "Female")
 gender = 'F';
 else
 gender = 'M';
 type = ddlType.SelectedItem.Text.ToString();
 }

 ddlConnUserId.Items.Remove("");
 ddlConnUserId.Items.Insert(0, "");
}

```

As the user ids must be populated in the drop-down list only once, when the page is loaded, the code to retrieve data is placed in a if block. After the data is retrieved from the table, it is filled into a DataSet object which is then, assigned to the user id drop-down list. Similarly, the connection user id drop-down is also populated. Then, the default values are set for the two drop-down lists and the radio button list. If the user makes any selection, the selected values are stored into appropriate variables.

8. Generate Click event of the button, **btnDone**. Add the following code to the event handler.

```

/// <summary>
/// Event Handler for Click event of btnDone
/// Contains the logic to call parameterized stored procedure
/// and pass parameters
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void btnDone_Click(object sender, EventArgs e)
{
 try
 {
 if (lblStatus.Text == "")
 {
 objAdapter.InsertCommand = new SqlCommand();
 objAdapter.InsertCommand.Connection = objConn;
 objAdapter.InsertCommand.CommandType = CommandType.StoredProcedure;
 objAdapter.InsertCommand.CommandText = "AddConnections";
 // Create parameter for user id
 SqlParameter objParam = new SqlParameter("@UserID",
 SqlDbType.Int);
 objParam.Direction = ParameterDirection.Input;
 objParam.Value = userId;
 // Add the parameter to the Parameters collection of
 // InsertCommand
 objAdapter.InsertCommand.Parameters.Add(objParam);
 // Create parameter for connection user id
 objParam = new SqlParameter("@ConnUserID", SqlDbType.Int);
 objParam.Direction = ParameterDirection.Input;
 objParam.Value = connUserId;
 }
}

```

```

// Add the parameter to the Parameters collection of
// InsertCommand

objAdapter.InsertCommand.Parameters.Add(objParam);

objParam = new SqlParameter("@ConnName", SqlDbType.NChar, 50);
objParam.Direction = ParameterDirection.Input;
objParam.Value = this.txtName.Text;
objAdapter.InsertCommand.Parameters.Add(objParam);

objParam = new SqlParameter("@ConnGender", SqlDbType.NChar, 1);
objParam.Direction = ParameterDirection.Input;
objParam.Value = gender;
objAdapter.InsertCommand.Parameters.Add(objParam);

objParam = new SqlParameter("@ConnType", SqlDbType.NChar, 20);
objParam.Direction = ParameterDirection.Input;
objParam.Value = type;
objAdapter.InsertCommand.Parameters.Add(objParam);

int result = objAdapter.InsertCommand.ExecuteNonQuery();

if (result > 0)

{
 lblStatus.Text = "Connection successfully added. You are
 now connected to " + txtName.Text;
 Clear();
}

}
}

```

The code creates parameters for each of the input fields, assigns them values and adds them to the Parameters collection of the InsertCommand object. These parameters will be passed to the stored procedure. Finally, the stored procedure is called through the `ExecuteNonQuery()` method. A `try-catch` block is included to handle any errors that may occur.

9. Add the following code to clear the input fields.

```
private void Clear()
{
 txtName.Text = "";
 txtAddress.Text = "";
 ddlConnUserId.Text = "";
}
```

The text box values and initial value of drop-down list are set to blank strings so that a new record may be entered.

10. Generate Click event of the button, **btnCancel**. Add the following code to the event handler.

```
/// <summary>
/// Event Handler for Click event of btnCancel
/// Contains the logic to clear the textboxes by calling Clear()
/// method
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void btnCancel_Click(object sender, EventArgs e)
{
 Clear();
}
```

11. Generate the SelectedIndexChanged event for the **DropDownList** control. Add the following code to the event handler.

```
protected void ddlConnUserId_SelectedIndexChanged(object sender,
EventArgs e)
{
 lblStatus.Text = "";
 if (userId == connUserId)
 {
 lblStatus.Text = "Both ids cannot be the same";
 }
 else
 {
 objAdapter.SelectCommand = new SqlCommand("SELECT FullName, Address
FROM Users where UserId=" + connUserId, objConn);
 SqlDataReader objReader = objAdapter.SelectCommand.ExecuteReader();
 objReader.Read();
 txtName.Text = objReader.GetString(0).ToString();
 txtAddress.Text = objReader.GetString(1).ToString();
 }
}
```

The code verifies if both the ids selected by the user are the same, and if yes, an error message is displayed in the Label. If the ids are not the same, then, the FullName and Address of the Connection id are retrieved from the table and displayed in the respective text boxes.

12. Save, build, and execute the application. The output is shown in figure 22.10.

The screenshot shows a Windows Internet Explorer window titled "Add Connections - Windows Internet Explorer". The URL in the address bar is "http://localhost/QuickConnect/AddConnection:". The page content is as follows:

**Welcome to QuickConnect - the easy way to connect to people**

Your User Id:

Connection User Id:

Connection Name:

Connection Address:  
212, Georgia House,  
Lake Purdue,  
Iowa

Connection Gender:  Male  Female

Connection Type:

[Back to HomePage](#)

Figure 22.10: Adding Connections

Upon success, the record will be entered into the **Connections** table in the database and a message is displayed as shown in figure 22.11. The user interface will be ready to accept another new record.

Add Connections - Windows Internet Explorer  
http://localhost/QuickConnect/AddConnection: Live Search

Welcome to QuickConnect - the easy way to connect to people

Your User Id: 299

Connection User Id:

Connection Name:

Connection Address:

Connection Gender:  Male  Female

Connection Type: Friend

Done Cancel

Connection successfully added. You are now connected to Nick Stewart

Figure 22.11: Result of Adding Connection

## 22.2 Part II - 60 Minutes

Accomplish the tasks shown in Exercise 1 and 2 by using LINQ to SQL and stored procedures.

**Hints:**

Create a new ASP.NET Web site

Add connection

Design the forms similar to the ones in Exercise 1 and 2

Add a LINQ to SQL Classes item to the Web site

Use the DataContext class in the code to call the stored procedure

## 22.3 Do It Yourself

Modify the applications in Part I and II to include Update and Delete operations on Users and Connections tables. Implement more validations for password and e-mail and also include appropriate error messages instead of a generic error message.

# Module - 23

## ASP.NET AJAX and XML Web Services

Welcome to the module, **ASP.NET AJAX and XML Web Services**. This module covers the features and functionality of ASP.NET AJAX. The module begins with a brief introduction to AJAX. The module then explains about the ASP.NET AJAX Control Toolkit. The module concludes with a description of XML Web services.

In this module, you will learn to:

- ➔ Describe ASP.NET AJAX
- ➔ Describe ASP.NET AJAX Extensions
- ➔ List ASP.NET AJAX Server Controls
- ➔ Describe the ASP.NET AJAX Control Toolkit
- ➔ Explain XML Web services
- ➔ List the steps for creating and calling an XML Web service

Web Development  
<http://www>



## 23.1 Introduction

Asynchronous JavaScript and XML (AJAX) is a framework that enables you to develop highly interactive and efficient Web applications that can target all popular browsers. ASP.NET AJAX is a .NET-based AJAX Framework. ASP.NET AJAX architecture comprises client-script libraries and server-side components. ASP.NET AJAX 2.0 is built-in with .NET Framework 3.5. Visual Studio 2008 IDE and .NET Framework 3.5 together allow developers to build ASP.NET AJAX applications easily.

## 23.2 Introduction to AJAX

Traditional Web applications, also called traditional Internet applications, have a number of drawbacks which led to the emergence of AJAX. The following sections trace the evolution of AJAX and also describe its features.

### 23.2.1 Traditional Internet Applications

In traditional Web-based applications, each user action or request from the user results in lengthy postbacks. The user has to wait for completion of the postback before further processing can take place. Such kind of programming is called synchronous Web programming. In synchronous Web programming, the browser client passes a request to the server synchronously.

Consider a scenario of a small book store to understand this. Assume that there is just one person at the counter who has to process orders, search books, prepare bills, and hand the books and bills over to the customers. After placing an order, the customer cannot give another order, because by then, the store owner is busy searching the books. So, while store owner is busy, the customer has to wait, and cannot perform any other purchase.

Now, consider a Web portal designed for this scenario. The person at the counter is represented by a Web server. This server processes requests from browser clients, which are the equivalent of customers at the book store.

In the actual book store, paper ledgers store the details of purchase orders and billing details. In the Web portal, a database server stores such data. Processing occurs at the Web server, which then, sends back the result to the browser client. This result is then displayed on the browser.

Figure 23.1 depicts this process.

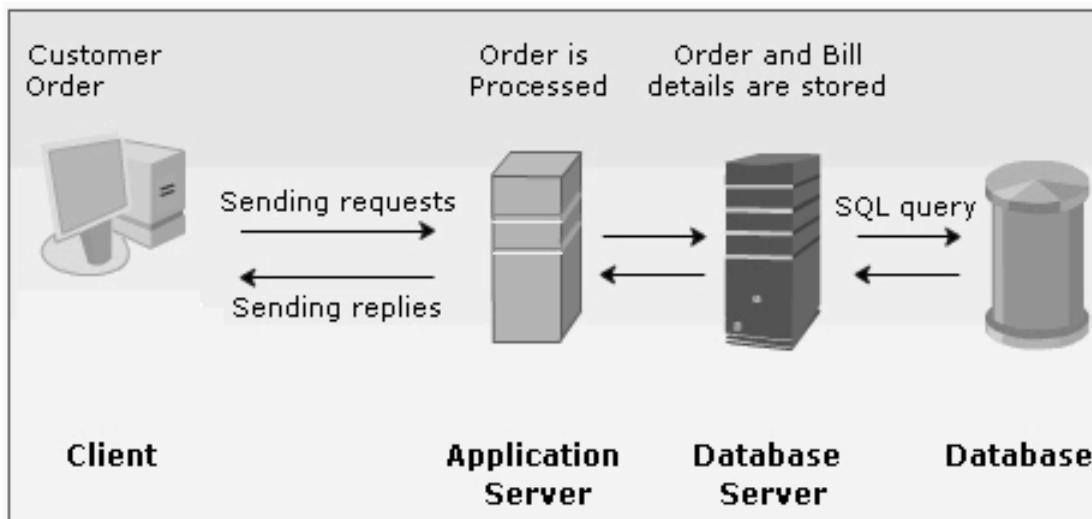


Figure 23.1: Book Store Web Portal

In a traditional Web architecture, the user must wait for the entire Web page to reload to see the new results from the server or perform further actions. In an application that requires a lot of interactivity, the user needs to reload the entire page many times.

Figure 23.2 depicts the traditional Web architecture.

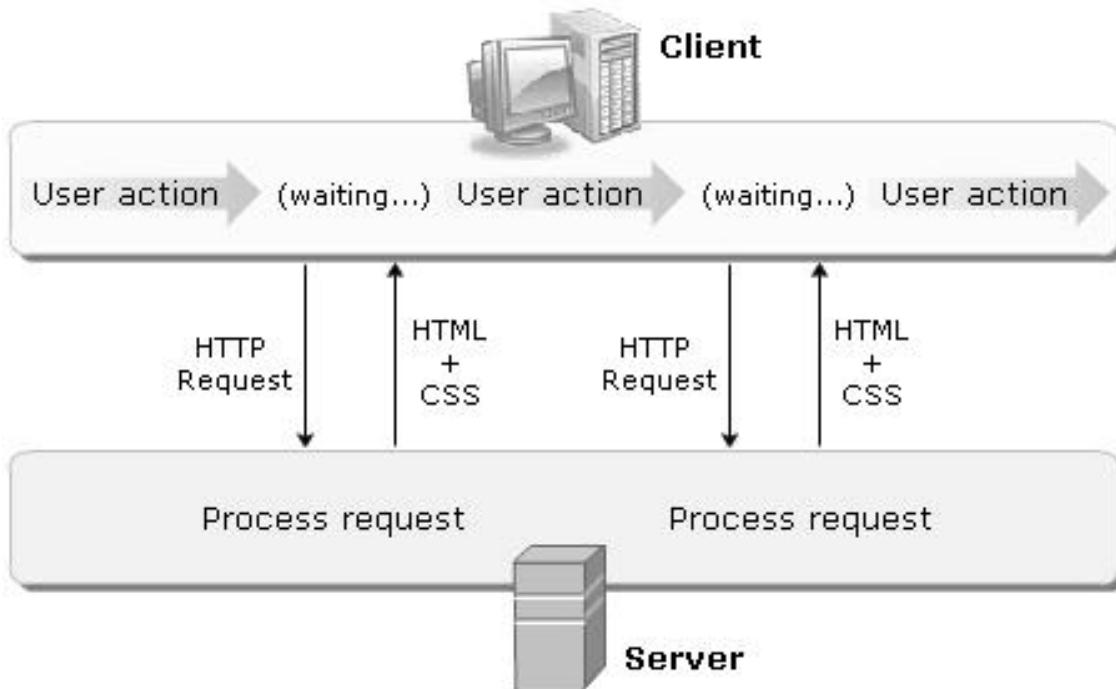


Figure 23.2: Traditional Web Architecture

Synchronous programming approach had several drawbacks, some of which are as follows:

- Lack of continuous interactivity on Web pages
- Reduced responsiveness due to long waiting periods
- Reduced efficiency
- Decrease in productivity of users

Asynchronous programming provides the key solution to these drawbacks. Web applications which use asynchronous programming pass the user request asynchronously to the server. This means that the user can continue with other actions while the server processes the requests. Hence, there is no delay or waiting period. If the book portal implements asynchronous programming, a customer can place an order, and continue to perform other purchases without waiting for the entire page to be posted back. A Web-based application that makes use of asynchronous programming and offers a highly interactive, responsive and personalized experience for users is known as Rich Internet Application (RIA).

### 23.2.2 Rich Internet Applications

A RIA is a Web application having the features and functionality of traditional desktop applications. In a RIA, the Web client handles processing necessary for the user interface. The application server handles tasks such as state and data processing, and so forth.

A RIA has a number of powerful benefits such as high interactivity and responsiveness, network efficiency, and so forth. One of the most popular RIA technologies is AJAX.

### 23.2.3 AJAX

AJAX is a group of technologies used to develop highly interactive and responsive Web applications. Some of these technologies include XML, JavaScript, DOM, and CSS. AJAX is an approach for Web application development that uses client-side scripting to exchange data with a Web server.

From a developer-oriented view, AJAX refers to a set of development components, tools, and techniques for creating highly interactive Web applications that give users a better experience. From an end user's point of view, an AJAX-enabled application results in faster round trips, and less time for page refreshes.

Some of the advantages of AJAX over traditional Internet applications are as follows:

#### → Asynchronous Processing

Using AJAX, you can make asynchronous calls to a Web server. This means that you will no longer have to wait for the entire page to be loaded before you perform your next action. This results in higher interactivity and faster responses.

**→ Minimal Data Transfer**

Since a full postback is not required in the asynchronous approach, entire form data need not be sent to the server. This minimizes the network utilization and results in quicker operations.

**→ Limited Processing on the Server**

In the asynchronous approach, the server is not required to process all of the form elements. You can send only the most essential data and limit the processing that is to be performed on the server.

**→ Better Responsiveness**

In AJAX-enabled applications, pages can be partially refreshed and this results in faster responses. A Web application framework enables development of Web applications, Web sites, and Web services by providing libraries and Application Programming Interfaces (APIs). A Web service is a component that enables applications to communicate by using predefined messages that are built around standard protocols such as HTTP, and others.

An AJAX Framework facilitates Web application development with AJAX.

AJAX comprises the following components and technologies:

**→ XMLHttpRequest**

XMLHttpRequest defines an API that is used by scripting languages such as JavaScript to load or transfer data to and from the server and the browser. In addition to fetching data in XML format, XMLHttpRequest can also fetch data in other formats such as XML, HTML, JSON, or even plain text.

**→ JavaScript**

JavaScript is a scripting language that provides the capabilities to communicate with the back-end server. It is the only client-side scripting environment supported by most of the modern Web browsers.

**→ DHTML/DOM Support**

Dynamic update of form elements is possible through the use of DOM.

**→ XML or JSON**

XML is often used to communicate with the Web server and exchange data in a standard fashion. However, there are situations where JSON is preferred as the communication notation instead of XML.

→ CSS

CSS is a technology that allows users to create and apply styles. AJAX server controls make ample use of CSS to generate visually attractive Web pages.

### 23.3 ASP.NET AJAX

Microsoft ASP.NET AJAX is an AJAX Framework that enables you to develop highly interactive and efficient Web applications that can target all popular browsers.

Figure 23.3 shows the ASP.NET AJAX architecture comprising server-side components and client-script libraries.

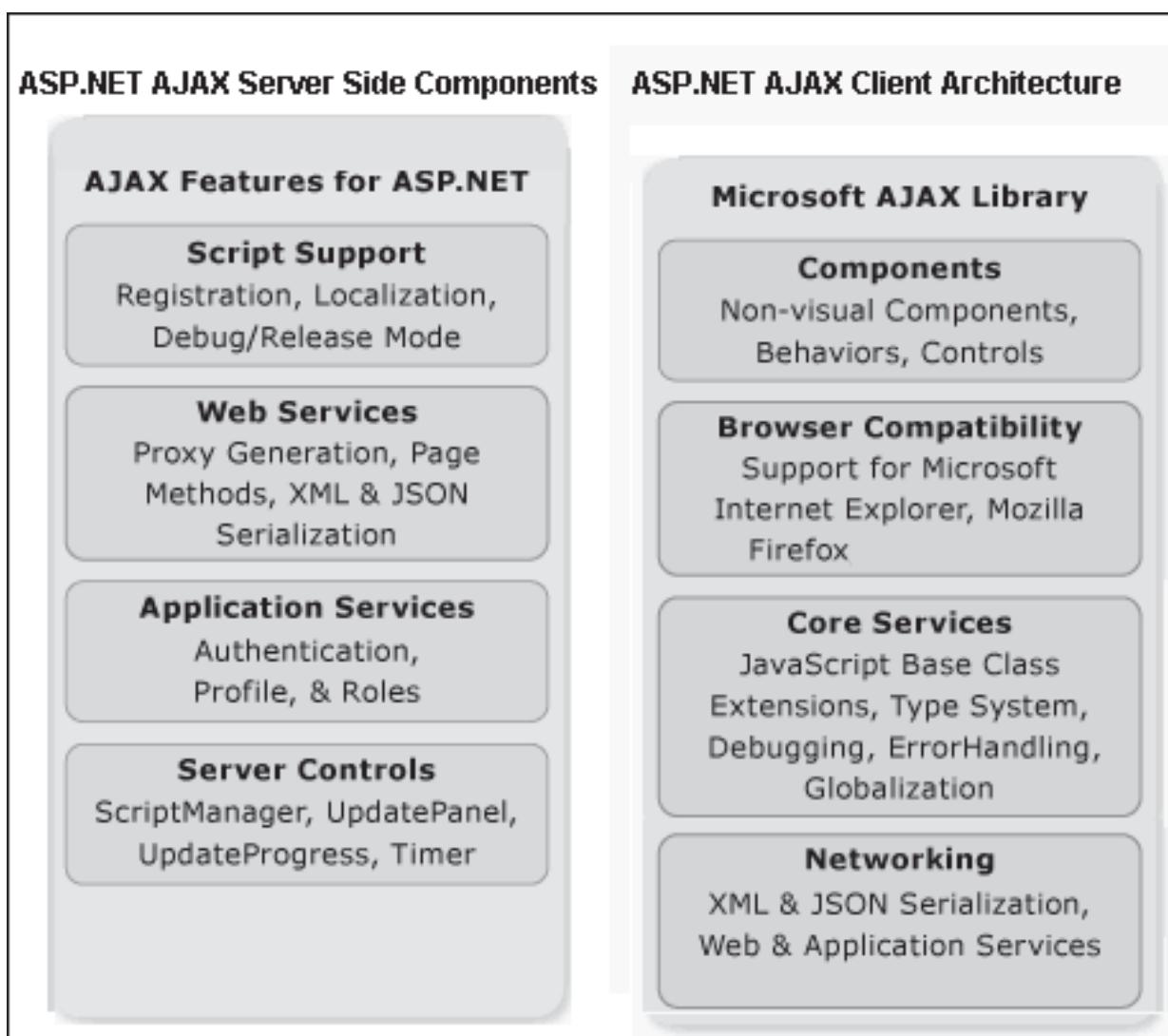


Figure 23.3: ASP.NET AJAX Architecture

The Microsoft AJAX Client Library represents the AJAX client architecture. AJAX applications use client-side JavaScript libraries to manage the user interface of a Web page, call server-based components, and render partial page updates.

The server architecture of ASP.NET AJAX includes Web services, ad hoc controls, and application services. Server components manage processing of client requests, data validation, data serialization, and so forth.

The Microsoft AJAX library comprises a set of JavaScript (\*.js) files that are linked from client pages whenever required. These JavaScript files are downloaded on each client machine that consumes ASP.NET AJAX pages.

The ASP.NET AJAX Client architecture includes layers such as Components, Browser Compatibility, Networking, and Core services.

The Components layer consists of elements such as non-visual components and controls.

Client components can implement rich functionality in browsers without using postbacks.

The Networking layer is the key part of the ASP.NET AJAX client library that manages asynchronous calls over XMLHttpRequest. XMLHttpRequest defines an API that allows a browser to communicate to a server without requiring a postback of the entire Web page.

The client library implements object-oriented concepts such as classes, namespace, and inheritance. The library also implements cross browser extensions to data types.

ASP.NET AJAX supports a number of server-based components, including Web services and controls. Server controls look similar to ASP.NET server controls, except that they have additional script code. This script code results in a better user experience by making use of the facilities provided by the AJAX client library.

JSON is one of the core components of the ASP.NET AJAX architecture. JSON is an emerging technology for passing structured data across the Web. It is a text-based data interchange format that can be easily read by humans and parsed by machines. JSON describes data using data structures such as collections and arrays. It is very popular among AJAX developers.

Almost all AJAX-based Frameworks including ASP.NET AJAX implement a JSON infrastructure.

AJAX is built-in with ASP.NET 3.5. It enables you to create highly interactive, responsive, and rich Web applications by providing:

#### → **JavaScript Type System**

The Microsoft AJAX Client Library adds a type system to the JavaScript objects so that ASP.NET AJAX applications can be developed in a structured way. This results in support for classes, namespaces, interfaces, and functionalities such as inheritance and reflection.

#### → **Server Control Integration**

Server controls such as ScriptManager, UpdatePanel, and so forth enable you to make ASP.NET applications AJAX-enabled.

→ **Cross-Browser Support**

ASP.NET AJAX enables browser compatibility and allows you to develop applications for most of the popular browsers.

→ **Rich UI Support**

ASP.NET AJAX provides a number of controls and components to help you develop applications with rich UIs.

Some of the key features of ASP.NET 3.5 AJAX are as follows:

- Support for using UpdatePanel with WebPart controls
- Support to use the ASP.NET Profile, Role, and Login application services using JavaScript
- Ability to have server-side history management
- New server control extenders that enable you to add AJAX/JavaScript functionality to existing controls
- Support for exposing Web service methods on the server that can be called and accessed using client side scripts

### 23.4 ASP.NET AJAX Extensions

A set of controls and services that include AJAX support is called ASP.NET AJAX Extensions. These Extensions provide AJAX support for server-side application development.

ASP.NET AJAX Extensions allow you to simulate AJAX behavior on existing Web sites.

In ASP.NET 3.5 with Visual Studio 2008, the extensions are built-in and you need not download them separately. If you open the Toolbox, you will see the list of available server extension controls under the AJAX Extensions tab.

ASP.NET AJAX Extensions are categorized into three areas namely, server controls and components, Web services bridge, and application services bridge.

→ **Server Controls and Components**

ASP.NET AJAX Extensions have features implemented in a number of namespaces. Table 23.1 describes these namespaces.

| Namespace                       | Description                                                                                                                            |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| System.Web.Configuration        | Defines classes to configure the ASP.NET 2.0 AJAX Extensions programmatically.                                                         |
| System.Web.Handlers             | Defines HTTP Handler classes for processing HTTP requests to a Web server.                                                             |
| System.Web.Script.Serialization | Defines classes providing JSON serialization and deserialization for managed types.                                                    |
| System.Web.Script.Services      | Offers attributes to customize Web service support for the ASP.NET 2.0 AJAX Extensions.                                                |
| System.Web.UI                   | Offers classes and interfaces that enable client-server communication and rich UIs through the use of the ASP.NET 2.0 AJAX Extensions. |
| System.Web.UI.Design            | Offers classes that you can use to extend design-time support for ASP.NET 2.0 AJAX Extensions.                                         |

Table 23.1: Namespaces for ASP.NET AJAX Extensions

ASP.NET AJAX Extensions provide AJAX capabilities to new and existing Web sites. Some of the client and server components of the ASP.NET AJAX Extensions Framework are listed in table 23.2.

| Name                        | Description                                                                                                                                                                                                       |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MicrosoftAjax.js            | Includes functions to extend JavaScript with object-oriented types and constructs.                                                                                                                                |
| System.Web.UI.ScriptManager | Manages and organizes the download of proper JavaScript files and client-side data. These include the AJAX library, proxy classes for remote services, localized version of script files, and globalization data. |
| System.Web.UI.UpdatePanel   | Specifies a region of a page that can be refreshed through an AJAX postback without posting back the entire page.                                                                                                 |
| System.Web.UI.Timer         | Server control that performs postbacks at specified intervals.                                                                                                                                                    |

Table 23.2: Client and Server Components of the ASP.NET AJAX Extensions Framework

#### → Web Services Bridge

Web applications are provided access to only the local resources that are in the context of the application, such as CSS files or images. For all other resource types, the applications have limited access. ASP.NET AJAX server Extensions provide a workaround for this by providing a Web services bridge. This bridge allows you to call external Web services from within client-side scripts.

#### → Application Services Bridge

Earlier, developers had to write long lines of code to implement functionality for services such as authentication and user profiles within AJAX applications. Now, with the application services bridge, these tasks have become easier. The application services bridge enables you to include ASP.NET application services such as authentication and user profiles in AJAX applications.

## 23.5 Creating ASP.NET AJAX Applications

When you select the New Web Site or New Project option in Visual Studio 2008, no template is displayed specifically to create an AJAX-enabled Web site though there is support to create an ASP.NET AJAX server control or server control extender as shown in figure 23.4.

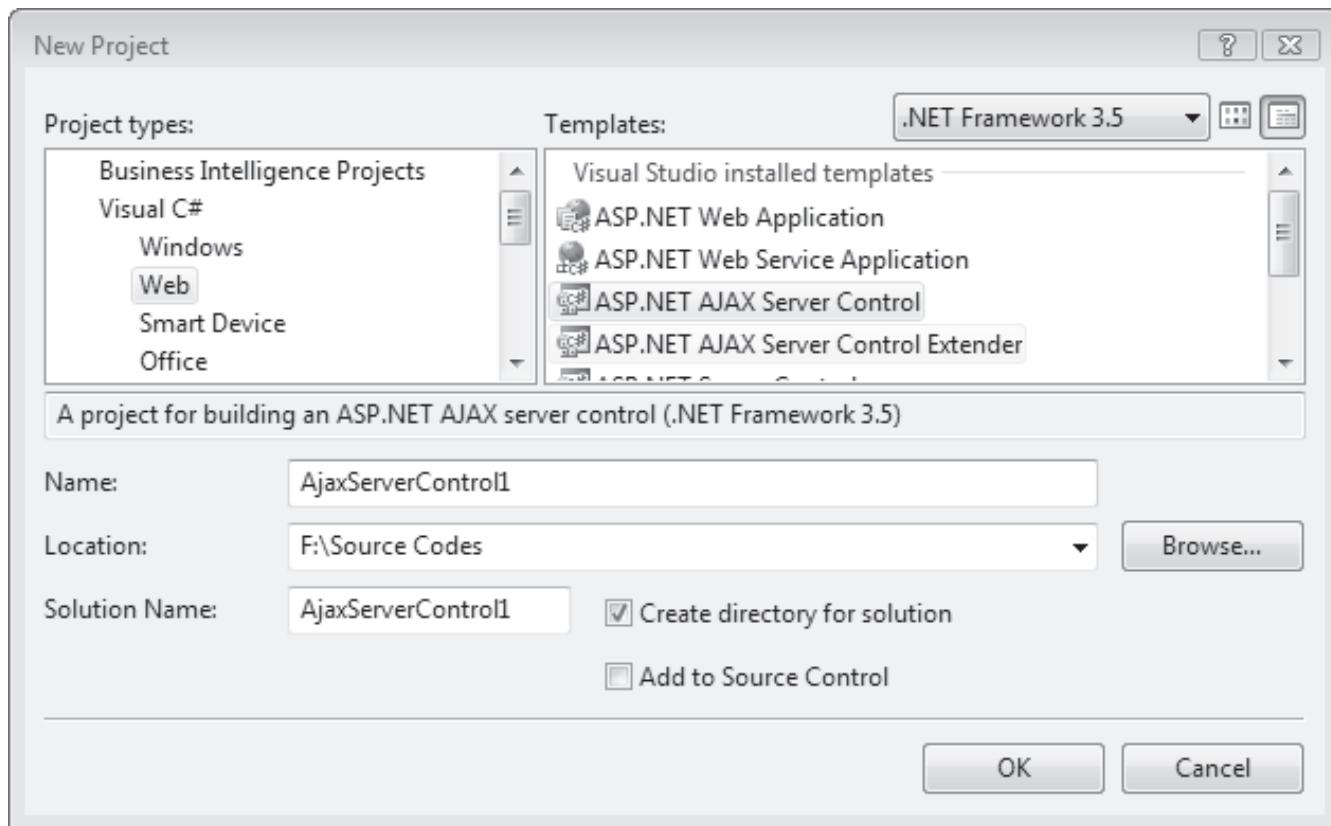


Figure 23.4: ASP.NET Project Templates

Apart from using **New Project** option, you can also create a new Web site by either selecting **ASP.NET Web Site** or **Empty Web Site** option in the **New Web Site** dialog box. Once the Web site is created, you need to add AJAX capabilities to it by either writing code or by adding AJAX Server Extensions such as the **ScriptManager** control.

Adding a **ScriptManager** control to a Web Form that has been created in an empty ASP.NET Web site automatically adds a **web.config** file. Alternatively, if you add a reference to the **System.Web.Extensions.dll** file in the project, it will also add the **web.config** file.

The **web.config** file will have AJAX related sections to register namespaces and assemblies.

If the **web.config** file already exists in the application before **ScriptManager** is added, there is no need to change it after the control is added. This is because the **web.config** file will have AJAX related sections added to it by default.

The following code snippet shows the auto-generated markup in the **web.config** file for the AJAX-enabled application that has the **web.config** file already existing (such as an application which is not an empty Web site). This code registers the **System.Web.Extensions** assembly with the **asp** prefix.

After registration, any control from the `System.Web.Extensions` assembly can be used with the `asp` prefix.

#### Code Snippet:

```
<controls>
 <add tagPrefix="asp" namespace="System.Web.UI" assembly="System.Web.
 Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken =
 31BF3856AD364E35"/>
 <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
 assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
 PublicKeyToken=31BF3856AD364E35"/>
</controls>
```

The following code snippet shows the auto-generated markup in the .aspx file for the AJAX-enabled application that does not have the `web.config` file existing. This code registers the `System.Web.Extensions` assembly with the `asp` prefix. After registration, any control from the `System.Web.Extensions` assembly can be used with the `asp` prefix.

#### Code Snippet:

```
<%@ Register Assembly="System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" Namespace="System.Web.UI"
TagPrefix="asp" %>
```

To make the application AJAX-enabled, you need to add ASP.NET AJAX Extensions controls. The steps to do the same are as follows:

1. First, switch to **Design view**.
2. Double-click the **ScriptManager** control on the **Toolbox** under **AJAX Extensions** tab to add it to the page.
3. Rename the control as **scriptmgrClock**.

The following code snippet shows the markup code to create an instance of `ScriptManager` control named `scriptmgrClock`. The control name is prefixed with `asp` in the start and end tags.

#### Code Snippet:

```
<form id="frmBook" runat="server">
 <asp:ScriptManager ID="scriptmgrClock" runat="server">
 </asp:ScriptManager>
</form>
```

You can replace `asp` in the tag prefix with any other word, however, you must ensure that the same word is used consistently throughout the markup. For example, in the first Code Snippet, if you had used `cc` instead of `asp`, in the third Code Snippet, the code to create `ScriptManager` control

would be prefixed with `cc.`

4. Lastly, on the Toolbox, under AJAX Extensions tab, double-click the `UpdatePanel` to add it to the page. Rename the control as `updPnlClock`. The `UpdatePanel` control performs partial-page updates and identifies content which will be refreshed independently of the rest of the page.
5. Drag and drop a `Timer` control inside the `UpdatePanel` control and rename it as `tmrClock`.

As a result of above actions, the markup for the form will be as shown in the following Code Snippet:

**Code Snippet:**

```
<form id="frmClock" runat="server">
<div style="width: 555px; height: 231px;">
 <center>Digital Clock with two versions of time</center>
 <asp:ScriptManager ID="scriptmgrClock" runat="server">
 </asp:ScriptManager>
 <asp:UpdatePanel ID="updPnlClock" runat="server">
 <ContentTemplate>

 <asp:Timer ID="tmrClock" runat="server" Interval="1000"
 ontick="tmrClock_Tick">
 </asp:Timer>

 </ContentTemplate>
 </asp:UpdatePanel>

</div>
</form>
```

Now, the Web site is AJAX-enabled. To view its capabilities you will need to add some ASP.NET server controls. This is done using following steps:

1. Add a `Label` control in the editable area of the `UpdatePanel` control and configure its properties as shown in the following Code Snippet:

**Code Snippet:**

```
<asp:Label ID="lblUpdatedTime" runat="server"
 BackColor="#99FFCC"></asp:Label>
```

2. Then, in **Code View**, set the `Label` to the current time as follows in the `Page_Load` event handler:

```
lblUpdatedTime.Text = DateTime.Now.Second.ToString();
```

3. Generate the `Tick` event for the `Timer` control and add code as follows:

**Code Snippet:**

```
protected void tmrClock_Tick(object sender, EventArgs e)
{
 lblUpdatedTime.Text = DateTime.Now.Second.ToString();
}
```

4. Lastly, click **Debug** menu and select **Start Debugging** to view the page in a browser.

When the application is executed, there will be no page flash when the time changes. This is because the page is not performing a postback and hence, not updating the whole page every time.

## 23.6 Partial Page Rendering

AJAX provides a feature known as partial page rendering. With partial page rendering, you can send specific portions of a page instead of sending the whole page. Also, you can send it asynchronously. This feature of partial page rendering or updates greatly nullifies the flickering effect and enables you to reduce the load on the server with better bandwidth utilization. It gives the users a better Web experience.

### 23.6.1 ScriptManager Control

The `ScriptManager` control is the heart of an AJAX-enabled Web page. The `ScriptManager` control is responsible for delivering the client script to the browser asynchronously and also enables you to perform partial page update. When a page contains multiple updatable panels, it is the `ScriptManager` control which manages partial page rendering. The `ScriptManager` control communicates with the events in the page life cycle, and achieves partial updates asynchronously.

The default value of the `EnablePartialRendering` property of the `ScriptManager` control is set to `true`. This enables the Web sites to make asynchronous call to the server and render information through partial postbacks. You must load the `ScriptManager` control before loading server controls on a Web page.

The markup syntax to create a `ScriptManager` control is as follows:

**Syntax:**

```
<asp:ScriptManager ID=<id> runat="server">
```

```
</asp:ScriptManager>
```

where,

<id> is the ID property value of the control.

When a ScriptManager control is dropped on a page, the following code is auto-generated:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

The code defines a ScriptManager with ID as **ScriptManager1**, and runat property set to server.

Table 23.3 lists the properties of the ScriptManager control.

Property	Type	Description
AllowCustomErrorsRedirect	bool	Enables you to handle the custom error section of the web.config file.
AsyncPostBackErrorMessage	String	Enables you to retrieve or assign error messages that will be sent to the client if an error is raised.
AsyncPostBackTimeout	Int32	Retrieves or sets the amount of time (in seconds) a client has to wait for an asynchronous request to complete.
ScriptLoadTimeout	Int32	Determine the amount of time required for loading scripts into the client.
IsDebugEnabled	bool	Determines whether debugging is enabled.
IsInAsyncPostback	bool	Determines whether the page is requested using an asynchronous postback mode.
Scripts	CollectionBase <Script Reference>	Retrieves a collection of script references which are sent to the client.
SupportsPartialRendering	bool	Retrieves or sets a value to indicate that all requests made by the client are partial updates (if set to true) or standard postbacks (if set to false).

Table 23.3: Properties of ScriptManager Control

The SetFocus(string) method of ScriptManager control sets the focus to a particular control after a request has completed.

The following code snippet demonstrates some of the properties of a `ScriptManager` control:

**Code Snippet:**

```
<asp:ScriptManager
ID="scriptmgrEmp" runat="server"
AsyncPostBackErrorMessage="There is an error" AsyncPostBackTimeout="1000" >
</asp:ScriptManager>
```

The following code snippet demonstrates some of the code-only properties of a `ScriptManager` control:

**Code Snippet:**

```
protected void Page_Load(object sender, EventArgs e)
{
 scriptmgrEmp.IsInAsyncPostBack == true;
 scriptmgrEmp.SupportsPartialRendering == true;
}
```

### 23.6.2 UpdatePanel Control

The `UpdatePanel` control defines an area on the page in order to contain controls taking part in partial page rendering. The `UpdatePanel` control allows you take advantage of partial page rendering without writing any client script.

To use the `UpdatePanel` control, you need to include a `ScriptManager` control in the page. You will place all the controls that require partial page updates within an `UpdatePanel` control.

There is no limitation on the number of `UpdatePanel` controls a page can have. They can also be nested. Each of the `UpdatePanel` controls can work independently, rendering different parts of the page.

The `UpdatePanel` control consists of control triggers. Any control within the `<ContentTemplate>` tag of the `UpdatePanel` control that creates a postback is a control trigger.

The following code snippet illustrates the `UpdatePanel` control:

**Code Snippet:**

```
<asp:UpdatePanel ID="updPnlClock" runat="server">
<ContentTemplate>
<asp:Label ID="lblTime" runat="server" Text="">
</asp:Label>
<hr />
<asp:Button ID="btnSubmit" runat="server" Text="Submit" Height="24px"
Width="59px" />
</ContentTemplate>
</asp:UpdatePanel>
```

The code creates an `UpdatePanel` control with a label, and a button control embedded inside the `ContentTemplate` of the `UpdatePanel`.

Consider that a page contains multiple `UpdatePanel` controls. When the trigger event of one of the `UpdatePanel` fires, all the `UpdatePanel` controls on the page get updated. This is because, by default, `UpdateMode` property of the `UpdatePanel` is set to `Always`.

If you set the property to `Conditional`, it will refresh the `UpdatePanel` only when a specific trigger is fired.

Table 23.4 lists some of the markup-enabled properties of the `UpdatePanel` control.

Property	Type	Description
<code>ChildrenAsTriggers</code>	<code>bool</code>	This property, if set to true, enables the child controls to trigger a refresh on postback.
<code>RenderMode</code>	<code>Enum (Block, Inline)</code>	This property describes the way the content will be displayed.
<code>UpdateMode</code>	<code>Enum (Always, conditional)</code>	This property specifies whether the <code>UpdatePanel</code> refreshes during a partial page update or refreshes only when a specific trigger is hit.
<code>IsInPartialRendering</code>	<code>bool</code>	This property checks whether the <code>UpdatePanel</code> control supports partial rendering for the current request.
<code>ContentTemplate</code>	<code>ITemplate</code>	This property gets or sets the content inside the <code>UpdatePanel</code> control.
<code>ContentTemplateContainer</code>	<code>Control</code>	This property represents the template container for updating request

Property	Type	Description
Triggers	UpdatePanelTriggerCollection	This property will get the list of triggers associated with the current UpdatePanel control

Table 23.4: Properties of ScriptManager Control

The following code snippet shows some of the markup enabled properties of the UpdatePanel control:

**Code Snippet:**

```
if (updpnlClock.IsInPartialRendering)
{
 // Do actions
}
```

### 23.6.3 Timer Control

The ASP.NET AJAX `Timer` control performs a postback at specific intervals of time. It is usually used with an `UpdatePanel` control in order to enable partial rendering of the contents in the panel at specific intervals of time. You can also use the `Timer` control for synchronous postbacks of the complete page in definite intervals of time.

### 23.6.4 Adding a Client Control

You can add an AJAX client control by choosing **WebSite → Add New Item** and selecting the AJAX Client Control template.

Figure 23.5 shows the AJAX Client Control template being selected.

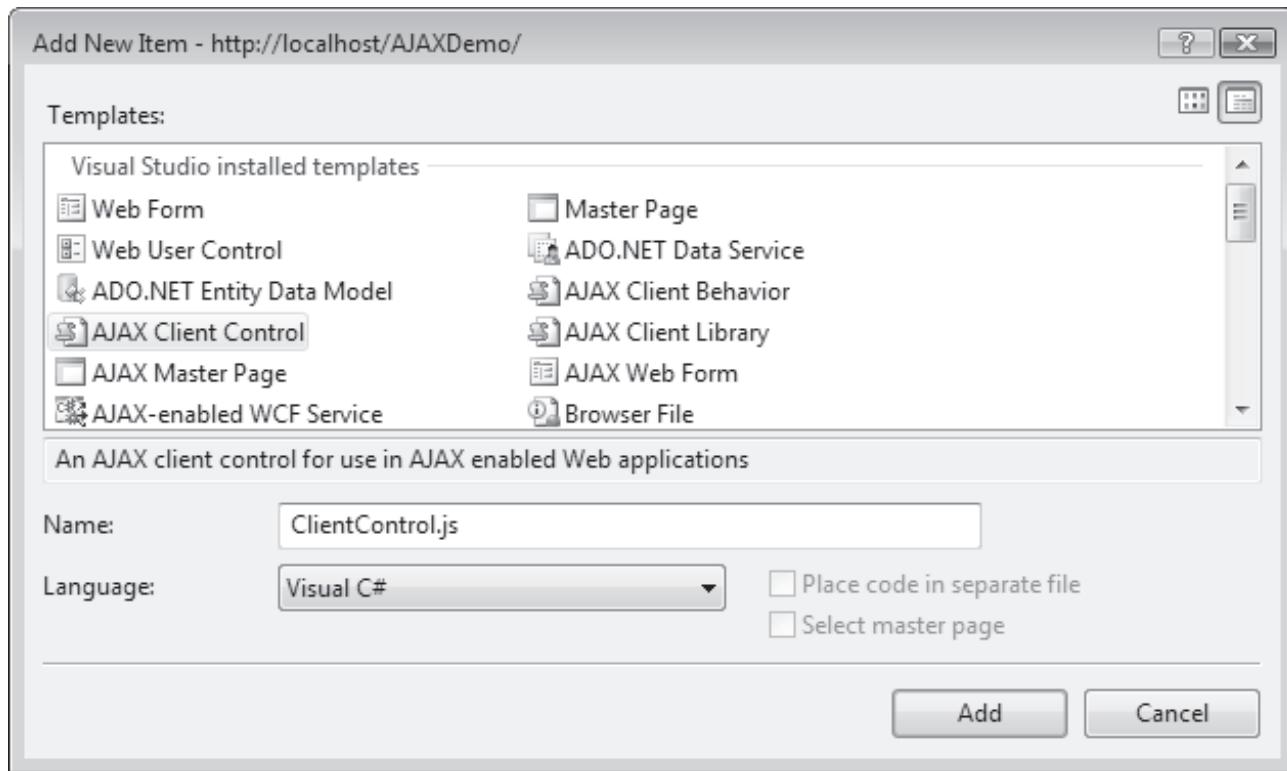


Figure 23.5: AJAX Client Control Template being Selected

### 23.6.5 PageRequestManager Class

The Microsoft AJAX Library provides a rich framework to simplify development of client-side applications. The `Sys` namespace is the root namespace in this library.

The `Sys.WebForms.PageRequestManager` class is responsible for partial postback. This class is not instantiated directly; its instance is available whenever partial-page rendering is enabled in a Web application. You can call the `getInstance()` method to get the instance of the `PageRequestManager` class.

The `PageRequestManager` class provides you with properties, methods, and events to enable partial page rendering. The code for the events is placed in the `.js` file of the client control.

Some events of `PageRequestManager` class are as follows:

→ `initializeRequest`

This event is triggered during initialization of an asynchronous postback, but before the asynchronous request begins. You can add handlers to this event like you do for any other event. These handlers will be called after the `initializeRequest` event is invoked.

The following code snippet attaches a handler to the `initializeRequest` event:

**Code Snippet:**

```
Sys.WebForms.PageRequestManager.getInstance().add_initializeRequest(
 Initialize);
function Initialize(sender, args)
{
 alert("The Request that was sent is getting initialized");
}
```

→ `beginRequest`

This event is triggered after the `initializeRequest` event. This event is also triggered before a partial page update, and before sending any data to the server for processing. You can add handlers to this event as you do for any other normal event.

The following code snippet attaches a handler to the `beginRequest` event:

**Code Snippet:**

```
Sys.WebForms.PageRequestManager.getInstance().add_beginRequest(
 BeginRequest1);
function BeginRequest1(sender, args)
{
 alert("Request is ready to be sent to server");
}
```

→ `pageLoading`

This event is triggered after the server processes the request, but before loading the page.

The response object, which is received from the server, holds a reference to panels which will get modified. If you want any animation or transition to happen before the updated data is placed in their respective positions, you can use this event.

The following code snippet attaches a handler to the `pageLoading` event.

**Code Snippet:**

```
Sys.WebForms.PageRequestManager.getInstance().add_pageLoading(
 PageLoading1);
function PageLoading1(sender, args)
{
 alert("My page is started loading");
}
```

→ pageLoaded

This event is triggered when all the contents of the page is updated. Now, if you want to clear certain animation or effects on the page which was initiated in the beginRequest, it can be done in this event. You can also add animation and graphics here in this event, instead of adding them in the pageLoading event.

The following code snippet attaches a handler to the pageLoaded event:

**Code Snippet:**

```
Sys.WebForms.PageRequestManager.getInstance().add_pageLoaded
(PageLoaded1);

function PageLoaded1(sender, args)
{
 alert("My page is loaded");
}
```

→ endRequest

This event is the last event which is triggered in the client postback cycle. This event fires irrespective of the postback being a success or failure like finally in a try...catch...finally block. It is used to notify the status of the request to the user.

The following code snippet attaches a handler to the endRequest event:

**Code Snippet:**

```
Sys.WebForms.PageRequestManager.getInstance().add_endRequest
(EndRequest1);

function EndRequest1(sender, args)
{
 alert("My Request has ended");
}
```

## 23.7 The AJAX Control Toolkit

The AJAX Control Toolkit was developed by ASP.NET community under an open source domain. The Toolkit consists of a number of controls that help to provide AJAX functionality in your Web sites. CSS is a style sheet language supported by most browsers and is an important part of AJAX. CSS is used to create and apply styles to the controls in the Toolkit.

The AJAX Control Toolkit is not built-in with Visual Studio 2008. You need to download and install it in order to use the controls in the Toolkit. Once it is installed, you need to add the .dll of the Toolkit to your Toolbox so that you can use the controls in the IDE.

The controls in the Toolkit, which extend existing ASP.NET server controls, are called extender controls or control extenders. The Toolkit also contains script controls and client components. The best thing about these controls is that they are very easy to use. There is no need to write lengthy JavaScript code to use these controls.

It is the responsibility of an extender control to associate the client component containing the new functionality with the server control. For example, an `AutoCompleteExtender` extends the server control, Text box, to enable auto-completion features. Properties of the `AutoCompleteExtender` control will be set in order to associate it with the Text box.

The `ToolkitScriptManager` control must be added to the Web Form before you can add any controls from the AJAX Control Toolkit. The `ToolkitScriptManager` control is also present in the AJAX Control Toolkit and can be added through the Toolbox. Some extender controls invoke methods defined inside Web services to implement the functionality. The path of the Web service and the name of the method are passed to the server control through properties.

The following code snippet shows the code that is auto-generated in the `.aspx` file when you add an extender control to a Web page for the first time. This code registers the assembly, namespace, and tag prefix required to use AJAX Control Toolkit.

### Code Snippet:

```
<%@ Register Assembly="AjaxControlToolkit"
 Namespace="AjaxControlToolkit"
 TagPrefix="ajaxToolkit"
%>
```

The tag prefix used in the `Register` directive is an alias for the namespace. Any code that later defines controls from the Toolkit can make use of this prefix.

The following code snippet shows an example of this:

**Code Snippet:**

```
<ajaxToolkit:AutoCompleteExtender
 runat="server"
 ID="autocomplexSearch"
 ...
 ...
</ajaxToolkit:AutoCompleteExtender>
```

The Toolkit consists of various extender controls, which can be broadly classified into two categories, as shown in table 23.5.

Category	Examples of Control	Purpose
Text Input	AutoCompleteExtender, TextBoxWaterMarkExtender, PasswordStrength, and DynamicPopulateExtender	These controls are used to extend along with text input controls like TextBox, DropDownList, and so forth
User Interface	Accordion, CascadingDropDown, ConfirmButtonExtender, AlwaysVisibleControlExtender, Rating, and TabContainer	These controls enable you to create a better UI

Table 23.5: Extender controls in the Toolkit

### 23.8 Controls in the AJAX Control Toolkit

AutoCompleteExtender is a control extender in the AJAX Control Toolkit that helps you achieve the functionality described earlier. It is attached to a `TextBox` to enable word suggestion for a particular field. The control enables a panel to appear having words prefixed with the text typed in the Text box.

A `TextBox` control can be extended with the `AutoCompleteExtender` control.

Some of the properties of this control are explained as follows:

- **MinimumPrefixLength:** This property specifies the minimum number of characters required in the Text box to get suggestions.
- **ServicePath:** This property specifies the path where the `AutoCompleteExtender` control will search for getting the word suggestions.
- **ServiceMethod:** This property specifies the method, which needs to be called for word suggestions.
- **TargetControlID:** This property specifies the control to which the `AutoCompleteExtender` is attached.

An example of an `AutoCompleteExtender` control is shown in the following Code Snippet:

**Code Snippet:**

```
<cc1:AutoCompleteExtender
ID="autocomplexSearch"
runat="server"
MinimumPrefixLength="2"
Enabled="True"
ServicePath("~/AutoCompleteService.asmx"
ServiceMethod="Search"
TargetControlID="txtSearch">
</cc1:AutoCompleteExtender>
```

In this code snippet, `MinimumPrefixLength` is set to 2, `ServicePath` points to a Web service named **AutoCompleteService** that is present in the same directory, `serviceMethod` name is given as **Search**, and `TargetControlID` is set as **txtSearch**.

A `ConfirmButtonExtender` enables you to confirm a client interaction, like a click of a button, with some custom message.

Table 23.6 describes some of the properties of a `ConfirmButtonExtender` control.

Property	Description
TargetControlID	This property specifies the control that is to be extended with <code>ConfirmButtonExtender</code> control.
ConfirmText	This property enables you to specify the confirmation text that needs to be displayed.

Table 23.6: Properties of `ConfirmButtonExtender`

The following code snippet creates a `ConfirmButtonExtender` control:

**Code Snippet:**

```
<cc1:ConfirmButtonExtender ID="cnfbtnexSubmit" runat="server"
ConfirmText="Are you sure you want to exit?"
Enabled="True"
TargetControlID="btnSubmit">
```

ASP.NET AJAX applications can also work with XML Web services.

### 23.9 XML Web Services

An XML Web service is an independent component that communicates with other applications using standard protocols over the network. XML Web services work with standard Web protocols such as HTTP, XML, and Simple Object Access Protocol (SOAP). This allows XML Web services to communicate with applications written in different languages, operating on different platforms, and communicating with different protocols. Web services created using ASP.NET are called XML Web services as they communicate on the network by exchanging messages in XML format.

Figure 23.6 illustrates the basic concept of XML Web services.

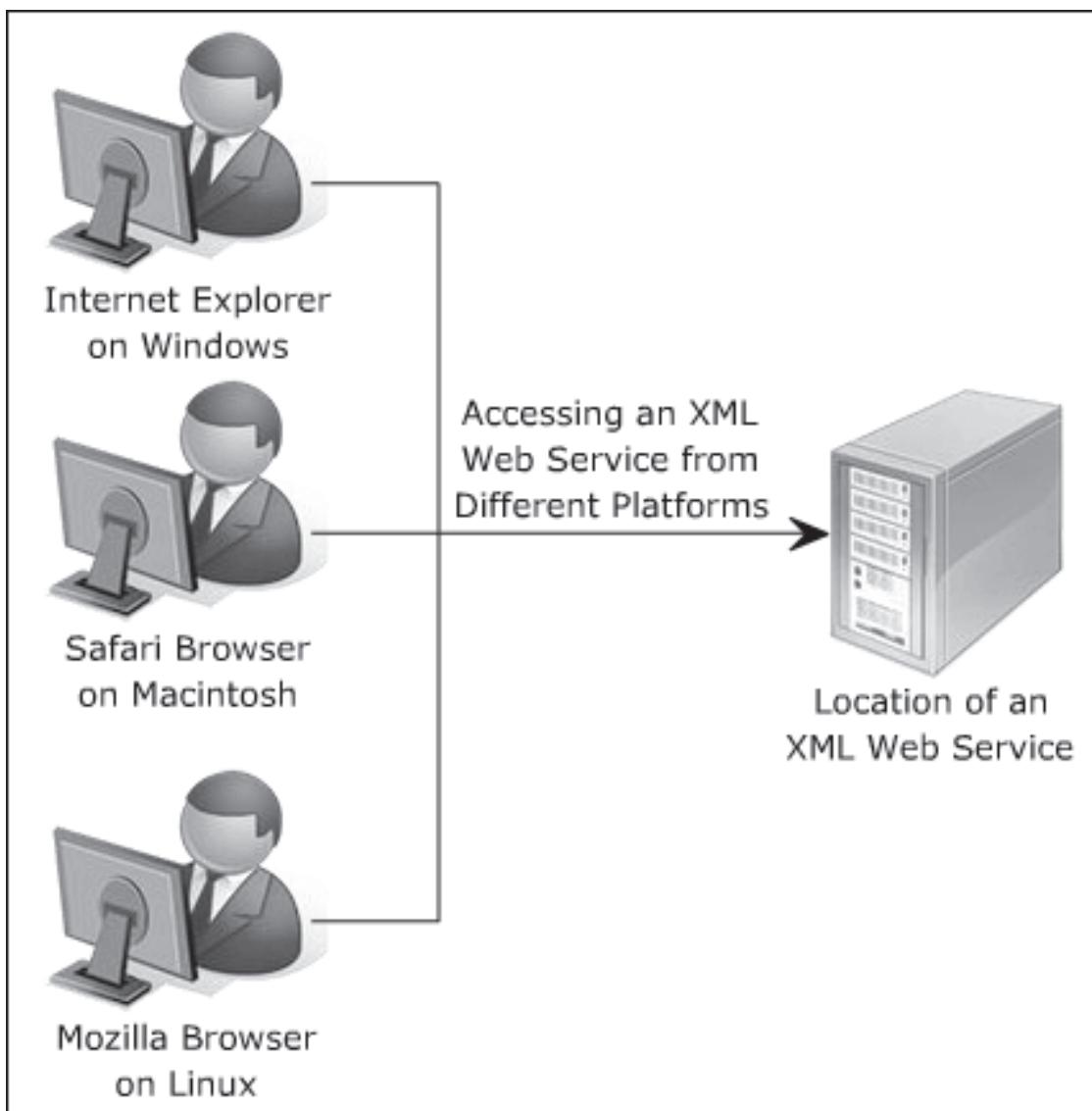


Figure 23.6: XML Web Services

Apart from the feature of interoperability, Web services provide some more features.

These features include:

- Stateless architecture
- Asynchronous architecture
- Platform and language-independent communication

XML Web services register themselves on a central network node to make the services available to users. The node is also known as Universal Description, Discovery, and Integration (UDDI) registry. After registration, Web clients locate XML Web services by using an XML document called the Web services Description Language (WSDL), and the Discovery (DISCO) specification.

The WSDL document provides the description about the Web service, whereas the DISCO specification provides algorithms to locate the Web service. All these operations are conducted using the XML Web services infrastructure.

Web services have many advantages as compared to traditional distributed computing technologies. The advantages of Web services include:

- Allowing cross business integration by providing access to third-party softwares and legacy applications irrespective of their platform and programming language. Legacy applications are applications created using software languages, platforms, and techniques introduced earlier than the current technology. They are mainly those applications in which a firm has invested reasonable time and money.
- Providing increased efficiency, scalability, and maintainability as applications are split into independent smaller components.
- Reducing complexity by providing encapsulation, which frees the clients from knowing the Web service architecture.
- Ensuring interoperability by using common Web standards.
- Providing on-demand integration of distributed components.

### 23.9.1 Calling an XML Web Service

In general, a Web client must perform a few steps to call a Web service. These steps include:

The client creates an object of the XML Web service proxy class. The proxy class is an object on the client computer acting as a representative of the service.

The client calls the method on the proxy class.

The process of calling a Web service is shown in figure 23.7.

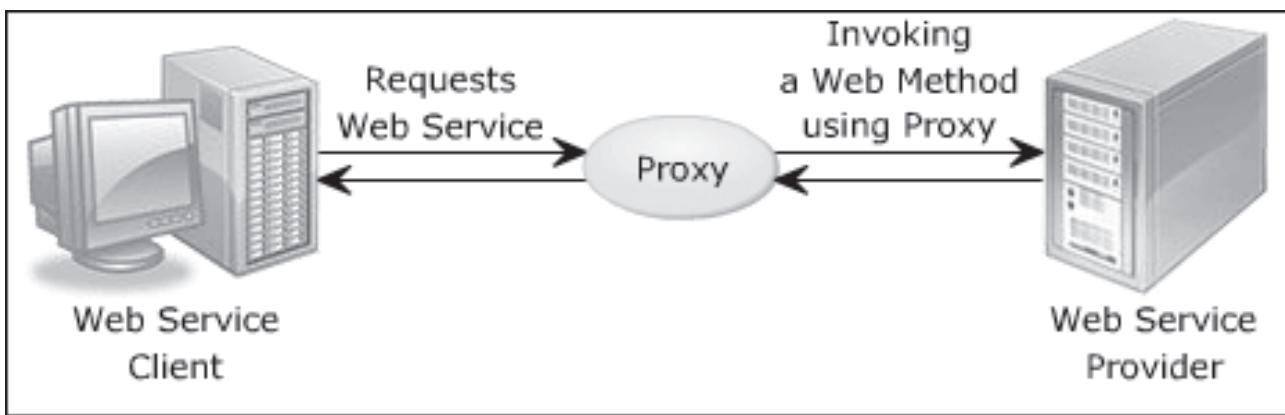


Figure 23.7: Consuming a Web Service

The arguments of the XML Web service method are serialized into a SOAP message by the XML Web service infrastructure on the client computer.

A SOAP message is an XML data representing either the request or response over the network. Serialization is the process of transmitting an object across a network in a binary format or in some other format. Extracting data from a series of bytes is referred to as deserialization.

The arguments are sent to the XML Web service over the network.

The Web server receives the SOAP message and deserializes it. This process of serializing and deserializing is summarized in figure 23.8.

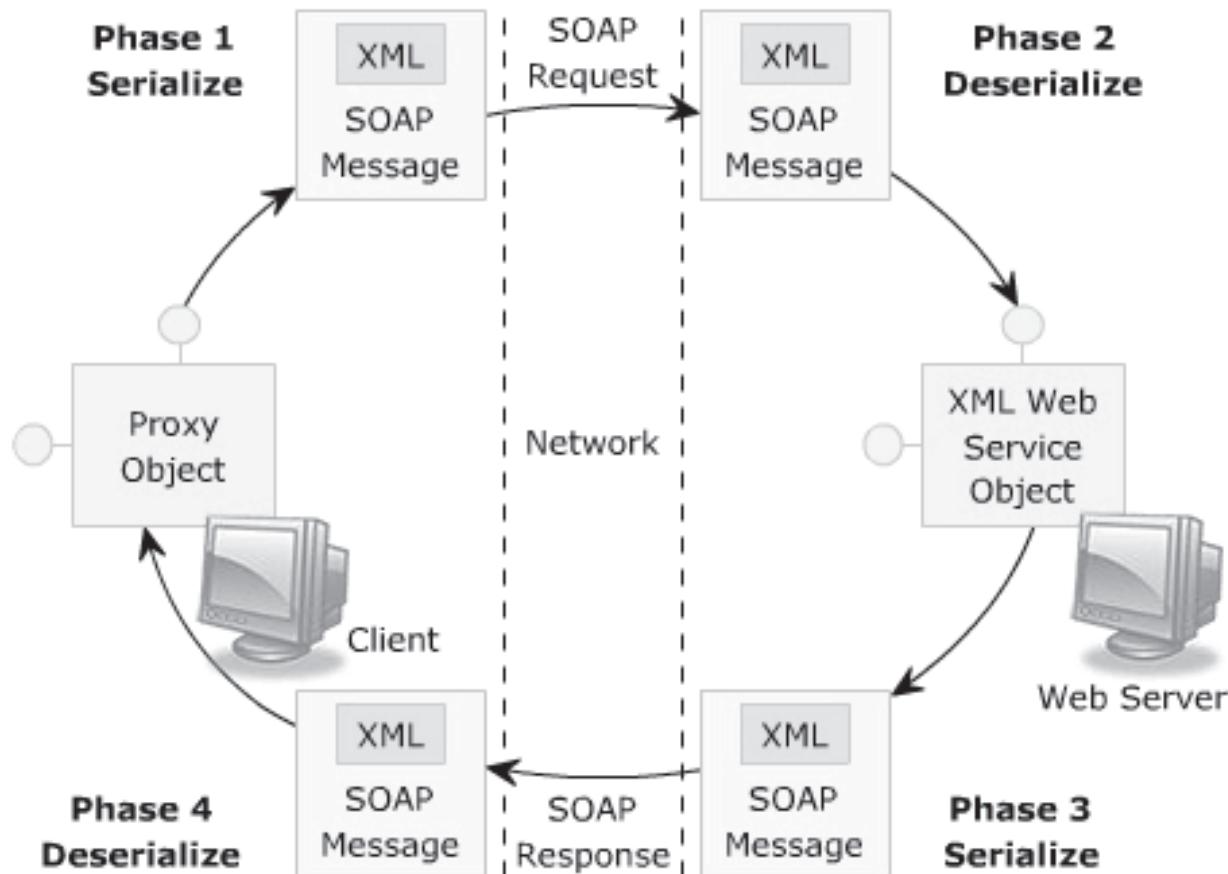


Figure 23.8: Serializing and Deserializing a Web Service

The infrastructure on the server invokes the method by creating an object of the class implementing the service and passes the arguments.

The method is executed and the value is returned to the client using the same process of sending the request message.

### 23.9.2 Working with Web Services

An ASP.NET Web service consists of an **.asmx** file that resides on the IIS. The file defines either a class providing the Web service functionality in form of methods, or refers to some other class that may contain the functionality.

Any method in the file that needs to be invoked remotely must contain the `[WebMethod]` attribute.

First, the client application collects the information or type description about the Web service, which needs to be called. The information is retrieved as an XML file, which is the WSDL file. This file describes the Web service and it is returned by the Web service broker. This is known as locating the Web service.

Now, the client sends the required parameters to the Web server using SOAP for calling the required Web

service. Then, the server returns the results using SOAP to the client.

Figure 23.9 depicts an ASP.NET Web service.

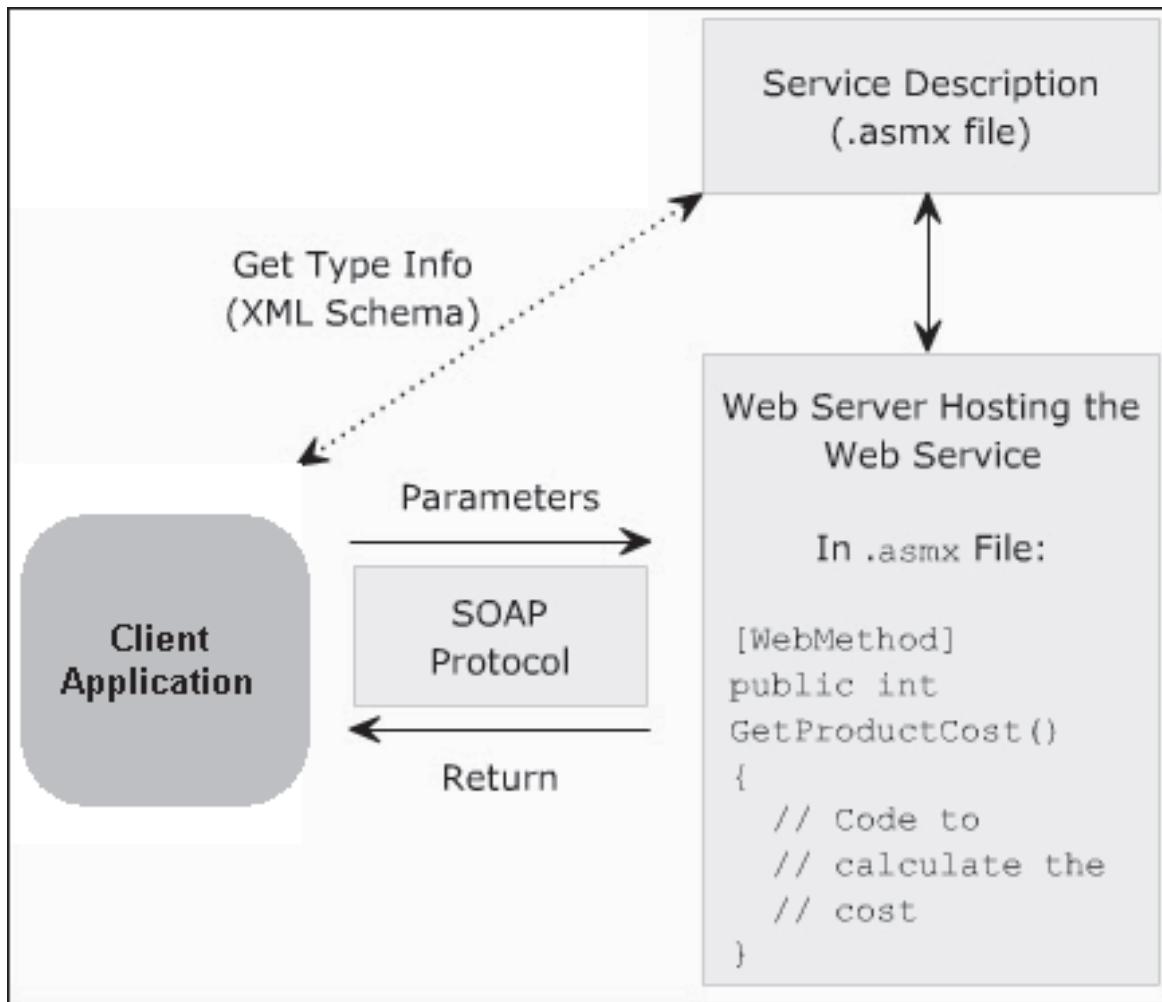


Figure 23.9: ASP.NET Web Service

The `@WebService` directive in an XML Web service defines attributes related to the XML Web service. The attributes are used to specify the programming language, the source file, and the class that implements the Web service. These attributes are defined in the .asmx file and are used by the ASP.NET parser and compiler. There are four attributes that are defined in this directive. These attributes are as follows:

→ Class

Identifies the class implementing the XML Web service. The class can either exist in the same file or in some other file.

If the class is in another file, it must be placed in the **Bin** directory of the Web application where the Web service resides. This attribute is mandatory.

→ **CodeBehind**

Identifies the source file, which implements the XML Web service. This identification is done when the class mentioned in the `Class` attribute exists in some other location, is not placed in the `Bin` directory, and is not compiled into an assembly.

→ **Debug**

Specifies a boolean value indicating whether the XML Web service should be compiled with debug symbols. Debugging symbols provide extra information, which is compiled into a binary file. This file is used to extract information such as the names of variables and methods from the source code during debugging.

→ **Language**

Identifies the language used while compiling the inline code of the `.asmx` file. This value can be any language supported by the .NET Framework such as C#, Visual Basic .NET, and JScript .NET.

The code shown here specifies the language, the source file, and the class implementing the Web service using the appropriate attributes.

```
<%@ WebService Language="C#" CodeBehind("~/App_Code/EmailValidator.cs"
Class="EmailValidator" %>
```

The code uses the “@” symbol, which is the standard syntax to declare any directive. The `Language` attribute is specified as C#, which means that the language used to write a Web service is C#. The `CodeBehind` attribute is set to the source file that implements the Web service. The “~” symbol indicates that the file is in the current directory. The `Class` attribute is set to the class that implements the Web service. This class is compiled automatically for the first time when the Web service is called.

A complete example of a Web service class is shown in the following Code Snippet:

**Code Snippet:**

```
[WebService(Namespace = "http://aptech-worldwide.com/webservices")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class TestService : System.Web.Services.WebService
{
 public TestService()
 {
 }
```

```
[WebMethod]
public DataSet UserValidation(string user, string passwd)
{
 string tableName = "Customers";
 ...
 SqlCommand sqlcomUserList = new SqlCommand("SELECT * FROM " +
 tableName + " WHERE UserName=' " + user + "' AND Password=' " + passwd
 + " ' ", sqlconElite);
 SqlDataAdapter sqldaUserList = new SqlDataAdapter(sqlcomUserList);
 DataSet dsetUsersList = new DataSet();
 sqldaUserList.Fill(dsetUsersList, tableName);
 return dsetUsersList;
}
```

The **TestService** class inherits from `System.Web.Services.WebService`. It uses the `[WebMethod]` attribute for the method **UserValidation**. This service class accepts a user name and password for the **UserValidation** method which then, validates them against the values stored in the database.

### 23.9.3 Publishing a Web Service

Consider a scenario of an airline company that has built a Web service, which will be used by traveling agencies across the world to book tickets. The Web service provides methods for checking the arrival and departure of flights and looking for the availability of flights and seats.

Now, the Web service must be accessible by the Web sites of the traveling agencies to invoke the Web methods. To do so, the Web service must be available on the network so that the Web application can locate them. To achieve this task, the Web service must be published on the Internet. By publishing a Web service, you make it available to the Web clients. Once the Web service is published, the users can access it through Web pages to invoke the required methods.

You can publish Web services on various servers such as IIS and Apache. An XML Web service is published on the IIS Web server using following steps:

1. Copy the Web service project folder to the **[Drive:]\\Inetpub\\wwwroot** folder (for example, `c:\\InetPub\\wwwroot`), if you have created it as a local filesystem project.
2. Open **Internet Information Services Manager** from the **Administrative Tools** folder in the **Control Panel**.
3. Expand the **Default Web Site** node, right-click the **Web service Project** folder, and select **Properties**.
4. Click the **Create** button in the **Application Settings** panel of the dialog box.
5. Click **OK**. The virtual folder will be configured as the root of the Web service application.

6. To test the Web service, right-click the **.asmx** file in the Web service folder and select **Browse**. The **.asmx** file will be displayed in the Web browser.

You can publish a Web service either on an Intranet or the Internet. Publishing a Web service on Intranet will limit the number of users accessing it. For example, a Web service published on the Intranet network of an organization will be accessed only by its employees operating within the network. To publish a Web service on Intranet:

1. Open the **Internet Information Services Manager** window.
2. Right-click the **Default Web Site** node in the left pane.
3. Select **New → Virtual Directory**. The **Virtual Directory Creation Wizard** appears.
4. Click **Next**. The **Virtual Directory Alias** screen appears.
5. Enter an appropriate name in the **Alias** text box that will be used to access the virtual directory.
6. Click **Next**. The **Web Site Content Directory** screen appears.
7. Enter the path for the virtual directory.
8. Click **Next**. The **Access Permission** screen appears.
9. Select the appropriate options and click **Next**. The wizard completion screen appears.
10. Click **Finish** to complete the configuration.

#### *23.9.4 Discovering Web Services*

The discovery process determines the information such as location and specifications of the Web service. It also allows you to determine the way you communicate with the XML Web service. The XML Web service automatically generates a discovery document by using the **disco.exe** tool.

To configure discovery information for a Web service:

1. Create an XML document and insert the `<?xml version="1.0" ?>` tag in the first line.
2. Add a `<discovery>` tag.
3. Include references for discovery, contract, and schema for service descriptions, discovery documents, and XML Schema Definition (XSD) schemas within the `<discovery>` tag.
4. Deploy the discovery document by copying it to the virtual directory on the Web server.

The **web.config** file is a configuration file that contains the default configuration settings for XML Web service. It allows you to customize the configuration information for the Web services that are deployed on the Web server. The **web.config** file is stored in a directory that contains the XML Web service application.

### 23.10 Error Handling in XML Web Services

Webservices must also implement some mechanisms for error or exception handling. The `SoapException` class represents an exception that may be generated when a Web service method is invoked via SOAP. `SoapException` is either thrown by Common Language Runtime (CLR) or any Web service method. The `SoapException` is thrown by CLR if there is any problem in the response format. The response format of a SOAP message includes proper implementation of XML tags.

The `SoapException` class is included within the `System.Web.Services.Protocols` namespace. The class consists of overloaded constructors, some of which are as follows:

→ `SoapException ()`

Assigns a new instance of the `SoapException` class.

→ `SoapException (String, XmlQualifiedName, Exception)`

Assigns a new instance of the `SoapException` class with the specified exception message, the type of exception that occurred in the code, and the root cause of the exception.

→ `SoapException (String, XmlQualifiedName, String)`

Assigns a new instance of the `SoapException` class with the specified exception message, the type of exception that occurred in the code, and Uniform Resource Identifier (URI) that recognizes the code that generated the exception. The `XmlQualifiedName` class represents an XML qualified name, which is a qualified namespace existing as `namespace:localname`.

When the XML Web service method is invoked using SOAP, a `SoapException` might occur. The exception details for the `SoapException` are specified using the different properties of the `SoapException` class.

The `SoapException` class consists of properties that allow you to specify information about the generated exception. Table 23.7 lists the properties of `SoapException` class.

Property	Description
Actor	Retrieves the source code that caused the exception.
Code	Retrieves the SOAP fault code type. It can only be set when creating a new instance of the <code>SoapException</code> class.
Detail	Retrieves an <code>XmlNode</code> that provides information about the generated application-specific exception. <code>XmlNode</code> illustrates a single node within an XML document.
Lang	Retrieves the language that is associated with the exception.
Message	Retrieves the message that defines the current exception. This property is derived from <code>Exception</code> class.

Table 23.7: Properties of `SoapException` Class

### 23.11 Check Your Progress

1. Which of the following statements are true about traditional Web-based applications?

(A)	The user has to wait for completion of the postback before further processing can take place.
(B)	Traditional Web-based applications use the synchronous programming approach.
(C)	Traditional Web-based applications have lack of continuous interactivity on Web pages.
(D)	Traditional Web-based applications make use of AJAX.

(A)	a, c, d	(C)	b, c, d
(B)	a, b, d	(D)	a, b, c

2. Which of the following statements are true about ASP.NET AJAX?

(A)	The Microsoft Client Library for AJAX represents the AJAX client architecture.
(B)	The client architecture of ASP.NET AJAX includes Web services, ad hoc controls, and application services.
(C)	The Microsoft AJAX library comprises a set of JavaScript (*.js) files that are linked from client pages whenever required.
(D)	The client library implements object-oriented concepts such as classes, namespace, and inheritance. The library also implements cross browser extensions to data types.

(A)	a, c, d	(C)	a, d
(B)	b, c, d	(D)	b, c

3. \_\_\_\_\_ is one of the core components of the ASP.NET AJAX architecture, is an emerging technology for passing structured data across the Web and is a text-based data interchange format that can be easily read by humans and parsed by machines.

(A)	XML	(C)	CSS
(B)	JASON	(D)	HTTP

4. Arrange in proper order the steps that the Web client must perform to call a Web service.

(A)	a. The client calls the method on the proxy class.
-----	----------------------------------------------------

(B)	The arguments of the XML Web service method are serialized into a SOAP message by the XML Web service infrastructure on the client computer.
(C)	The client creates an object of the XML Web service proxy class.
(D)	The infrastructure on the server invokes the method by creating an object of the class implementing the service and passes the arguments.
(E)	The method is executed and the value is returned to the client using the same process of sending the request message.
(F)	The Web server receives the SOAP message and deserializes it.
(G)	The arguments are sent to the XML Web service over the network.

(A)	c, a, b, g, f, d, e	(C)	a, d, c, b, e, g, f
(B)	a, d, c, b, e, g, f	(D)	a, c, b, d, e, f, g

5. The SoapException class is included within the \_\_\_\_\_ namespace.

(A)	System.Web.Services.Soap	(C)	System.Web.Services
(B)	System.Web.Services.Protocols	(D)	System.Web.Services.Exception

6. A method in the XML Web service (.asmx) file that needs to be invoked remotely must contain the \_\_\_\_\_ attribute.

(A)	[Web]	(C)	[WebService]
(B)	[WebMethod]	(D)	[WebServiceMethod]

## Module Summary

In this module, **ASP.NET AJAX and XML Web Services**, you learnt about:

- **ASP.NET AJAX**
- AJAX is a group of technologies used to develop highly interactive and responsive Web applications. ASP.NET AJAX is an AJAX Framework that enables you to develop highly interactive and efficient Web applications that can target all popular browsers. A set of controls and services that include AJAX support is called ASP.NET AJAX Extensions. These Extensions provide AJAX support for server-side application development. You can add AJAX capabilities to a Web site by either writing code or by adding AJAX Server Extensions such as the ScriptManager control. The AJAX Control Toolkit consists of a number of controls that help to provide AJAX functionality in your Web sites.
- **XML Web services**

An XML Web service is an independent component that communicates with other applications using standard protocols over the network. Web services that are created using ASP.NET are called XML Web services as they communicate on the network by exchanging messages in XML format.

# Module - 24

## ASP.NET AJAX and XML Web Services (Lab)

Welcome to the module, **ASP.NET AJAX and XML Web Services (Lab)**.

In this module, you will learn to:

- Use AJAX Extensions
- Use ASP.NET AJAX Server Controls
- Use ASP.NET AJAX Control Toolkit
- Create and call XML Web services

Web Development

http://www



## 24.1 Part I – 90 Minutes

### Exercise 1

#### **Problem:**

United Research Corporation (URC), a Chicago based company, conducts research on database servers used. They send survey forms to various IT professionals and get feedback on servers used. Now, URC wants to keep up with the modern times and wants to conduct the survey online through a Web site. They want users to have a rich and interactive experience and also expect the Web site to be fast and efficient.

Assume that you are a developer at URC and that you have been given the task of creating the survey application. You must use Visual Studio 2008, ASP.NET 3.5, and AJAX to build this application.

#### Solution:

1. Download the AJAX Control Toolkit from the following link:

<http://ajaxcontroltoolkit.codeplex.com/releases/view/43475>

1. Ensure that the filename is **AjaxControlToolkit.Binary.NET35.zip**.
2. Unzip and install the Toolkit into any folder of your choice.
3. Launch Visual Studio 2008 IDE.
4. Create an ASP.NET Web site named **DataSurvey** on the localhost.
5. Rename **Default.aspx** to **SurveyForm.aspx**. Also, rename all instances of **Default** to **SurveyForm**.

To make the AJAX Control Toolkit controls usable in all ASP.NET projects or Web sites, you need to add them to the Toolbox. You can achieve this using the following steps.

1. Right-click the Toolbox and select Add Tab option as shown in figure 24.1.

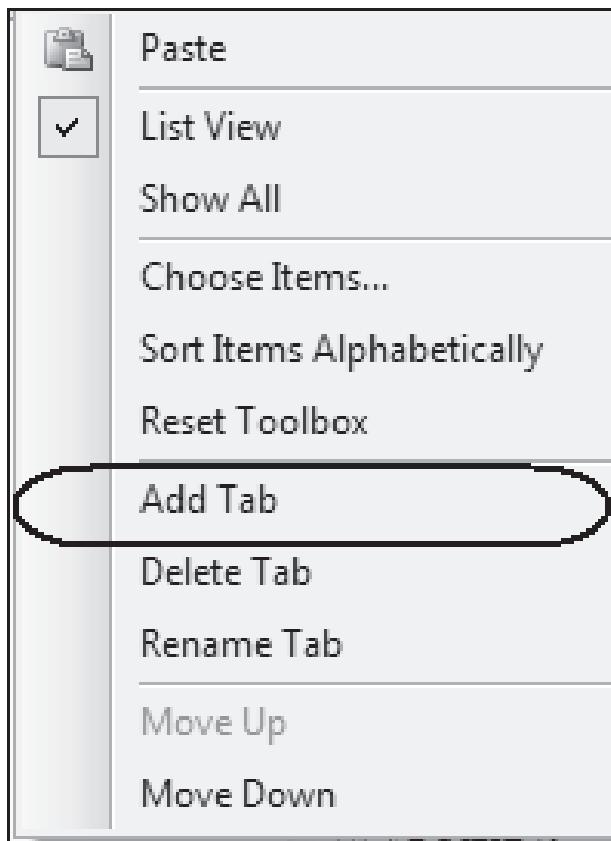


Figure 24.1: Adding a new tab to the Toolbox

2. Rename the blank new tab that is added to the Toolbox as AJAX Control Toolkit.

3. Right-click the tab and select **Choose Items...** command. The **Choose Toolbox Items** dialog box is launched as shown in figure 24.2.

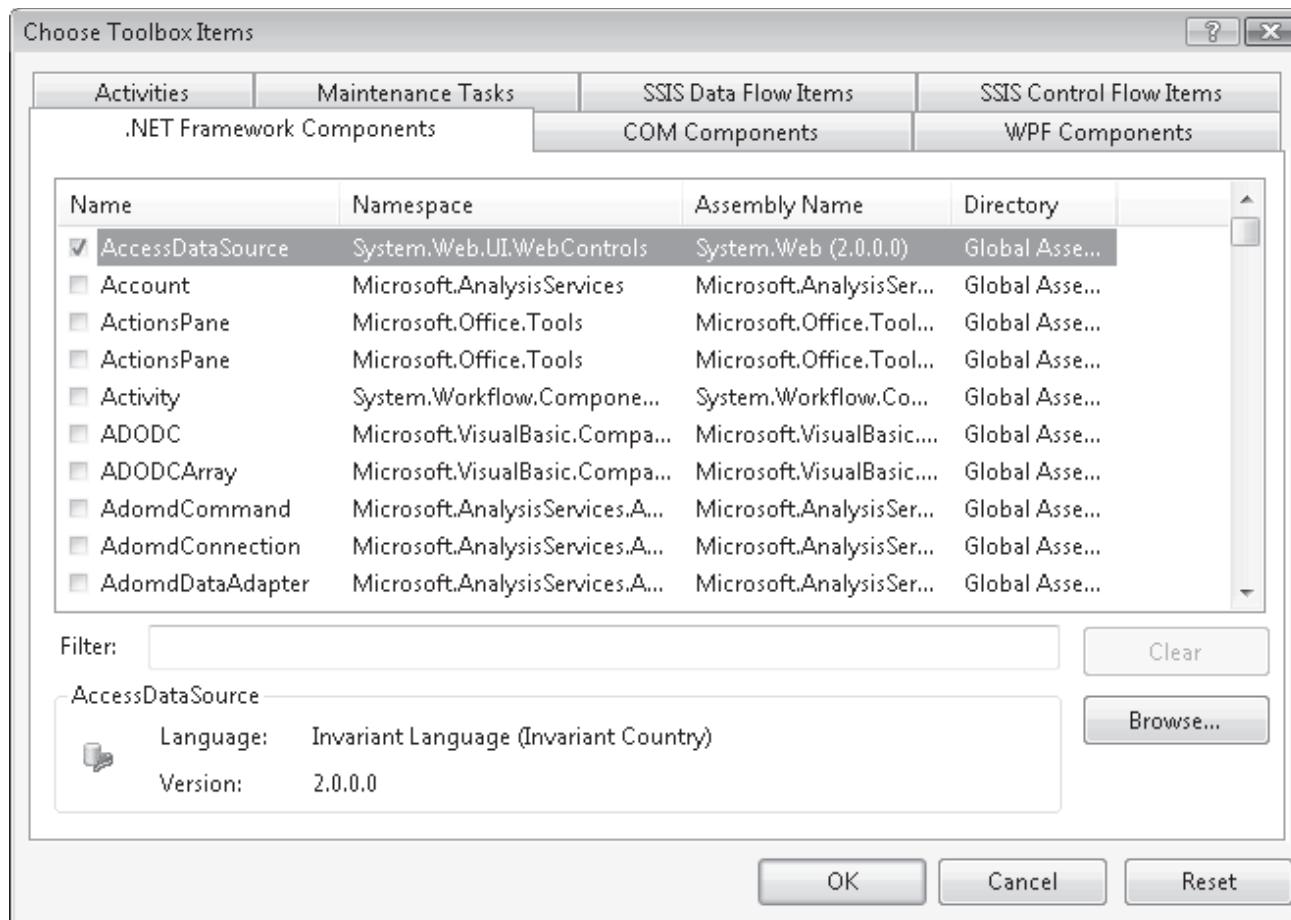


Figure 24.2: Choose Toolbox Items Dialog Box

4. Click **Browse** in the **Choose Toolbox Items** dialog box and select the **AjaxControlToolkit.dll** file from the folder where you had installed it earlier.

Figure 24.3 shows the dll file being selected.

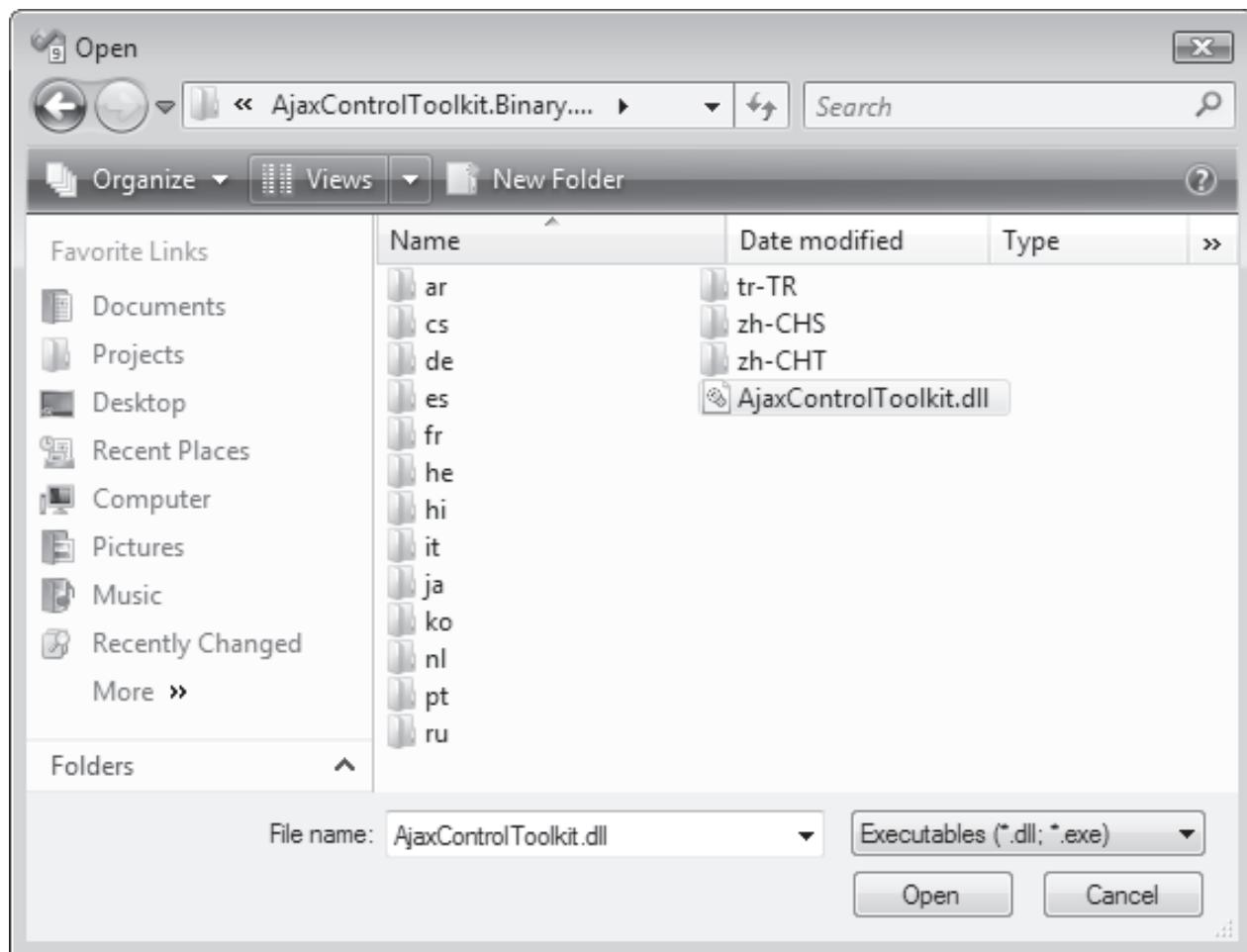


Figure 24.3: Selecting AjaxControlToolkit.dll file

5. After selecting the file, click **Open**. This will revert to the **Choose Toolbox Items** dialog box.

The controls from the AJAX Control Toolkit are displayed as shown in figure 24.4.

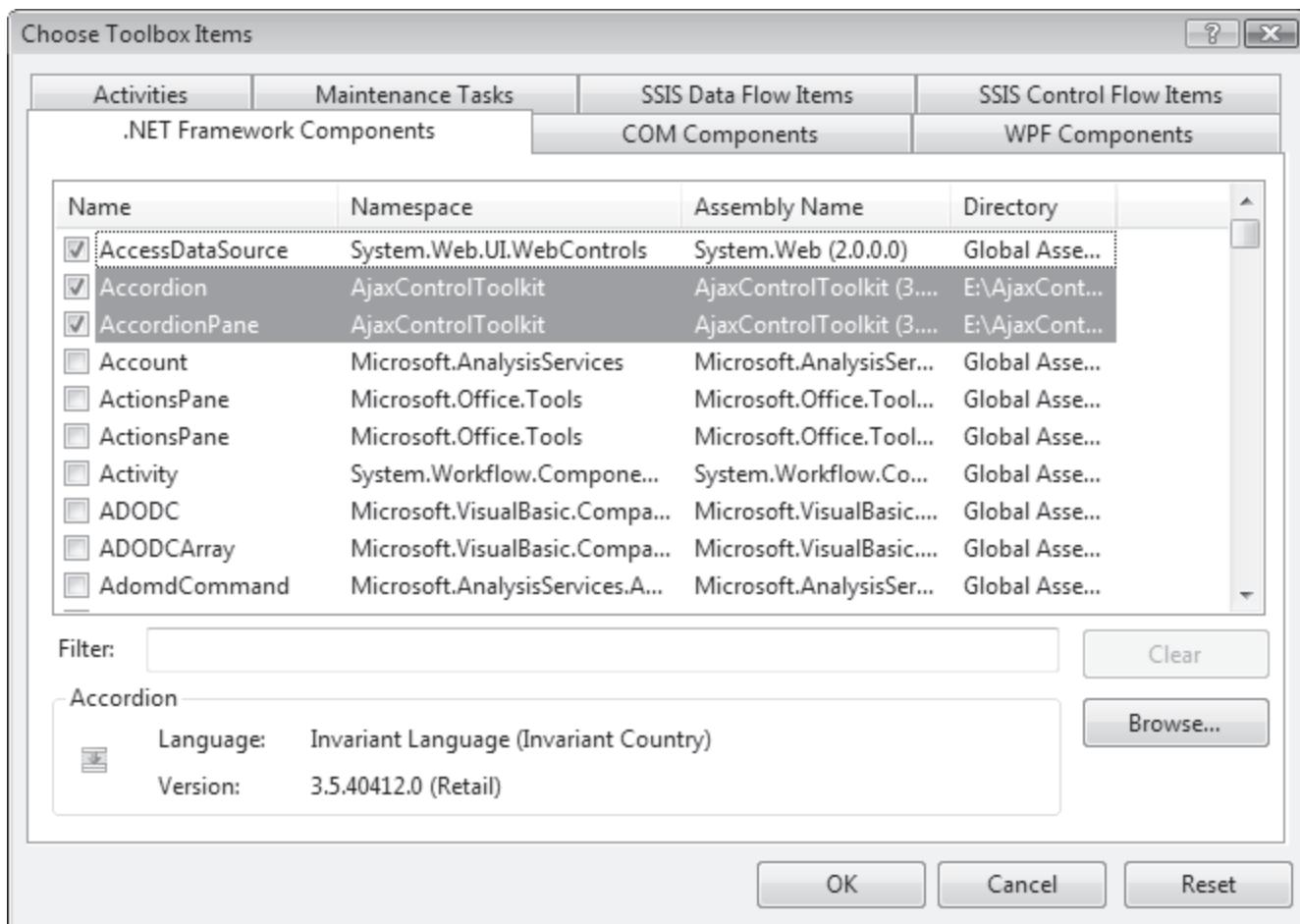


Figure 24.4: AJAX Control Toolkit Items

- Click **OK**. The controls from the AJAX Control Toolkit will be added to the Toolbox tab.

Figure 24.5 shows the AJAX Control Toolkit controls.

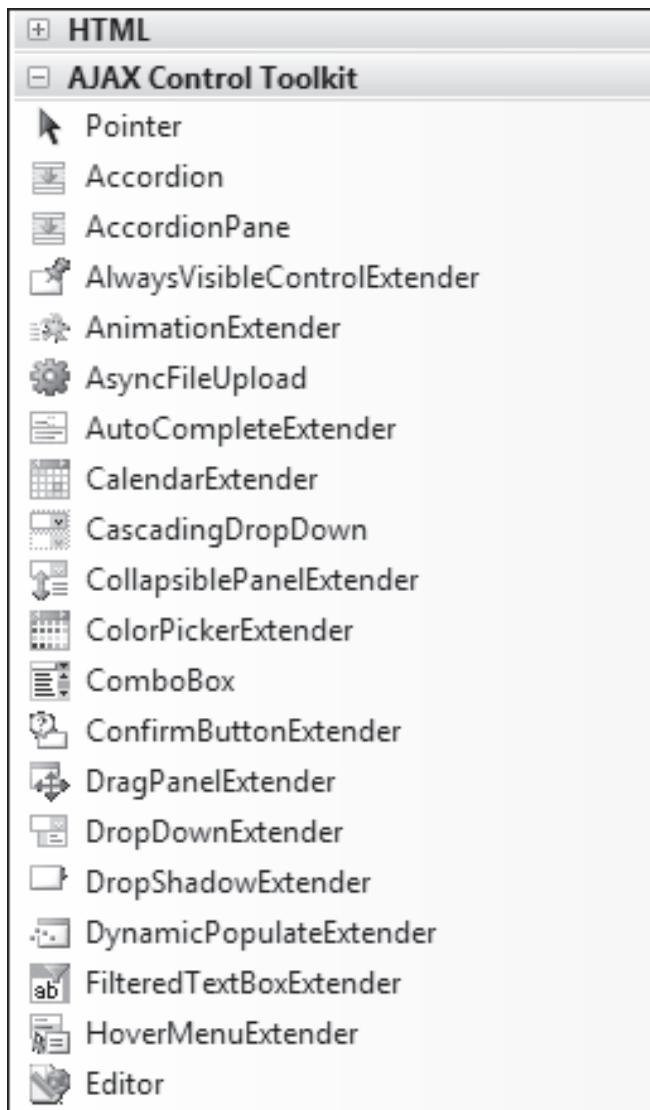


Figure 24.5: AJAX Control Toolkit Controls

7. Switch to Design view of **SurveyForm.aspx**.
8. Add the following markup to the form.

```
<head runat="server">
<title>Data Survey</title>
<style type="text/css">
#frmSurvey
{
```

```

 height: 699px;
 width: 601px;
 }

 .style1
 {
 font-family: "Bookman Old Style";
 font-weight: bold;
 text-decoration: underline;
 font-size: large;
 }

```

</style>

</head>

<body>

<form id="frmSurvey" runat="server">

<div>

<span class="style1">Data Survey by United Research Corporation<br /></span>

</div>

</form>

</body>

9. Switch to **Design view**.
10. Double-click the **ToolkitScriptManager** control from the AJAX Control Toolkit tab in the Toolbox.
11. Double-click the **UpdatePanel** control from the **AJAX Extensions** tab.
12. Set the properties of these controls as shown in table 24.1.

Server Control	Property	Value
ToolkitScriptManager	ID	toolkitscrpmgrData
UpdatePanel	ID	updpnldata

Table 24.1: Properties of the AJAX Extensions Controls

13. Add a **Panel** control outside the **UpdatePanel** area.
14. Add an **HTML Table** control to the Panel.
15. Add six **Label** controls and five **TextBox** controls to the **Table** control.

16. Configure their properties as shown in table 24.2.

Server Control	Property	Value
Panel	ID	pnlData
	BackColor	FFFFDF
Label	ID	lblName
	Text	Name:
TextBox	ID	txtName
Label	ID	lblEmail
	Text	Email Id:
TextBox	ID	txtEmail
Label	ID	lblDOB
	Text	Date of Birth:
TextBox	ID	txtDOB
Label	ID	lblCompany
	Text	Company:
TextBox	ID	txtCompany
Label	ID	lblNumEmployees
	Text	Number of Employees:
TextBox	ID	txtNumEmployees
Label	ID	lblData
	Text	Database Server Used:

Table 24.2: Properties of the Server Controls

17. Add a ComboBox control from the AJAX Control Toolkit tab.

This control was not available in ASP.NET 3.5, but because of the AJAX Control Toolkit, it is now available. This control is helpful when you want to an editable list of choices. If the user wants to type in some value that is not available in the list of choices, this control helps to do so.

17. Add a Label and a TextBox.

18. Set the properties of these controls as shown in table 24.3.

Server Control	Property	Value
ComboBox	Name	cboServer
	DropDownStyle	DropDownList
Label	ID	lblOutcome
	Text	Your Experience with the Server:
TextBox	ID	txtOutcome
Button	ID	btnOK

Server Control	Property	Value
	Text	Ok
	Align	Right

Table 24.3: Properties of the Toolkit and Server Controls

19. Arrange the controls on the user interface inside the Panel as shown in figure 24.6.

The figure shows a Windows application window with a title bar 'Untitled - Microsoft Windows Application'. Inside the window, there is a panel control containing seven text input fields (text boxes) and one button. The fields are labeled: 'Name', 'Email Id', 'Date of Birth', 'Company', 'Number of Employees', 'Database Server Used', and 'Your Experience with the Server'. The 'Database Server Used' field includes a dropdown arrow. Below the panel is an 'OK' button.

Figure 24.6: Controls within the Panel

20. Add appropriate validation controls and set their properties. Next, you need to add some extender controls to the server controls.  
 21. Hover the mouse near the textbox for Date of Birth.

You will see a smart tag next to the textbox for Date of Birth as shown in figure 24.7. This smart tag will enable you to add extenders to the textbox easily.

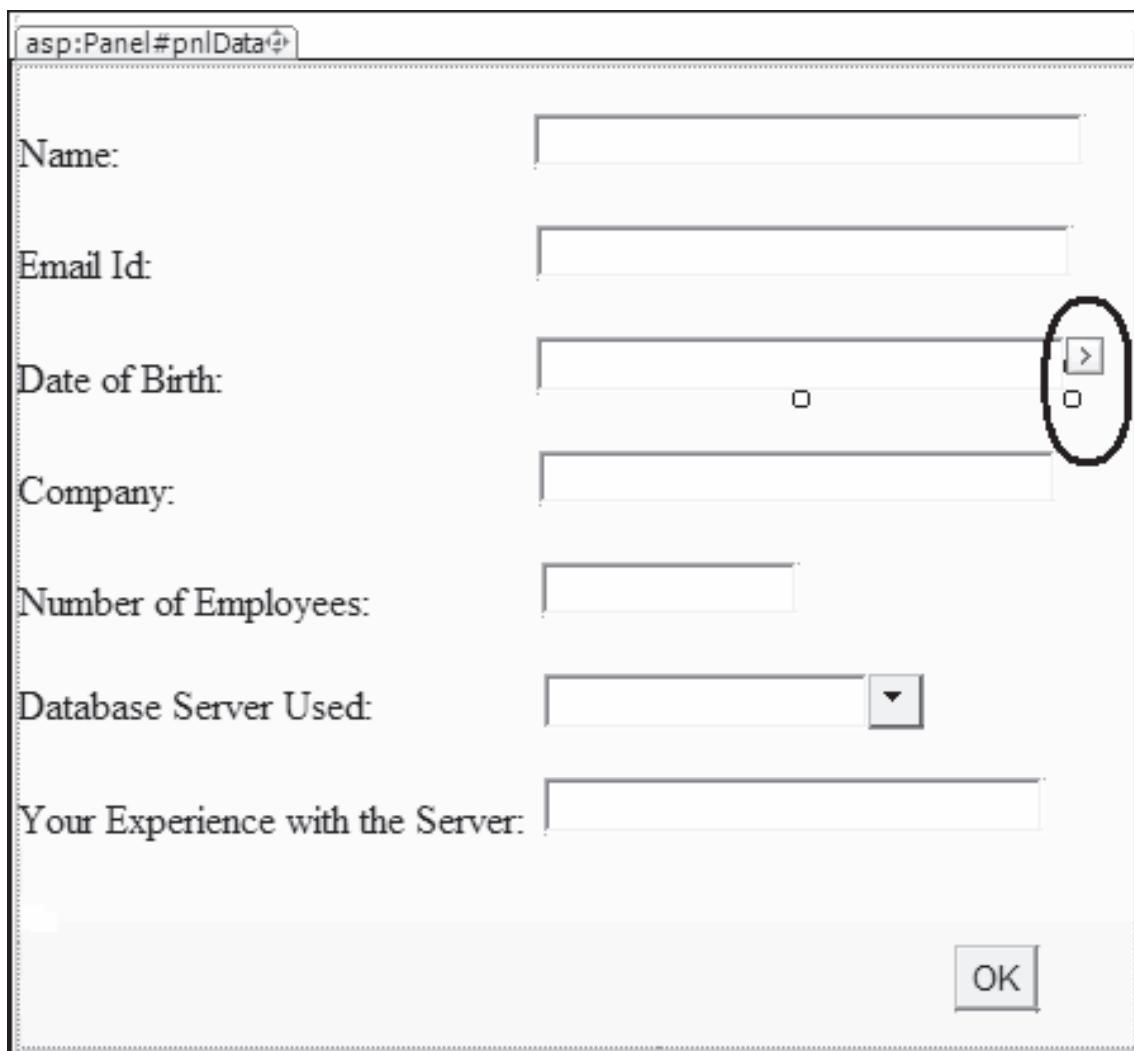
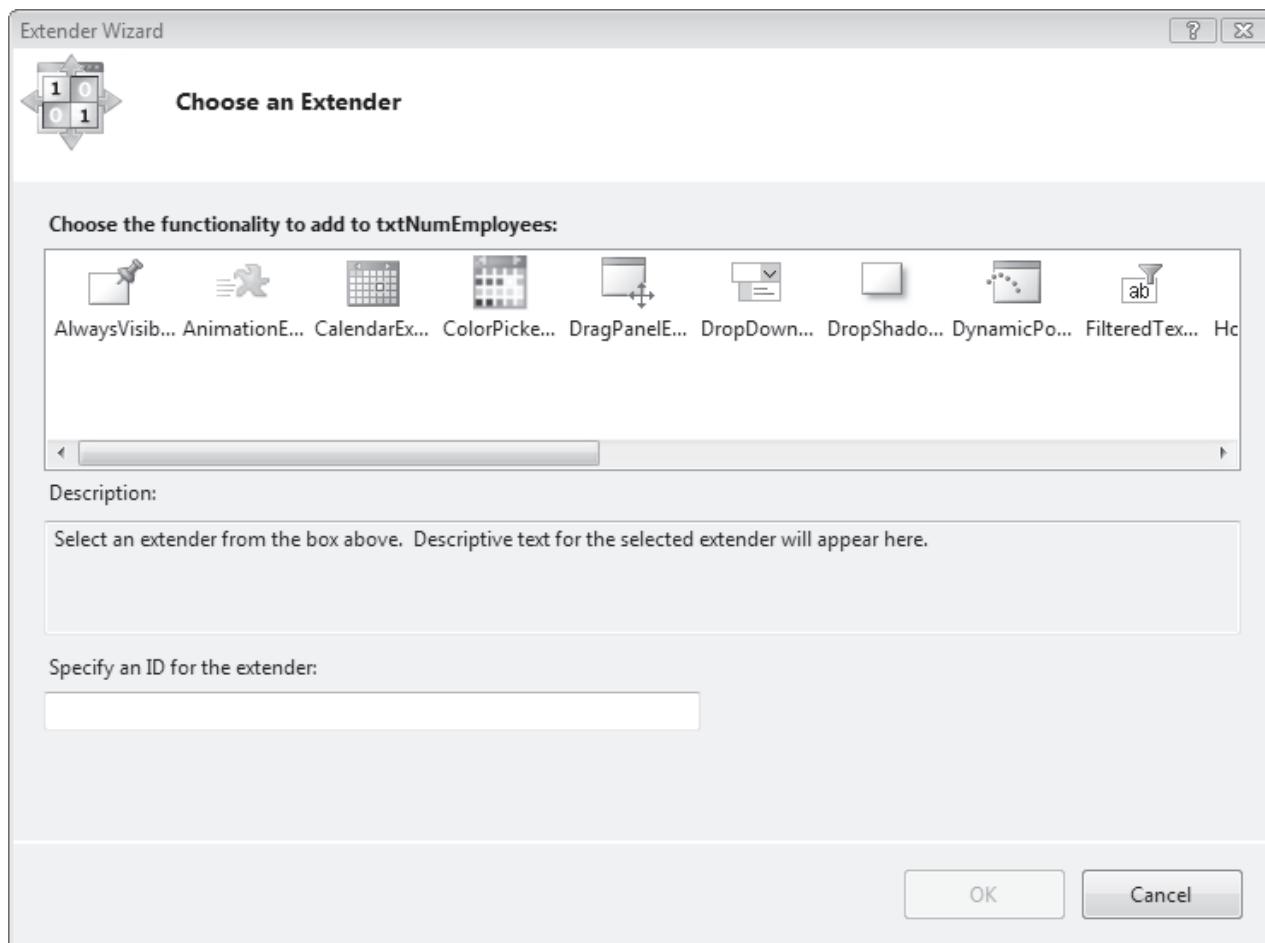


Figure 24.7: Smart tag for TextBox

22. Click the smart tag. This will display the **TextBox Tasks** smart tag menu.
23. Select **Add Extender** from the menu.

The **Extender Wizard** will be displayed as shown in figure 24.8.



**Figure 24.8: Extender Wizard**

24. Select **CalendarExtender**. This is a control in the AJAX Control Toolkit that enables you to display a drop-down calendar with a textbox.
25. Specify the name for this control as **txtDOB\_CalendarExtender**. When you see the automatically-generated markup for this control in Source view, it will appear as follows:

```
<asp:CalendarExtender ID="txtDOB_CalendarExtender" runat="server"
 Enabled="True" TargetControlID="txtDOB">
</asp:CalendarExtender>
```

You can customize various properties of this control like TargetControlID, the ID of the TextBox to extend with the calendar, Format, and PopupButtonID, the ID of a control to show the calendar popup when clicked. If this property is not set, the calendar will pop up when the textbox receives focus. You can also set the SelectedDate to indicate the date the Calendar extender is initialized with. For the current example, retain the simple properties as shown in the code.

26. Next, add an **AutoCompleteExtender** for the **txtNumEmployees** textbox. Rename the extender as **txtNumEmployees\_AutoCompleteExtender**.

An AutoCompleteExtender is an extender control that provides autocompletion suggestions when the user types part of a word or words. For example, if you have a textbox for displaying countries and the user types 'Ch', you can display autocompletion suggestions such as 'Chile', 'China', 'Chechen Republic', and so forth. Here, you will use autocompletion suggestion for number of employees such as '200-500', '501-1000', and so forth. The logic and the contents for autocompletion are usually stored in a Web service, through a Web method. These are then specified as attribute values in the extender control.

27. In the autogenerated markup for the AutoCompleteExtender, specify the Service Path as **ExtenderService.asmx** and Service Method as **GetNumbers** as follows:

```
<asp:AutoCompleteExtender ID="txtNumEmployees_AutoCompleteExtender"
 runat="server" DelimiterCharacters="" Enabled="True" ServicePath="ExtenderService.asmx"
 TargetControlID="txtNumEmployees" ServiceMethod="GetNumbers">
</asp:AutoCompleteExtender>
```

These properties indicate the name of the Web service and the name of the Web method that will contain the logic to populate the autocompletion list.

28. Similarly, create an AutoCompleteExtender for **txtOutcome** and specify the Service Path as **ExtenderService.asmx** and Service Method as **GetOutcome** as follows:

```
<asp:AutoCompleteExtender ID="txtOutcome_AutoCompleteExtender"
 runat="server" DelimiterCharacters="" Enabled="True"
 ServiceMethod="GetOutcome" ServicePath="ExtenderService.asmx"
 TargetControlID="TextBox2" >
</asp:AutoCompleteExtender>
```

29. Next, select **Website → Add New Item**. Select **Web service** and rename it as **ExtenderService** as shown in figure 24.9.

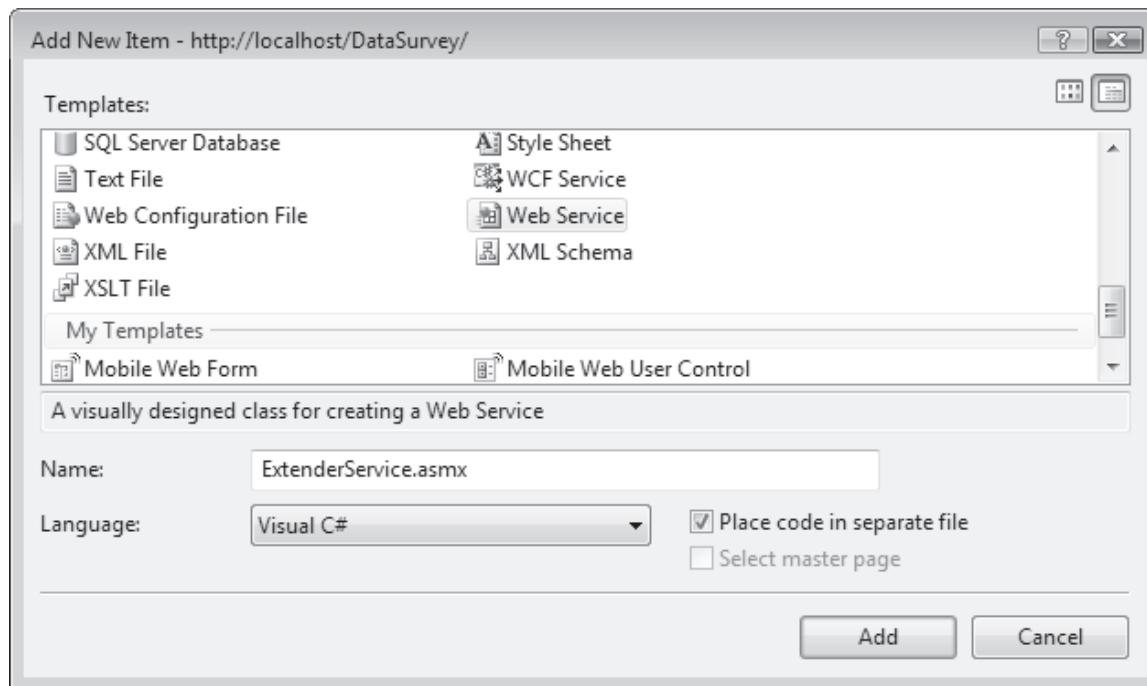


Figure 24.9: Adding Web service

30. To add the Web service to the application, click **Add**.
31. Within the **ExtenderService.asmx.cs** file, uncomment the following line:

```
[System.Web.Script.Services.ScriptService]
```

This line of code indicates that the service will be a script service.

A generic string collection will be used to populate the autocompletion logic. `List<string>` will be used to create the generic string collection.

30. To use the generic `List` class, add a new namespace `System.Collections.Generic` with a using directive to the namespaces section of the Web service as follows:

```
using System.Collections.Generic;
```

31. For creating the autocompletion logic for `txtNumEmployees` and `txtOutcome` respectively, add two Web methods as follows:

```
[WebMethod]
public string[] GetNumbers(string prefixText, int count)
{
 List<string> items = new List<string>();
```

```

 items.Add("<=100");
 items.Add("200-500");
 items.Add("501-1000");
 items.Add(">1000");
 return items.ToArray();
 }

 [WebMethod]
 public string[] GetOutcome(string prefixText, int count)
 {
 List<string> items = new List<string>();
 items.Add("Satisfied");
 items.Add("Dissatisfied");
 items.Add("Strongly Satisfied");
 items.Add("Strongly Dissatisfied");
 items.Add("Neutral");
 return items.ToArray();
 }
}

```

The two Web methods populate a generic string List collection with string values that will be displayed in the respective textboxes as autocomplete suggestions.

32. Switch to the codebehind file of **SurveyForm**.

33. Declare a private static string array in **SurveyForm.aspx.cs** as follows:

```
private static string[] serverList;
```

32. Create the following method that initializes a set of strings which can be added to the ComboBox control.

```

public string[] GetWordListText()
{
 if (serverList == null)
 {
 string[] tempWordListText = new string[]
 {

```

```

 "SQL Server 2008",
 "MySQL 5",
 "Oracle 10g",
 "Other"

 };

 Array.Sort(tempWordListText);

 serverList = tempWordListText;

}

return serverList;
}

```

This method creates and returns a string array containing various database server names.

33. Add the following code to Page \_ Load event handler.

```

protected void Page_Load(object sender, EventArgs e)
{
 if (!IsPostBack)
 {
 cboServer.DataSource = GetWordListText();
 cboServer.DataBind();
 }
}

```

This code will bind the ComboBox to the string array when the page loads.

34. Generate SelectedIndexChanged event for the ComboBox and add the following code:

```

protected void cboServer_SelectedIndexChanged(object sender, EventArgs e)
{
 cboServer.Text = cboServer.SelectedItem.Text;
}

```

This code will initialize the text to be displayed in the ComboBox when the user selects an item from the ComboBox.

35. Add a new Web Form to the application and rename it as **ThankYou.aspx**. Add two strings to the main area in Design View of the Web Form so that the page looks like figure 24.10.

*Thank you for completing this survey.*

Close the browser window to exit.

Figure 24.10: User Interface of ThankYou Page

36. Save, build, and execute the application. The output is shown in figure 24.11. As seen in the output, when you type part of a text in the textbox, an autocompletion list with suggestions is displayed.

The screenshot shows a Windows Internet Explorer window with the title "Data Survey - Windows Internet Explorer". The URL in the address bar is "http://localhost/DataSurvey/SurveyForm.aspx". The main content area displays a form titled "Data Survey by United Research Corporation". The form contains several input fields: "Name" (text box), "Email Id" (text box), "Date of Birth" (text box), "Company" (text box), "Number of Employees" (text box with an open dropdown menu showing "200", "<=100", "200-500", "501-1000", and ">1000"), "Database Server Used" (dropdown menu), and "Your Experience with the Server" (text box). At the bottom right of the form is an "OK" button.

Figure 24.11: Output

Similarly, you can test the autocomplete boxes for the other controls as well as the drop-down calendar on the Date of Birth textbox.

**Exercise 2**

Omega Coffee based in Canada is expanding their coffee chain to various outlets across the country. They now want to display a simple menu online through a Web application. The menu lets users choose a particular type of coffee and the cup size, following which the price of the coffee would be intimated to the customer. Assuming that you are a developer hired by Omega Coffee, you need to create this Web application. You will make use of Web services to retrieve the price information.

**Solution:**

1. Create an ASP.NET Web service named **OmegaService** as shown in figure 24.12.

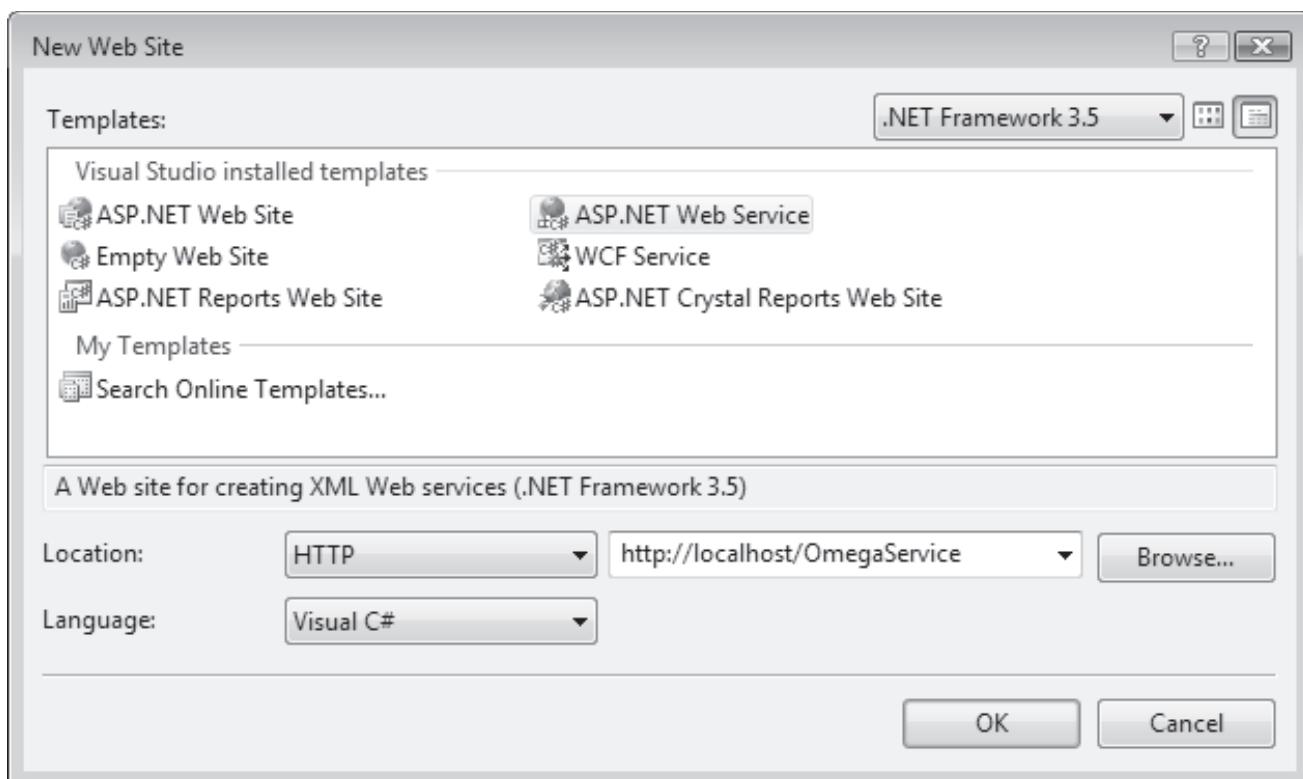


Figure 24.12: Creating a new Web Service

2. Rename the default **Service.asmx** to **CoffeePriceService.asmx**.
3. Add two methods to the **CoffeePriceService.asmx.cs** file as follows:

```
[WebMethod]
public decimal GetPrice(string type, string size)
{
 // suffix the price with m as it is a decimal value
```

```
decimal price = 0.0m;
switch (type)
{
 case "Espresso": price = 15 + GetSizePrice(size);
 break;

 case "Cappuccino": price = 25 + GetSizePrice(size);
 break;

 case "Turkish": price = 40 + GetSizePrice(size);
 break;

 case "Arabian": price = 60 + GetSizePrice(size);
 break;
}

return price;
}

public decimal GetSizePrice(string size)
{
 if (size == "Small")
 {
 return 4.25m;
 }
 else if (size == "Medium")
 {
 // suffix the price with m as it is a decimal value
 return 6.5m;
 }
 else
 {
 // suffix the price with m as it is a decimal value
 return 8.25m;
 }
}
```

4. Save and build the Web site.

5. Test the Web service in a browser by right-clicking **CoffeePriceService.asmx** in the **Solution Explorer** and selecting **View in Browser**. Figure 24.13 shows the Web service with the name of the Web method, **GetPrice()**, displayed on the browser.

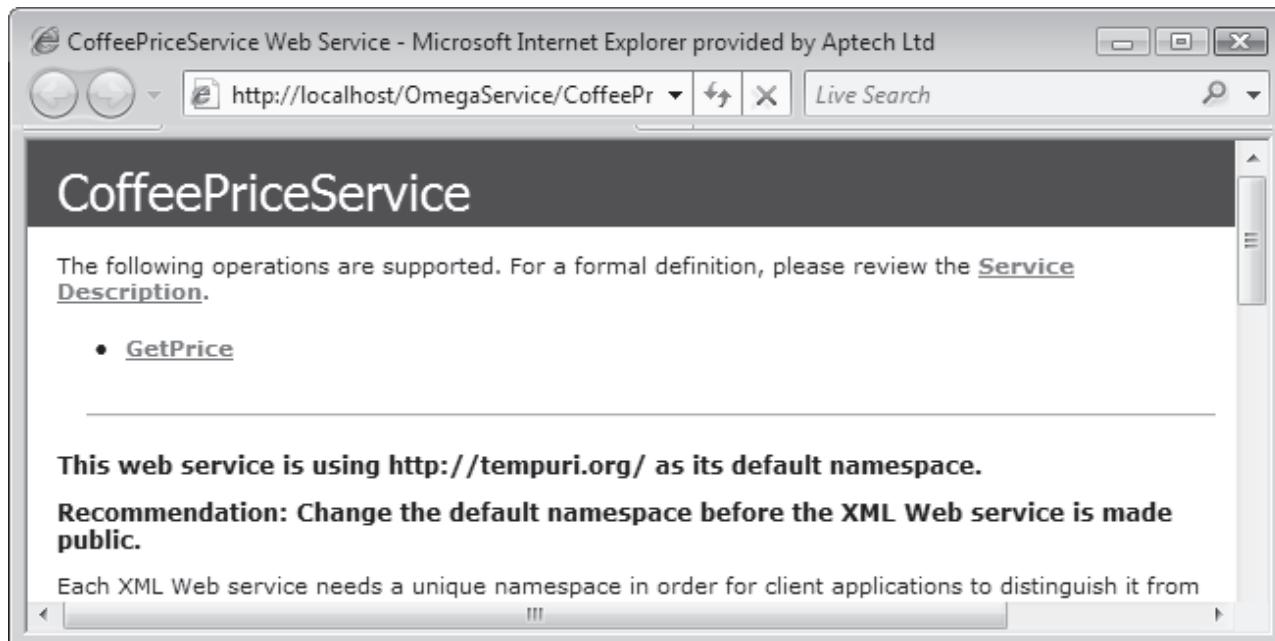


Figure 24.13: Web Service

6. Provide parameters to the Web method as shown in figure 24.14.

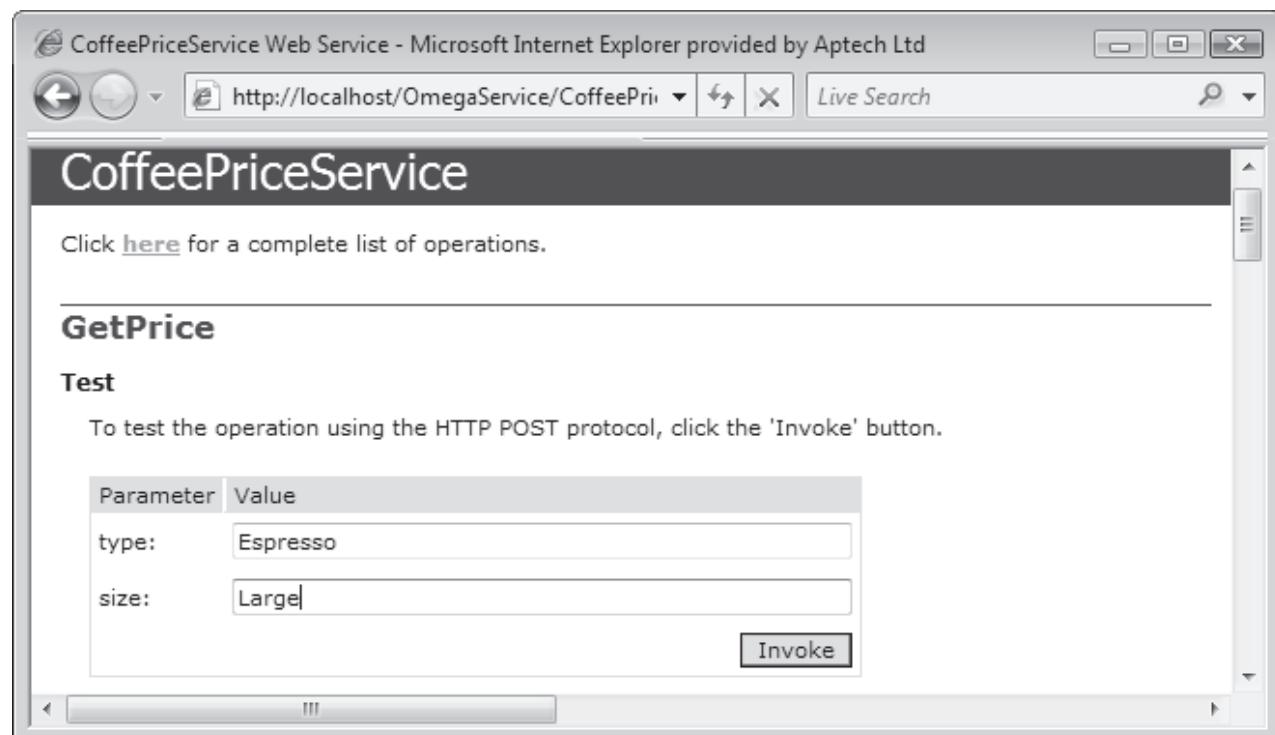


Figure 24.14: Providing Parameters to the Web Method

When you click Invoke, the output is displayed as shown in figure 24.15.

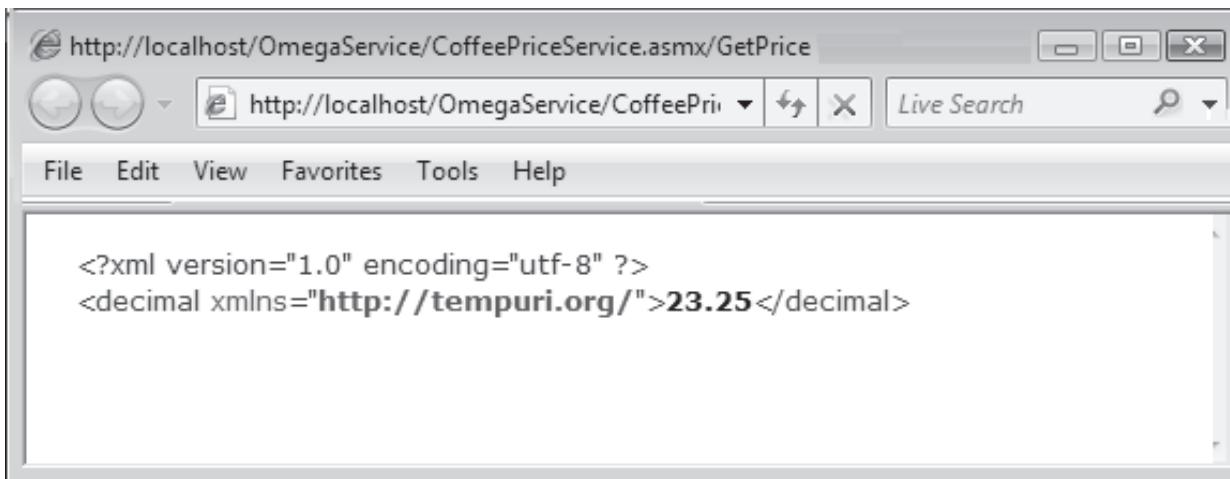


Figure 24.15: Web Service Output

Now you need to call the Web service in another Web application.

7. Create a new Web site named **OmegaCoffee** with **Location** set as **HTTP**.
8. Rename the default Web page to **MenuPage.aspx**.
9. Add markup to **MenuPage.aspx** as follows:

```
<head id="heading" runat="server">
 <title>Omega Coffee</title>
 <style type="text/css">
 .style1
 {
 text-decoration: underline;
 font-size:large;
 text-align:center;
 }
 </style>
</head>
<body>
<form id="form1" runat="server">
<div class="style2">
 Omega Coffee Chain

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

```

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

```

10. Add an HTML table to the form.
11. Add three Label and two DropDownList controls to the form. Also, add an Image and Button control.
12. Configure their properties as shown in table 24.4.

Server Control	Property	Value
Label	ID	lblType
	Text	Enter the Coffee Type:
DropDownList	ID	ddlType
	Items	Espresso
		Cappuccino
		Turkish
		Arabian
Image	ID	imgCoffee
	ImageUrl	~/Images/coffee.jpg
Label	ID	lblSize
	Text	Enter Size:
DropDownList	ID	ddlSize
	Items	Small
		Medium
		Large
Button	ID	btnDone
	Text	Done
Label	ID	lblPrice
	Text	

Table 24.4: Properties of Controls on MenuPage Web Form

13. Arrange the controls as shown in figure 24.16.

**Omega Coffee Chain**

Enter the Coffee Type:	Espresso ▾
Enter Size:	Small ▾
Done	

*lblPrice*

Figure 24.16: User Interface of MenuPage Web Form

14. Declare two variables in **MenuPage.aspx.cs** as follows:

```
// Declare variables
string type, size;
```

These variables will be used to store the type and size of the coffee chosen by the user.

15. Add the code to `Page_Load` event handler in **MenuPage.aspx.cs** as follows:

```
/// <summary>
/// Sets the type and size values
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_Load(object sender, EventArgs e)
{
```

```
if (!IsPostBack)
{
 type = ddlType.Items[0].Text.ToString();
 size = ddlSize.Items[0].Text;
}
else
{
 // Set the defaults so that if the user does not select
 // any option, the default value would be read

 type = ddlType.SelectedItem.Text.ToString();
 size = ddlSize.SelectedItem.Text.ToString();
}
```

The code creates two variables type and size, and assigns the value of type and size respectively chosen by the user. If the user has not chosen any value, the default value in the drop-down controls is taken.

To consume the Web service created earlier, you first need to add a reference to the Web service namespace.

1. Right-click the project name, **OmegaCoffee**, in **Solution Explorer**.
2. Select **Add Web Reference** from the context menu.

This displays the **Add Web Reference** dialog box as shown in figure 24.17.

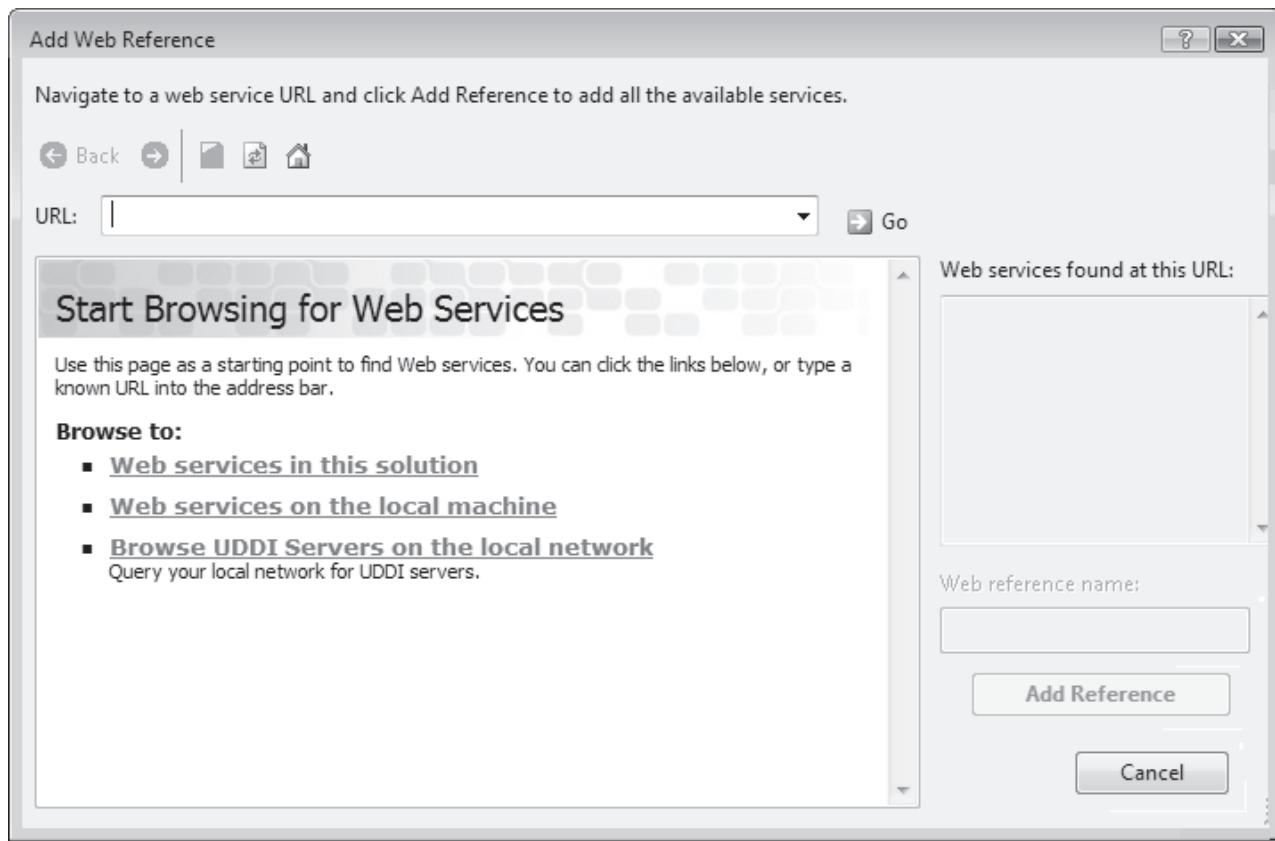


Figure 24.17: Add Web Reference Dialog Box

3. Click '**Web services in this solution**'. **CoffeePriceService** is displayed in the **Add Web Reference** dialog box.

4. Type the service reference name as **OmegaServiceReference** as shown in figure 24.18.

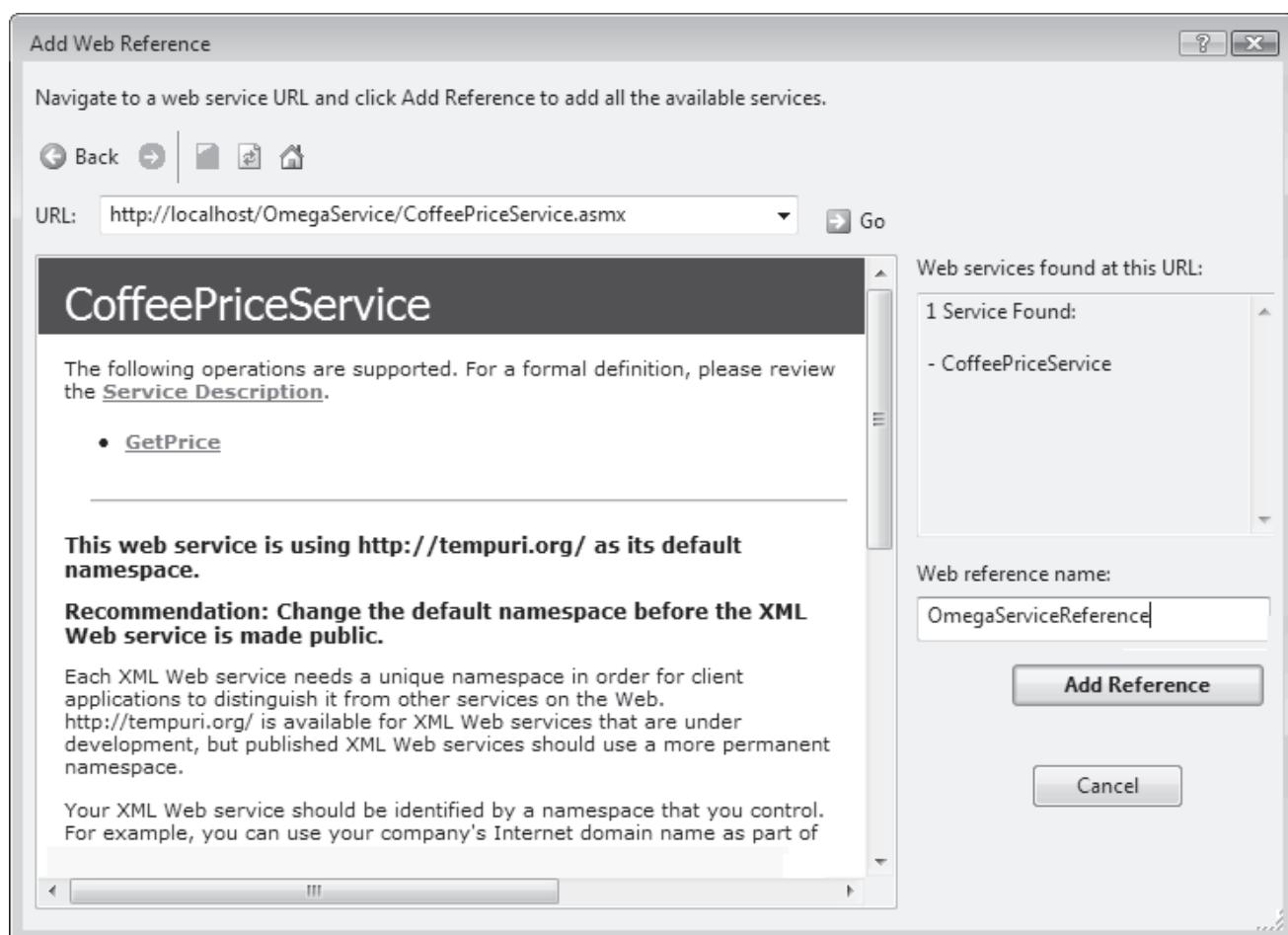


Figure 24.18: Adding Web Reference

5. Click **Add Reference** to add the service reference.

6. Figure 24.19 shows the files that are automatically created as a result of adding the service reference. The disco file is the discovery file and the wsdl file is the WSDL document for the Web service.

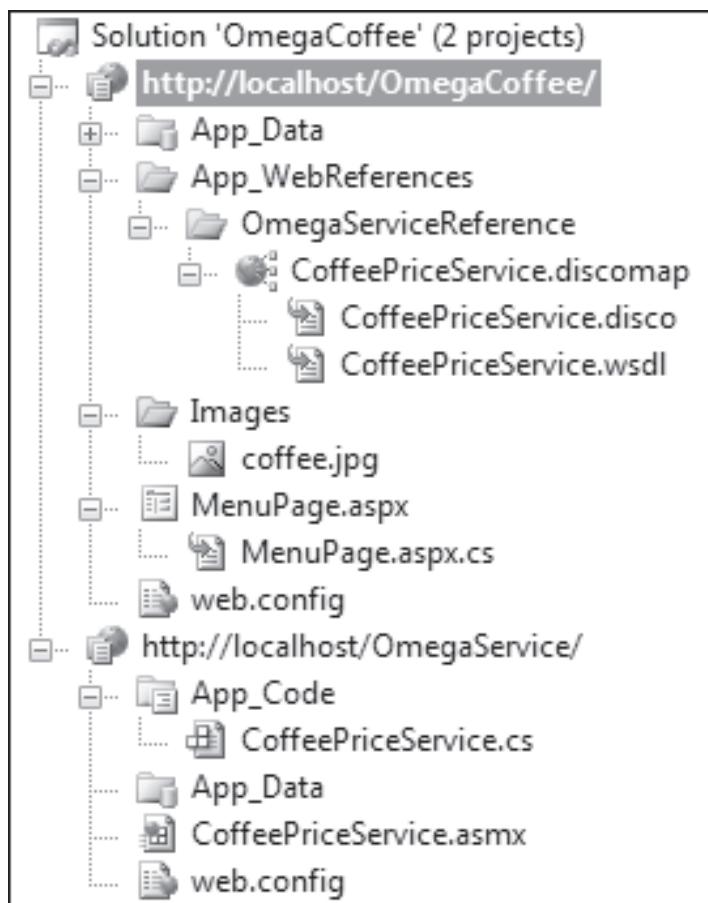


Figure 24.19: Service Reference and the Autogenerated Files

6. Generate Click event for the button, **Done**, and add the following code to the event handler:

```

/// <summary>
/// Retrieves the price of coffee based on size and type using Web
/// service instance
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void btnDone_Click(object sender, EventArgs e)
{

```

```
CoffeePriceService objCoffeePrice = new CoffeePriceService();
lblPrice.Text = "The Price of the Coffee is: $";
lblPrice.Text += objCoffeePrice.GetPrice(type,size).ToString ();
}
```

This code creates a new instance of the Web service, **CoffeePriceService**, and calls the **GetPrice()** Web method with the parameters type and size. The result of the method call is converted to string format and assigned to the Text property of the label, **lblPrice**.

7. Save, build, and execute the application.

The output depicted in figure 24.20 shows the coffee type as Turkish and size as Medium and the resulting price for this set.

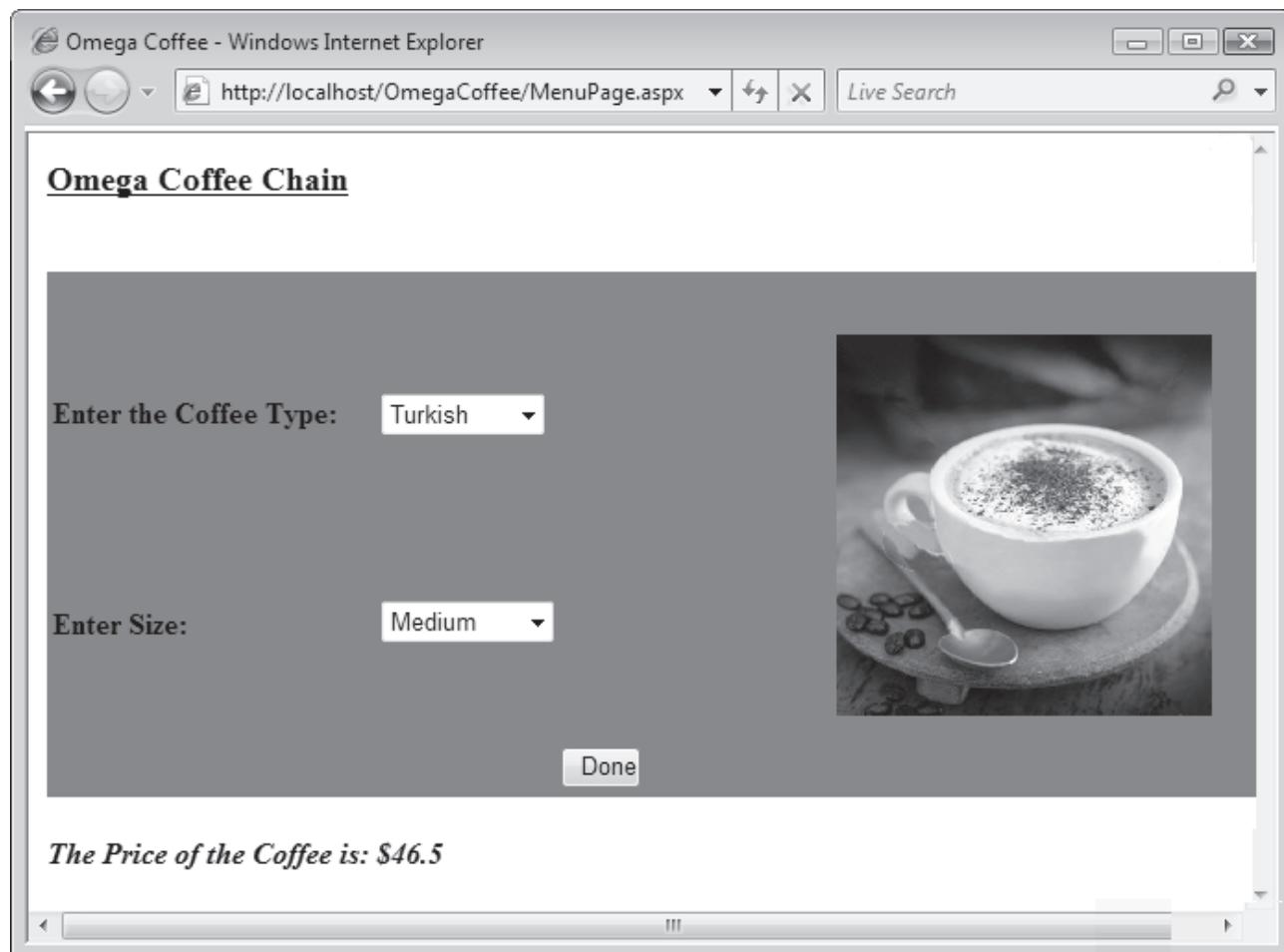


Figure 24.20: Displaying Price Based on Type and Size Selected

## 24.2 Part II - 30 Minutes

1. Modify the DataSurvey Web application shown in Part I and add a Timer control from AJAX Extensions tab in the Toolbox. The Web Form should have an additional Label displaying the current time including seconds. The time value in the Label should refresh every 1 second.

**Hints:**

Add a Timer control inside the UpdatePanel

Set the Interval property to 1000

Handle its Tick event to update the latest time in the Label control

2. Create a Web application having a simple login Web Form. It should make use of AJAX Toolkit controls such as AutoCompleteExtender and ConfirmButtonExtender.

**Hints:**

Using the smart tag with the TextBox control, you can add the extenders

Configure their properties

## 24.3 Do It Yourself

**Harris Associates** is a legal firm in **Chicago** city. They often undertake prosecution of high profile criminal cases. The firm has recently expanded their operations overseas with local representatives in each region. They have a centralized online billing system. This system has a unique condition - currency values must be provided in United States (US) dollars only. Hence, they have approached you to seek your expertise as a Web application developer. You must create a Web service-based application for a global currency converter.

# Answers to Knowledge Checks

## Module 1

### Knowledge Check 1

1. C
2. A

### Knowledge Check 2

1. D
2. A

### Knowledge Check 3

1. D
2. C

## Module 2

### Knowledge Check 1

1. C
2. D

### Knowledge Check 2

1. D

### Knowledge Check 3

1. C

## Module 3

### Knowledge Check 1

1. C
2. D

# Answers to Knowledge Checks

## Answers to Knowledge Checks

---

### Knowledge Check 2

1. A
2. B
3. B

### Knowledge Check 3

1. C
2. D

Answers

## Module 4

### Knowledge Check 1

1. C
2. B

### Knowledge Check 2

1. D
2. A

### Knowledge Check 3

1. B
2. A

## Module 5

### Knowledge Check 1

1. C
2. B

### Knowledge Check 2

1. A
2. A
3. B

# **Answers to Knowledge Checks**

---

## **Answers to Knowledge Checks**

---

### **Module 6**

#### **Knowledge Check 1**

1. C
2. B

#### **Knowledge Check 2**

1. B
2. B

### **Module 7**

#### **Knowledge Check 1**

1. D
2. B

#### **Knowledge Check 2**

1. B
2. A

#### **Knowledge Check 3**

1. A
2. D

### **Module 8**

#### **Knowledge Check 1**

1. B
2. B

#### **Knowledge Check 2**

1. B
2. D

Answers

# Answers to Knowledge Checks

---

## Answers to Knowledge Checks

---

### Knowledge Check 3

1. B

## Module 9

### Knowledge Check 1

1. C
2. D

### Knowledge Check 2

1. C
2. A

Answers

## Module 10

### Knowledge Check 1

1. A
2. A

### Knowledge Check 2

1. C
2. C

### Knowledge Check 3

1. A

### Knowledge Check 4

1. D

## Module 11

### Knowledge Check 1

1. D

### Knowledge Check 2

1. D

## **Answers to Knowledge Checks**

---

### **Answers to Knowledge Checks**

---

#### **Knowledge Check 3**

1. D
2. B

## **Module 12**

#### **Knowledge Check 1**

1. A

#### **Knowledge Check 2**

1. A

#### **Knowledge Check 3**

1. A
2. A

## **Module 13**

#### **Knowledge Check 1**

1. C
2. A, D
3. A, B, D
4. B, C, D
5. B, C, D
6. D
7. A

## **Module 15**

#### **Knowledge Check 1**

1. A, C
2. C, D
3. B, C, D

Answers

## Answers to Knowledge Checks

---

### Answers to Knowledge Checks

---

4. A, D

5. A, B

6. A, B

## Module 17

### Knowledge Check 1

1. A, C, E

2. A, C, D

3. C

4. A

5. A-5, B-4, C-1, D-2, E-3

Answers

## Module 19

### Knowledge Check 1

1. C, E

2. A-3, B-5, C-1, D-4, E-2

3. A, C, E

4. A, D, E

5. A-2, B-1, C-4, D-3

6. A, B, D

7. C

## Module 21

### Knowledge Check 1

1. A

2. A

3. B

4. A, B, C

## **Answers to Knowledge Checks**

---

### **Answers to Knowledge Checks**

---

5. A

6. B

## **Module 23**

### **Knowledge Check 1**

1. A, B, C

2. A

3. B

4. A

5. B

6. B

**Answers**